Michal Brandejs

Mikroprocesory Intel Pentium

Fakulta informatiky, Masarykova univerzita, Brno

Michal Brandejs

Mikroprocesory Intel – Pentium

Tento text byl vydán v nakladatelství Grada v roce 1994. Po vypršení platnosti nakladatelské smlouvy byl text autorem jako další vydání elektronicky zveřejněn dne 1. 9. 2010.

Obsah

Seznam obrázků

1. Řada procesorů Intel 86

V roce 1968 pánové Robert Noyce a Gordon Moore založili firmu **Intel** (*Int*egrated *El*ectronics), která o dva roky později představila první mikroprocesor označený **4004**. Měl 4bitovou strukturu, byl sestaven z 2 250 tranzistorů MOS a pracoval rychlostí 60 000 operací za sekundu. Adresoval 1 280 půlslabik dat a 4 KB instrukcí. Zapojoval se do kapesních kalkulátorů.

V roce 1972 firma dává na trh první 8bitový mikroprocesor **8008** sestavený z 3 300 tranzistorů MOS a pracující rychlostí 30 000 operací za sekundu. K procesoru bylo možné připojit paměť kapacity 16 KB.

Mikroprocesor **8080** z roku 1974 byl konstruován pro široké uplatnění. Je to 8bitový procesor, na čipu je 4 500 NMOS tranzistorů, pracuje s rychlostí 200 000 operací za sekundu. Instrukční repertoár je kompatibilní s typem 8008 a je rozšířen o 30 nových instrukcí. Adresovací kapacita je 64 KB. Později byly na trh uvedeny mírně vylepšené modifikace 8080: 8080A a 8085A.

První 16bitový mikroprocesor Intel **8086** je z roku 1978. Jde o první člen řady 86. Jeho instrukční repertoár je částečně „zdola kompatibilní“, protože programy pro 8080 po rekompilaci bylo možné provozovat i na tomto procesoru. Adresovací kapacita procesoru je 1 MB paměti. K procesoru 8086 byl vyprojektován pomocný specializovaný procesor pro určitý typ výpočtů, nazývaný koprocesor. Nejznámějším je koprocesor pro matematické operace v pohyblivé řádové čárce Intel 8087.

Z roku 1979 pochází Intel **8088**, což je modifikace 8086. Liší se tím, že má vnější 8bitovou strukturu, a tím jej lze zapojovat do 8bitového prostředí. Firma IBM ho převážně používala v počítačích IBM PC a IBM PC/XT. Procesory Intel 80186 a 80188 jsou rozšířením procesorů 8086 a 8088 o dva kanály rychlého přístupu k paměti (DMA), tři programovatelné časovače, programovatelný řadič přerušení, generátor časovacích impulsů a o několik málo instrukcí.

Mikroprocesor Intel **80286** byl dán na trh v roce 1983. O rok později firma IBM představuje počítač IBM PC/AT osazený tímto procesorem. Procesor 80286 má dva pracovní režimy: *reálný* a *chráněný*. V reálném režimu je procesor vlastně jenom rychlejší 8086 (1 MB paměti, instrukční repertoár jenom o několik instrukcí bohatší než 8086). V chráněném režimu jsou programátorovi dostupné všechny vlastnosti 80286. Tento režim není slučitelný s 8086, procesor adresuje až 16 MB paměti a rovněž poskytuje podporu pro virtuální paměť do 1 GB. Pro spolupráci s matema-

tickým koprocesorem Intel287 je vybaven prostředky umožňujícími současnou práci procesoru a koprocesoru. Dále procesor podporuje sdílení více programů v operační paměti včetně ochrany paměťových segmentů úrovněmi oprávnění.

Zrod 32bitového mikroprocesoru **Intel386** se datuje rokem 1985. Má tři pracovní režimy: *reálný*, *chráněný* a *virtuální 8086*. V reálném režimu je procesor, stejně jako 80286, slučitelný s 8086, i když můžeme používat 32bitové prostředí. V chráněném režimu procesor pracuje jako plně 32bitový a poskytuje všechny své vlastnosti. Tento režim opět není slučitelný s 8086. Dostupná kapacita fyzické paměti je 4 GB a virtuální paměti 64 TB. Režim virtuální 8086 lze zapnout v rámci chráněného režimu procesoru pro konkrétní úlohy. Úloha, která je zpracovávána jako „virtuální 8086“, se v prostředí chráněného režimu chová tak, jako by byla řešena procesorem 8086 s většinou jeho vlastností. Procesor Intel386 spolupracuje buď s koprocesorem Intel387, nebo Intel287. Procesor Intel386 SX je 32bitový procesor pro 16bitové vnější prostředí.

Rok 1989 přinesl typ lišící se od předchozích pokročilejší výrobní technologií. Procesor **Intel486** (nebo stejný název je Intel486 DX) je programově slučitelný s procesorem Intel386. Liší se pouze výkonem (je 3 až 5krát rychlejší) a na čipu má integrovánu rovněž jednotku operací v pohyblivé řádové čárce (FPU) a rychlou vyrovnávací paměť (Cache). Varianta Intel486 SX nemá na čipu funkční FPU. Varianta Intel486 DX2 pracuje vnitřně na dvojnásobné frekvenci proti frekvenci okolí procesoru (např. udávaná frekvence 66 MHz zpravidla znamená, že na této frekvenci procesor pracuje uvnitř, okolí pracuje na frekvenci 33 MHz).

Procesor Pentium je od roku 1993 novým procesorem firmy Intel z řady procesorů 86. Je 100% binárně kompatibilní s předchozími typy procesorů 8086/88, 80286, Intel386 DX, Intel386 SX, Intel486 DX, Intel486 SX a Intel486 DX2. Tato kniha je převážně určena jako popis procesoru Pentium. Díky kompatibilitě zdola bylo možné zahrnout do textu i poznámky k předchůdcům ve výše uvedené řadě.

2. Principy programování procesoru

V této kapitole se čtenáři podávají základní principy programování procesoru z pohledu programátora v asembleru. Samostatně se popisují tyto principy procesoru:

- Typy dat
- Organizace paměti
- Registry
- Zásobník
- Adresovací techniky
- Přerušení a výjimky
- Ovládání vstupů a výstupů

2.1 Typy dat

Elementární jednotkou množství informace je **1 bit** (zkratkou označujeme malým písmenem **b**). Nejmenší adresovatelnou jednotkou, tj. nejmenší množství informace, které se ukládá do paměti, je **1 slabika** (zkratkou označujeme velkým písmenem **B** z anglického Byte). Slabika je tvořena 8 bity. Každá slabika má v paměti svoji *adresu*, proto můžeme s každou ze slabik pracovat.

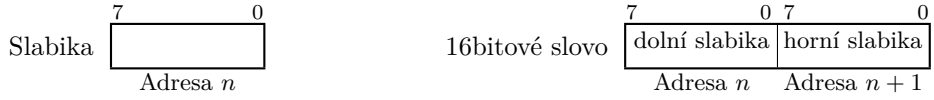
Bity ve slabice číslujeme. *Bit nejnižšího řádu* (tj. dvojkového[1] řádu 2^0) označujeme číslem 0 a *bit nejvyššího řádu* (tj. 2^7) označujeme číslem 7. Bit nejnižšího řádu se v originální literatuře označuje Least Significant Bit (LSB) a bit nejvyššího řádu jako Most Significant Bit (MSB). Kreslíme-li schematicky obsah paměťového objektu, kreslíme bit 0 vpravo a bit nejvyššího řádu vlevo.

Kapacita adresovacího prostoru procesoru se udává ve slabikách. Používají se tyto násobné jednotky (uveďme je ve spojení se slabikami):

[1] Nebo též binárního.

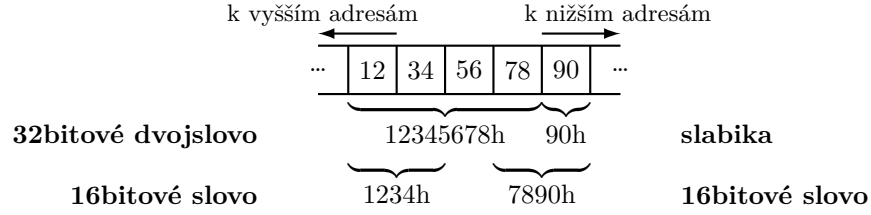
Řád	*Jednotka*
2^{10}=1 024	1 KB[2]
2^{20}=1 048 576	1 MB
2^{30}=1 073 741 824	1 GB
2^{40}=1 099 511 627 776	1 TB

Procesory Intel dále pracují se spojením dvou slabik na sousedních adresách do jednotky nazývané **slovo** (anglicky Word, někdy proto též zkratka W). Bity slova jsou číslovány 0 až 15. Bity 0 až 7 tvoří tzv. *dolní slabiku* a bity 8 až 15 tvoří *horní slabiku*. Dolní slabika je uložena na nižší adrese. Adresa dolní slabiky je také adresou slova. Adresa horní slabiky se používá jenom tehdy, pokud potřebujeme zvlášť adresovat horní slabiku.



Obr. 2.1 Formát slabiky a 16bitového slova v paměti

Dvojslovo (anglicky Doubleword, z toho zkratka D) je tvořeno stejnými principy jako slovo, avšak ze 4 slabik. Bity jsou číslovány 0 až 31. Slovo tvořené bity 0 až 15 je dolní slovo, slovo tvořené bity 16 až 31 je horní slovo. Adresa nejnižší slabiky je adresou dvojslova. **Čtyřslovo** (z anglického Quadword) je tvořeno osmi slabikami na sousedních adresách stejnými principy.



Obr. 2.2 Formát 32bitového dvojslova a 16bitového slova v paměti

Připomeňme, že adresy slov nemusejí být **zarovnány** (v originále „aligned“) na sudé adresy, tj. že adresa slova nemusí být dělitelná dvěma. Dvouslova nemusejí být zarovnány na adresy dělitelné čtyřmi a čtyřslova na adresy dělitelné osmi. Tím se zvyšuje flexibilita datových typů a zjednodušuje práce při míchání několika typů

[2]Ve zkratce KB píši „K“ velké proto, abych odlišil k = 10^3 = 1000 a K = 2^{10} = 1024.

dohromady. Na druhou stranu zarovnávání datových typů na adresy dělitelné jejich délkou vede ke zrychlení práce procesoru. To proto, že procesor k paměti nepřistupuje po slabikách, nýbrž po větších jednotkách. Pokud např. zpracovávané dvojslovo není zarovnáno, musí pro něj procesor do paměti nadvakrát.

Předchozími pojmy jsme si vymezili rámce pro ukládání objektů do paměti, přičemž základními typy jsou slabika, slovo a dvojslovo. Následující pojmy se týkají interpretace obsahu objektů v paměti.

- **Celé číslo se znaménkem** (Integer): celé číslo se znaménkem je hodnota ve dvojkovém doplňkovém kódu v 32bitovém dvojslově, 16bitovém slově nebo 8bitové slabice. Znaménkový bit je bitem nejvyššího řádu. Pro záporné číslo má hodnotu 1, nulový je pro čísla kladná a nulu. Rozsah zobrazení pro slabiku je -128 až 127. Pro slovo $-32\,768$ až $32\,767$. Pro dvojslovo je -2^{31} až $2^{31}-1$.
- **Celé číslo bez znaménka** (Ordinal): celé číslo bez znaménka bývá uloženo ve dvojslově, slově nebo slabice. Rozsah zobrazení pro slabiku je 0 až 255, pro slovo 0 až 65 535 a pro dvojslovo 0 až $2^{32}-1$. Typ se též označuje jako neznaménkové celé číslo (Unsigned Integer).
- **Dvojkově kódovaná desítková celá čísla** (BCD Integer – Binary Coded Decimal Integer): v kódu dvojkově kódovaných desítkových číslic (zkráceně označován BCD) je jedna desítková číslice (0 až 9) uložena ve 4 bitech. Podrobnosti o BCD kódu jsou uvedeny na str. 275.
- **Blízký ukazatel** (Near Pointer): je to efektivní adresa. Je tvořena offsetem v rámci segmentu. Podrobnosti viz sekce 2.2.
- **Vzdálený ukazatel** (Far Pointer): je to logická adresa složená ze segmentu a offsetu. Podrobnosti viz sekce 2.2.
- **Bitové pole** (Bit Field): je to posloupnost bitů. Může začínat na libovolné pozici libovolné slabiky a může být dlouhá až 32 bitů.
- **Bitový řetězec** (Bit String): je to posloupnost bitů délky až $2^{32}-1$ bitů začínající na libovolné pozici libovolné slabiky.
- **Řetězec slabik** (Byte String): je posloupnost slabik (i slov nebo dvojslov). Řetězec může začínat na libovolné slabice může být dlouhý až $2^{32}-1$ slabik (4 GB).
- **Číslo v pohyblivé řádové čárce** (Floating-Point): používá jej jednotka operací v pohyblivé řádové čárce (FPU). Podrobnosti viz v kapitole 3.

Poznámky k předchozím procesorům

Dvojslovo a 32bitové zpracování je vlastní teprve procesorům od Intel386, stejně tak i datové typy bitové pole a bitový řetězec.

2.2 Organizace paměti

Paměť připojená na sběrnici procesoru se nazývá **fyzická paměť**. Je organizována jako posloupnost slabik. Každé slabice je přiřazena **fyzická adresa**, která je v intervalu 0 až $2^{32} - 1$ (4 GB).

Technické prostředky správy paměti odstiňují programátora od fyzických adres a poskytují mu tzv. **virtuální paměť**. Správa paměti dává programátorovi k dispozici segmentování a stránkování paměti. **Segmentování paměti** je mechanismus poskytující násobný a nezávislý přístup k adresovému prostoru. **Stránkování paměti** podporuje vytváření rozsáhlejšího adresového prostoru, než je dostupná kapacita fyzické paměti s použitím vnější diskové paměti. Mechanismy se mohou používat oba, nebo jenom některý z nich.

Odkazuje-li se program na paměť, použije tzv. **logickou adresu**. Tato se segmentováním překládá na nesegmentovou tzv. **lineární adresu**. Stránkování může lineární adresu dále přeložit na **fyzickou adresu**. Není-li stránkování zapnuto, je fyzická adresa totožná s lineární.

Paměť můžeme používat buď jako nestrukturovaný adresový prostor, podobně jako bychom přímo používali fyzické adresy, nebo lze paměť rozdělit do vzájemně nezávislých prostorů nazývaných **segmenty**. Různé segmenty se vytvářejí pro uložení instrukcí prováděného programu, pro jejich data nebo zásobník. Jeden proces může mít až 16 383 segmentů různých velikostí a typů. Jeden z důležitých významů existence segmentů je zvýšení spolehlivosti operačního systému. Např. zásobníky se umísťují do speciálních segmentů. V případě, že by do zásobníku bylo uloženo více položek, než je kapacita segmentu, detekuje se pokus o překročení hranic segmentu a nemůže tedy dojít k přepisu instrukcí nebo dat jiného nebo i vlastního procesu.

Logická adresa se skládá ze dvou složek, ze **selektoru** a **offsetu**. Selektor ukazuje na jeden z **popisovačů** v některé tabulce. Popisovač mj. obsahuje bázi (adresu začátku) segmentu a limit (velikost) segmentu. Offset je potom relativní adresa uvnitř segmentu (počítá se od začátku segmentu). Offset nesmí překročit limit. Logickou adresu, ve které se jednotlivé složky (tj. selektor a offset) oddělují dvojtečkou, zapisujeme ve tvaru:

selektor **:** *offset*

Stránkováním, je-li zapnuto, se lineární adresa rozděluje na **stránky** stejné velikosti. Stránka může potom být buď v paměti, nebo na disku. Paměť je totiž stránkováním rozdělena na **rámce** stejné velikosti jako stránky. Rámce potom operační systém „pronajímá“ stránkám, které jsou v daném okamžiku potřebné. Momentálně nepotřebné stránky se odkládají na disk. Odpovídající adresaci potom zajistí právě mechanismus stránkování. Není-li potřebná stránka v některém z rámců, generuje procesor tzv. výjimku, čímž se předá řízení programové rutině zajišťující přenos potřebné stránky do paměti.

Stránkování je mechanismus určený operačnímu systému, aplikační programátor se problémy stránkování nemusí zabývat. Pro něj (ale ne jenom) je určena segmentace. Proto ještě zde v úvodu uveďme několik faktů týkajících se segmentace:

Nesegmentovaný model paměti
: Procesor nemá sice možnost segmentaci vypnout, lze však všechny potřebné popisovače nastavit tak, aby ukazovaly na jeden a tentýž prostor lineárních adres. V tomto případě, kdy segmenty pokrývají celý dostupný adresový prostor, není možné (bez použití stránkování) kontrolovat procesy, zda přistupují jenom do svého rozsahu adres, těžko se hledají některé chyby v programech atd. Výjimka se generuje pouze při přístupu mimo dostupný adresový prostor.

Segmentovaný model paměti
: Segmentovaný logický adresový prostor lze hypoteticky rozdělit až na 16 383 segmentů po 4 GB[3]. Vzniklý virtuální adresový prostor se potom stránkováním mapuje do fyzické operační paměti. Výhodou segmentového modelu je, že se kontrolují přístupy k jednotlivým segmentům: kontroluje se překročení limitu, kontroluje se oprávnění přístupu k segmentu a kontroluje se typ operace prováděné nad segmentem.

Poznámky k předchozím procesorům

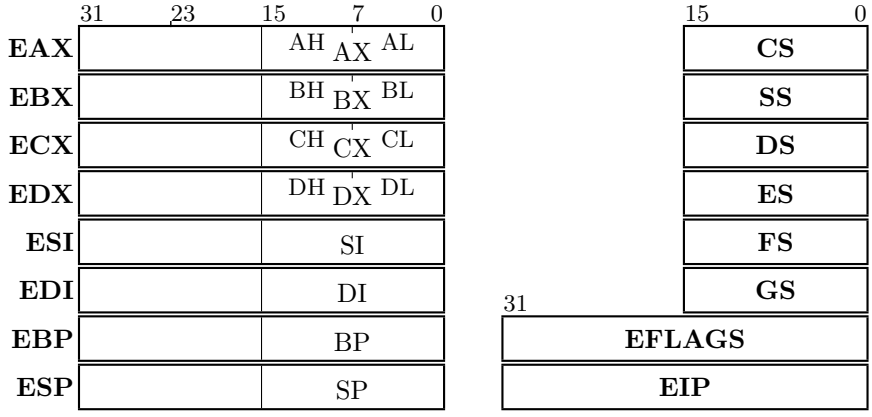
Vše, co bylo výše uvedeno, plně platí v tzv. **chráněném režimu** procesorů od Intel386. Chráněný režim byl zaveden již od procesoru Intel286, ale protože šlo o procesor 16bitový, mohly segmenty mít maximální velikost 64 KB. Procesoru 8086 byl vlastní pouze ten způsob adresace, který je ve všech vyšších typech implementován pro kompatibilitu pod názvem **reálný režim**. Popis reálného režimu je uveden v kapitole 4 od str. 43.

2.3 Registry

Na tomto místě si představíme 3 skupiny registrů. Registry jsou schematicky nakresleny na obr. 2.3.

1. **Všeobecné registry**. Jde o osm 32bitových registrů v levém sloupci obrázku. Většinou jsou určeny pro volné užití programátorem.
2. **Segmentové registry**. Je to šest 16bitových registrů určených pro uložení selektoru.
3. Příznakový registr **EFLAGS** a registr ukazatele instrukce (instrukční registr) **EIP**.

[3] Což opět hypoteticky znamená virtuální adresový prostor o kapacitě 64 TB.



Obr. 2.3 Programátorovi přístupné registry procesoru Pentium

2.3.1 Všeobecné registry

Mezi všeobecné registry patří tyto: EAX, EBX, ECX, EDX, EBP, ESP, ESI a EDI. Do všech těchto registrů lze ukládat výsledky a operandy pro aritmetické a logické operace. Rovněž je lze použít i pro adresové výpočty vyjma, registru ESP, který nelze použít pro indexaci. Dolních 16 bitů je přístupných pod označením AX, BX, CX, DX, BP, SP, SI a DI.

Zvlášť jsou ještě označeny dolní a horní slabiky registrů AX, BX, CX a DX. Jejich jména jsou AL, AH, BL, BH, CL, CH, DL a DH (viz obr. 2.3). Za těmito označeními lze vidět samostatné 16 nebo 8bitové registry. Horní slova všeobecných registrů samostatně používat nelze.

Jména registrů mají svůj význam: EAX (Accumulator) jako všeobecný střádač. EBX (Base) jako bázový registr. Jeho obsah je často interpretován jako offsetová část adresy. ECX (Counter) jako čítač např. v instrukcích cyklů. EDX (Data) je všeobecný datový registr.

Registry ESP, EBP, ESI a EDI se zpravidla používají pro uložení offsetu. Registr ESP (Stack Pointer) obsahuje offset adresy vrcholu zásobníku (viz str. 25). Kompletní adresa vrcholu zásobníku je SS:ESP (viz dále). Registr EBP (Base Pointer) je určen převážně pro uložení offsetové části adresy při práci se zásobníkem. Nejčastěji se používá pro adresaci operandů předávaných do podprogramu prostřednictvím zásobníku. ESI a EDI jsou tzv. indexové registry. ESI (Source Index) je indexový registr pro uložení offsetové části adresy zdrojového operandu a EDI (Destination Index) cílového operandu. Toto přesně definované určení indexových registrů je nut-

né dodržet v instrukcích pracujících s řetězci. Jinak registry EBP, ESI a EDI patří do skupiny všeobecných registrů. Postavení registru ESP je výsadní (viz instrukce PUSH a POP).

Poznámky k předchozím procesorům

V této podobě jsou všeobecné registry od Intel386. Předcházející typy měly pouze jejich 16 a 8bitové části. Pro uložení offsetu se směly použít pouze registry BX, BP, SI a DI.

2.3.2 Segmentové registry

Skupina segmentových registrů obsahuje šest 16bitových registrů pro uložení selektoru. Registr CS (Code Segment) ve spojení s EIP je určen pro adresaci kódu (instrukcí). Používá se pro adresaci instrukčních segmentů.

Registr DS (Data Segment) ve spojení s všeobecnými registry adresuje data. Jako pomocné datové segmentové registry lze použít ještě ES (Extra Segment), FS a GS. Ve skupině řetězcových instrukcí má speciální význam registr DS (pro adresaci zdrojového operandu) a ES (pro adresaci cílového operandu). Všechny datové segmentové registry lze použít pro adresaci datových segmentů. Registr SS (Stack Segment) adresuje zásobník (tj. speciální datový segment pro uložení zásobníku) ve spojení s registry ESP a EBP.

Poznámky k předchozím procesorům

Registry FS a GS jsou zavedeny od Intel386. Velikost všech segmentových registrů je stále stejná, pouze se liší jejich význam: v 8086 a reálném režimu vyšších procesorů uchovávají segmentovou část adresy. V chráněném režimu uchovávají selektor.

2.3.3 Příznakový registr EFLAGS

Příznakový registr EFLAGS (Flags) obsahuje dvě skupiny jednobitových informačních příznaků. První skupina (CF, PF, AF, ZF, SF, OF) jsou *příznaky nastavované procesorem* po provedení instrukce. V nich jsou uloženy informace o znaménku výsledku právě provedené instrukce, o přeplnění při aritmetické operaci atd. Druhá skupina (TF, IF, DF, IOPL, NT, RF, VM, AC, VIF, VIP, ID) jsou *příznaky nastavované programátorem*, který jimi řídí chod procesoru. Jedná se o zablokování přerušovacího systému počítače, zapnutí krokovacího režimu procesoru, změnu směru zpracovávání řetězcových operací atd. V této kapitole se omezíme na výklad pouze příznaků nastavovaných procesorem a těch příznaků, které jsou společné pro všechny režimy (ostatní příznaky jsou popsány na str. 59). Rozložení funkčních bitů je patrné z obr. 2.4 a jednotlivé příznaky mají následující význam:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	**ID**	**VIP**	**VIF**	**AC**	**VM**	**RF**
0	**NT**	**IOPL**		**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	0	**AF**	0	**PF**	1	**CF**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Obr. 2.4 Příznakový registr EFLAGS procesoru Pentium

CF (Carry Flag) se nastaví na jedničku, jestliže při právě provedené aritmetické operaci došlo k přenosu z nejvyššího bitu, a to při práci s 8, 16 nebo 32bitovým operandem. Tohoto příznaku je také využíváno při operacích logického posuvu a rotace. Příznak může měnit i programátor instrukcemi STC, CLC a CMC.

PF (Parity Flag) se nastaví na jedničku, pokud dolní osmice bitů výsledku právě provedené operace obsahuje sudý počet „1“ (sudá parita výsledku). Při lichém počtu jedniček (lichá parita) se příznak nuluje. PF se vypočítává pouze z dolní osmice bitů bez ohledu na šířku operandu.

AF (Auxiliary Carry Flag) je rozšířením příznaku CF pro přenos přes hranici nejnižší půlslabiky operandu (vždy z bitu 3 do 4 bez ohledu na šířku operandu). Má význam v BCD aritmetice (viz str. 275).

ZF (Zero Flag) je nastaven na jedničku při nulovém výsledku právě dokončené operace.

SF (Sign Flag) indikuje kladný (SF=0) nebo záporný (SF=1) výsledek právě provedené operace. Tento bit je kopií znaménkového bitu výsledku operace.

OF (Overflow Flag) se nastaví na jedničku, pokud při právě dokončené operaci došlo k aritmetickému přeplnění (nebo též říkáme přetečení). Tj. ke stavu, kdy výsledek spadá mimo rozsah zobrazení. Procesor oznámí přeplnění, pokud se přenos do znaménkového bitu nerovnal přenosu ze znaménkového bitu.

DF (Direction Flag) řídí směr zpracovávání řetězcových operací. Při DF nastaveném instrukcí STD se obsah registrů ESI a EDI zvyšuje (řetězce se zpracovávají odpředu) a při DF vynulovaném instrukcí CLD se obsah ESI a EDI snižuje (řetězce se zpracovávají odzadu).

Programování v asembleru vyžaduje detailní znalost vlivu instrukcí na jednotlivé příznaky, ty se buď nastavují, nebo nechávají nedotčeny. Proto při studiu instrukcí je nutné těmto informacím věnovat pozornost.

Poznámky k předchozím procesorům

V 8086 obsahuje registr FLAGS (nebo též F) tyto příznaky: CF, PF, AF, ZF, SF, TF, IF, DF a OF. V procesoru Intel286 přibývají příznaky IOPL a NT. Od Intel386

je registr 32bitový, jmenuje se EFLAGS a přibývají příznaky RF a VM. V Intel486 přibývá příznak AC. Příznaky VIF, VIP a ID jsou vlastní až Pentiu.

2.3.4 Registr ukazatele instrukce EIP

Registr **EIP** (Instruction Pointer) obsahuje vždy offsetovou část právě prováděné instrukce. Kompletní adresa právě prováděné instrukce je v registrech CS:EIP (viz dále). Programátor neplní tento registr přímo, pouze prostřednictvím instrukcí pro předávání řízení (např. JMP). Obsah registru je po provedení každé instrukce zvýšen o její délku tak, aby ukazoval na instrukci následující.

2.4 Zásobník

Řada instrukcí procesoru pracuje se zásobníkem buď přímo (např. instrukce pro plnění zásobníku PUSH), nebo nepřímo (např. instrukce volání podprogramu CALL, ukládající do zásobníku návratovou adresu). Procesor implementuje zásobník jako strukturu LIFO (Last In - First Out) kdekoli v operační paměti. Zásobník se ukládá do datového segmentu majícího příznak, že jde o zásobník. Na operacích se zásobníkem se podílejí tyto registry:

SS (Stack Segment)
: Zásobníky jsou uloženy v paměti. Počet zásobníků v systému je omezen pouze maximálním počtem segmentů (tj. 16 383), protože v jednom datovém segmentu s příznakem, že jde o zásobník, je uložen právě jeden zásobník. Maximální velikost zásobníku vychází z maximální velikosti segmentu (tj. 4 GB). V daném okamžiku je k dispozici pouze jeden zásobník – ten, jehož selektor obsahuje registr SS. Hovoří-li se v dalším textu o zásobníku, má se na mysli právě tento momentálně nastavený zásobník. Procesor používá obsah registru SS automaticky při provádění všech instrukcí pracujících se zásobníkem.

ESP (Stack Pointer)
: V registru ESP je uložen offset ukazatele vrcholu zásobníku (momentálně nastaveného). Ukazatel se používá při všech operacích uložení do zásobníku a výběru ze zásobníku. Jde o instrukce PUSH, POP, CALL, RET, INT apod. Operace vložení do zásobníku (PUSH) sestává ze dvou akcí:

  1. procesor sníží obsah registru ESP o velikost zásobníkové položky,
  2. potom na adresu, jejíž offset je v ESP, zapíše ukládanou hodnotu.

  Operace vybírání ze zásobníku (POP) sestává z těchto dvou akcí:

  1. procesor z adresy, na níž ukazuje offset v ESP, vybere hodnotu,

2. potom zvýší obsah registru ESP o velikost zásobníkové položky.

Z výše uvedeného plyne, že zásobník „roste“ směrem dolů – od vyšších adres k nižším.

EBP (Base Pointer)
Registr EBP používáme pro adresaci datových struktur uložených do zásobníku – např. skupiny parametrů předávaných do volaného podprogramu. Podprogram potom může zásobník používat na ukládání svých lokálních proměnných a tím měnit obsah registru ESP. Zkopírujeme-li obsah ESP do EBP dříve, než do zásobníku začneme přidávat další položky, máme v EBP neměnný offset ukazující na místo, od něhož lze kdykoli odvodit adresu potřebného parametru v zásobníku. Práci se strukturami parametrů předávaných do podprogramu nám usnadní instrukce ENTER a LEAVE.

Při každém použití adresové hodnoty v EBP se implicitně odkazujeme na právě nastavený segment se zásobníkem (tj. na segment odkazovaný registrem SS). Toto implicitní nastavení lze změnit.

Poznámky k předchozím procesorům

V procesoru 8086 a v reálném režimu vyšších procesorů nemá procesor žádný prostředek, kterým by hlídal maximální mez naplnění zásobníku. Je na programátorovi, aby si pro uložení zásobníku vymezil dostatečnou kapacitu paměti.

V 16bitových procesorech (8086, Intel286) je velikost položky ukládané do zásobníku vždy 16 bitů. Ve vyšších typech je 16 nebo 32 bitů podle typu datového segmentu, v němž je zásobník uložen.

2.5 Adresovací techniky

Příkazy určené procesoru zadává programátor v asembleru ve formě **instrukcí**. Instrukce je z jeho pohledu většinou složena ze dvou částí: z operačního znaku a operandů. **Operační znak** je vlastním příkazem pro procesor a **operandy** obsahují vlastní data (příp. doplňující informace). Operandů může být nula, jeden, dva nebo tři a mohou být zpřístupněny (adresovány) následujícími technikami.

První dvě adresovací techniky zpřístupňují operand uložený v jednom z všeobecných registrů procesoru nebo operand přímo uložený v instrukci.

2.5.1 Registr

Operand je uložen v některém 32bitovém (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), 16bitovém (AX, BX, CX, DX, SI, DI, SP, BP) nebo 8bitovém (AH, AL,

BH, BL, CH, CL, DH, DL) všeobecném registru. Některé instrukce také připouštějí uložení operandu v registrech CS, DS, ES, FS, GS, SS nebo EFLAGS.

Rovněž některé instrukce připouštějí pracovat s tzv. **registrovými páry**. V dalším textu spojujeme registry do páru znakem „&“, např. EDX&EAX znamená, že jsou spojeny registry EDX a EAX. V tomto případě EDX obsahuje 32 bitů vyšších řádů a EAX 32 bitů nižších řádů.

Příklad: Příklady adresovacích technik si budeme uvádět na instrukci
`MOV AH,`*operand*
která naplní registr AH zadanou hodnotou. Např. instrukce `MOV AH,BL` přepíše obsah registru BL do AH.

2.5.2 Přímý operand

Operand je uložen přímo v instrukci (za operačním znakem) jako konstanta.
Příklad: Instrukce `MOV AH,50` naplní registr AH číslem 50.

2.5.3 Paměťové operandy

Další techniky zpřístupňují operandy uložené ve fyzické paměti. Každá adresovací technika popisuje způsob výpočtu efektivní adresy (tj. vypočtené offsetové části adresy).

Segmentový registr se určuje většinou implicitně. Implicitní volba je dána několika faktory. Hlavním faktorem je typ operace s paměťovým operandem. Podle typu zpracovávaného operandu se potom vybírá segmentový registr, který použije segmentační jednotka pro stanovení lineární adresy operandu. Typy zpracovávaných operandů a použitý segmentový registr jsou uvedeny v tabulce na obr. 2.5.

Je-li offsetová část adresy uložena v některém ze všeobecných registrů, použije se jako implicitní segmentový registr DS vyjma případů, kdy je v instrukci použit registr ESP nebo EBP. Potom se jako implicitní použije registr SS.

2.5.4 Změna segmentového registru

V případě potřeby můžeme pro jednu instrukci „zrušit“ implicitní přiřazení segmentového registru offsetovému explicitním uvedením instrukčního prefixu před operačním znakem instrukce. Prefixy používané pro explicitní přiřazení segmentových registrů jsou uvedeny na str. 159.

Programátor v asembleru uvede před identifikaci operandu jméno segmentového registru, který se má použít, a oddělí je dvojtečkou.

Při přístupu k	*se použije registr*	*Operace*
instrukcím	**CS** (Code Segment)	Výběr operačního znaku nebo přímého operandu.
zásobníku	**SS** (Stack Segment)	Při všech přístupech k zásobníku, resp. vždy ve spojitosti s registry ESP a EBP.
datům	**DS** (Data Segment)	Při všech přístupech k datům v paměti vyjma zásobníku a přímých operandů. V řetězcových operacích pro segmentování zdrojového operandu.
alternativním datům	**ES** (Extra Segment)	V řetězcových operacích pro segmentování cílového operandu.

Obr. 2.5 Implicitní použití segmentových registrů vzhledem k typu zpracovávaného operandu.

Příklady:

```
MOV AH,CS:[BX]        Nepřímá adresa CS:BX (nikoli DS:BX)
ADD AH,DS:[BP][SI]    Kombinovaná adresa segmentovaná DS
ADC AH,GS:Adresa2     Přímá adresa segmentovaná přes GS
MOV AH,SS:Adresa      Přímá adresa segmentovaná přes SS
```

Prefixy pro změnu implicitního segmentu jsou k dispozici pro všechny segmentové registry. Instrukčními prefixy však nelze změnit nastavení segmentového registru v těchto případech:

- Cílový řetězec v řetězcových operacích je vždy segmentován registrem ES.
- Operace uložení do zásobníku a výběru ze zásobníku se vždy segmentuje registrem SS.
- Výběr instrukcí se vždy segmentuje registrem CS.

2.5.5 Výpočet efektivní adresy

Adresovací techniky, o kterých bude diskutováno dále, se už týkají pouze výpočtu efektivní (vypočtené offsetové části) adresy. Offset se počítá z těchto čtyř složek:

- přírůstku (Displacement) uloženého v instrukci; někdy též označováno jako přímá adresa,
- báze (obsah registru s bází),
- indexu (obsah indexového registru),

- násobícího faktoru (Scaling Factor), kdy se indexový registr násobí hodnotou 2, 4 nebo 8.

Offset vypočtený součtem výše uvedených komponent se nazývá **efektivní adresa**. Každá z komponent může mít kladné nebo záporné znaménko s výjimkou násobícího faktoru. Úplný výraz pro výpočet efektivní adresy je na obr. 2.6. Každá z níže uvedených technik kombinuje tyto složky jiným způsobem.

segment + báze + (index * násobící faktor) + přírůstek

$$\left\{\begin{array}{c}\textbf{CS}\\\textbf{SS}\\\textbf{DS}\\\textbf{ES}\\\textbf{FS}\\\textbf{GS}\end{array}\right\} + \left\{\begin{array}{c}\textbf{EAX}\\\textbf{ECX}\\\textbf{EDX}\\\textbf{EBX}\\\textbf{ESP}\\\textbf{EBP}\\\textbf{ESI}\\\textbf{EDI}\end{array}\right\} + \left\{\begin{array}{c}\textbf{EAX}\\\textbf{ECX}\\\textbf{EDX}\\\textbf{EBX}\\\textbf{ESP}\\\textbf{EBP}\\\textbf{ESI}\\\textbf{EDI}\end{array}\right\} * \left\{\begin{array}{c}\textbf{1}\\\textbf{2}\\\textbf{3}\\\textbf{4}\end{array}\right\} + \left\{\begin{array}{c}\text{žádný přírůstek}\\\text{8bitový přírůstek}\\\text{32bitový přírůstek}\end{array}\right\}$$

Obr. 2.6 Kompletní tvar pro výpočet efektivní adresy

2.5.6 Přírůstek

Efektivní adrese operandu umístěného ve fyzické paměti se přiřadí přírůstek uložený v instrukci. V tomto případě jde o přímou adresu, protože není kombinována s dalšími technikami.
Příklad: `MOV AH,Adresa`, kde `Adresa` je symbolicky zapsaná přímá adresa slabiky, jejíž obsah se uloží do AH.

V těchto příkladech jsme již nuceni se částečně dotknout nespecifikovaného programovacího jazyka na úrovni asembleru. Poznamenejme proto, že potřebujeme-li přímou adresu zadat absolutní hodnotou, musíme zapsat `MOV AH,DS:[101]`. V takto specifikované adrese musí být uveden segmentový registr a číslo v hranatých závorkách se nechápe jako konstanta, jíž se má naplnit registr `AH`, ale jako adresa požadovaného obsahu. Instrukce `MOV AH,[Adresa]` je ekvivalentní instrukci `MOV AH,Adresa`.

2.5.7 Nepřímá adresa

Efektivní adrese se přiřadí obsah jednoho z bázových registrů.

Příklad: `MOV AH,[EBX]`. Je-li jméno registru EBX uvedeno v hranatých závorkách, znamená to, že se do AH neuloží jeho obsah (v tomto případě by to bylo nemožné i pro nekompatibilitu typů), ale obsah ležící na adrese uvedené v zadaném registru.

2.5.8 Bázovaná adresa

Efektivní adresa se získá sečtením přírůstku umístěného v instrukci a jednoho z bázových registrů.
Příklad: `MOV AH,[EBP+Adresa]`

Bázovaná adresa je určena pro přístupy k datovým strukturám typu záznam. Do bázového registru se průběžně ukládají adresy začátků záznamů a přímá adresa obsahuje vzdálenost žádaného prvku od začátku záznamu ve slabikách.

2.5.9 Indexovaná adresa

Efektivní adresa se získá sečtením přírůstku umístěného v instrukci a jednoho z indexových registrů (jimi mohou být všechny všeobecné registry mimo ESP). Hodnota čtená z indexového registru se násobí násobicím faktorem, který je 1 (nenásobí se), 2, 4 nebo 8.
Příklad: `MOV AH,[Adresa+ESI*2]` a `MOV AH,Adresa[ESI*2]`
jsou ekvivalentní instrukce.

Indexování se používá v případě přístupu k datové struktuře s pevnou začáteční adresou (např. pole). Přímá adresa ukazuje pak na začátek pole a hodnota v registru je indexem vybírajícím daný prvek pole, přičemž násobící index odráží velikost prvku ve slabikách.

2.5.10 Kombinovaná adresa: přírůstek+báze+index

V tomto případě je efektivní adresa vypočítána jako součet obsahu bázového registru, indexového registru a přírůstku uvedeného v instrukci.
Příklad: `MOV AH,Adresa[EBX][EDI]` nebo `MOV AH,[Adresa+EBX+EDI]`
Poznamenejme, že instrukce např. `MOV AH,[Adresa+EBX+EDI+1]` je stále stejného typu, protože překladač hodnotu `(Adresa+1)` vyčíslí a uloží jako přírůstek.

Nejčastěji je tento způsob adresace používán při přístupu k složitým datovým strukturám, jako je pole v záznamu. Bázový registr ukazuje na začátek záznamu, přímá adresa určuje vzdálenost začátku pole uvnitř záznamu a hodnota indexového registru vybírá jednotlivé prvky daného pole.

2.5.11 Kombinovaná adresa: přírůstek+báze+(index*faktor)

Tento způsob výpočtu efektivní adresy je kombinací všech předchozích způsobů a vychází z obr. 2.6. Lze jej použít pro adresaci dvojdimenzionálních polí, jejichž prvky jsou 2, 4 nebo 8slabikové.

Poznámky k předchozím procesorům

Násobící faktor je zaveden až od Intel386. V procesorech 8086 a Intel286 se jako bázové registry směly použít pouze BX a BP; jako indexové registry pouze SI a DI. V reálném režimu procesorů Intel386 a vyšších nemůže být efektivní adresa vyšší než 0FFFFh.

2.6 Přerušení a výjimky

Procesor má k dispozici dva mechanismy pro přerušení běhu procesů:

1. **Výjimka** (Exception) je synchronní událost. Je to reakce procesoru na momentální podmínky, vzniklé během provádění instrukce.
2. **Přerušení** (Interrupt) je asynchronní událost, kterou zpravidla vyvolávají vnější zařízení, dožadující se pozornosti.

Vznik výjimky a přerušení znamená dočasné přerušení provádění jednoho procesu a předání řízení jinému procesu, po jehož skončení se pokračuje v provádění původního. Hlavní rozdíl mezi výjimkou a přerušením spočívá v jejich původu. Výjimku lze zpravidla opakovaně vyvolat novým spuštěním téhož procesu, zatímco přerušení vzniká shodou časových událostí, nastávajících mezi procesy a vnějšími zařízeními.

Obsluhu výjimek a přerušení zpravidla aplikační programátoři neřeší; je to věc operačního systému. Jsou však přerušení, která mohou výrazně pomoci i aplikačním programátorům, a operační systémy dávají možnosti s nimi pracovat. Podrobnosti týkající se mechanismu výjimek a přerušení jsou uvedeny v kapitole 5.7.

2.7 Ovládání vstupů a výstupů

Vstupní a výstupní operace (zkráceně vstupní/výstupní operace nebo V/V operace[4]) provádějí procesory Intel prostřednictvím V/V bran (I/O Ports). V/V brány jsou registry v řadičích periferních zařízení. V/V brána může být vstupní, výstupní nebo obousměrná. Tím je určeno, že z brány lze data pouze číst, pouze zapisovat nebo číst i zapisovat. Některé brány slouží k řízení V/V zařízení a některé k přenosu dat. V/V brány lze adresovat dvěma možnými způsoby:

[4] V originální literatuře Input/Output nebo též I/O.

1. prostřednictvím zvláštního adresového prostoru a V/V instrukcí,
2. mapováním V/V bran do adresového prostoru fyzické paměti.

2.7.1 Zvláštní adresový prostor V/V bran

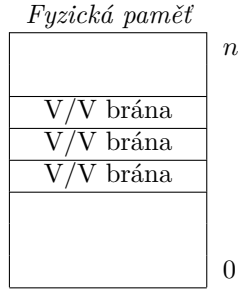
Procesor pro adresaci V/V bran poskytuje oddělený adresový prostor. Tento adresový prostor je 16bitový. Z toho plyne, že tento adresový prostor pojme 2^{16} (64 K) samostatně adresovatelných bran. Brány jsou 8bitové a lze je po dvojicích či čtveřicích spojovat do 16 či 32bitových bran.

Při přístupu k operační paměti procesor nastaví na adresovou sběrnici adresu paměti a po datové sběrnici přenáší data. Stejně tak je tomu i při adresaci V/V bran. Je-li na adresové sběrnici adresa ukazující do paměti nebo do adresového prostoru V/V bran, sděluje výstupní signál M/$\overline{\text{IO}}$ (viz popis signálů na str. 386).

Pro přístup k tomuto adresovému prostoru slouží instrukce IN, OUT (viz str. 255) a řetězcové instrukce z nich odvozené (viz str. 261). Instrukce umějí přenášet data pouze mezi střádačem a V/V bránou.

2.7.2 Mapování V/V bran do adresového prostoru fyzické paměti

Druhá možnost, jak adresovat V/V brány, je mapovat jejich adresový prostor do adresového prostoru fyzické paměti[5] (viz obr. 2.7).



Obr. 2.7 Mapování V/V bran do adresového prostoru fyzické paměti

Výhodou tohoto uspořádání je, že pro ovládání V/V bran lze použít všechny instrukce pro práci s pamětí a všechny dostupné adresovací techniky. Data lze pře-

[5] V počítačích kompatibilních s IBM PC bývá tímto způsobem konstruována video-paměť VGA.

nášet mezi bránou a libovolným registrem instrukcí MOV, manipulovat s řídicími bity instrukcemi AND, OR, TEST atd.

Nevýhodou je, že nelze zaručit serializaci V/V operací. Následuje-li např. instrukce čtení z V/V brány po instrukci zápisu do V/V brány, nelze zaručit, že čtení bude zahájeno až po dokončení zápisu. Lze však využít např. instrukce CPUID, kterou lze jako serializační[6] mezi kritické instrukce vložit. CPUID lze použít na libovolné úrovni oprávnění. Zvláštní pozor je nutné také dávat při použití stránkování a vyrovnávání (adresovací prostor V/V bran nelze stránkováním odložit apod.). Další informace, týkající se serializace, jsou uvedeny na str. 155.

2.7.3 V/V operace v chráněném režimu

V/V operace probíhají v chráněném režimu stejně jako v reálném. Rozdíl je pouze v tom, že nastupuje kontrola, zda lze V/V operaci na dané úrovni oprávnění provést (podrobnosti jsou uvedeny v kapitole 5).

- Před provedením V/V instrukce (IN, OUT) se kontroluje IOPL v registru EFLAGS a kontroluje se mapa přístupných V/V bran v TSS právě aktivního procesu (viz kapitola 5).
- Přístup k V/V bránám mapovaným do adresového prostoru se kontroluje běžnými prostředky segmentové a stránkové ochrany.

[6] Další serializační instrukce jsou INVD, INVLPG, IRET, IRETD, LGDT, LIDT, LLDT, LTR, MOV do DR, MOV do CR, RSM, WBINVD a WRMSR.

3. FPU

Na jednom čipu je spolu s procesorem integrována jednotka operací v pohyblivé řádové čárce (FPU – Floating-Point Unit). Je vlastně to, co byl matematický koprocesor Intel387 pro procesor Intel386. Programové ovládání FPU je totožné s ovládáním předchozích typů matematických koprocesorů.

3.1 Typy dat zpracovávaných FPU

FPU pracuje s osmi různými typy dat (viz též obr. 3.1). První dva typy jsou shodné s typy, které používá procesor bez FPU.

16bitové celé číslo (Word Integer) je číslo se znaménkem uložené v 16bitovém slově ve dvojkovém doplňkovém kódu (viz str. 19).

32bitové celé číslo (Short Integer) je číslo se znaménkem uložené v 32bitovém dvojslově ve dvojkovém doplňkovém kódu.

64bitové celé číslo (Long Integer) je číslo se znaménkem uložené v 64bitovém objektu ve dvojkovém doplňkovém kódu.

Zhuštěné BCD číslo (Packed BCD) je 18 BCD číslic (viz str. 275) se znaménkem v 80 bitech. Znaménko je uloženo v bitu nejvyššího řádu stejně jako u celých čísel.

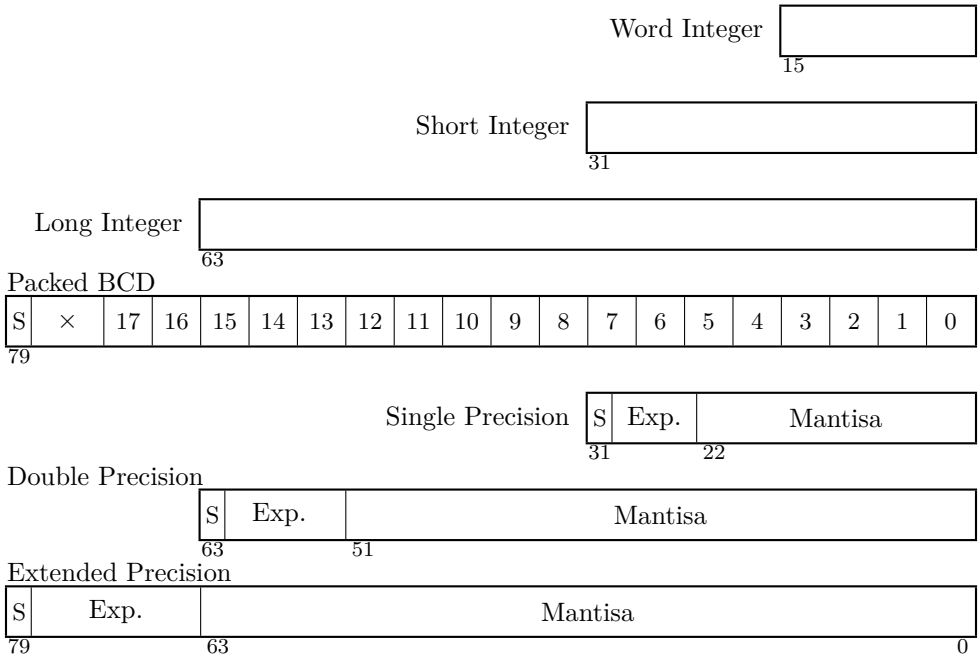
32bitové reálné číslo (Single Precision) podle standardu IEEE.

64bitové reálné číslo (Double Precision) podle standardu IEEE.

80bitové reálné číslo (Extended Precision) podle standardu IEEE.

FPU interně ukládá a zpracovává pouze 80bitová reálná čísla. Ostatní tvary jsou použity na vstupu a výstupu z FPU a FPU je interně převádí do a z 80bitových reálných čísel.

Reálná čísla jsou uložena podle standardu **IEEE 754** (Institute of Electrical and Electronics Engineers). Objekt, ve kterém je uloženo reálné číslo, je rozdělen do tří částí. Nejvyšší bit, na obr. 3.1 je označen písmenem S, je znaménkový bit. Je-li nulový, je číslo uložené v objektu kladné. Část označená „Exponent“ nese informaci o velikosti čísla a „Mantisa“ uchovává číslice čísla.



Obr. 3.1 Typy dat zpracovávaných FPU

Do mantisy jsou ukládány významné binární číslice. Neukládají se nevýznamné levostranné nuly. Aby se šířka mantisy využila co nejvíce, ukládá se v **normalizovaném tvaru**, tj. první významná číslice je umístěna v nejvyšším bitu. Z takto definovaného pravidla vyplývá, že v nejvyšším bitu musí být vždy jednička vyjma případu, kdy je v objektu uloženo číslo nula. Vezmeme-li nulu jako speciální případ, může být přesnost rozšířena o jeden bit tak, že binární jedničku nejvyššího řádu do objektu neukládáme.

„Mantisa“ 16bitových a 32bitových reálných čísel tedy podle standardu IEEE obsahuje v nejvyšším bitu druhou významnou binární číslici. V 80bitovém reálném číslu není nejvyšší významná číslice ignorována.

„Mantisa“ vyjadřuje binární číslo ve tvaru $1.xxxxxx$. „Exponent“ potom určuje počet binárních řádů, o které musíme desetinnou tečku pomyslně posunout, abychom získali požadované číslo. Posouváme-li tečku doleva, bude exponent záporný, posouváme-li doprava, bude exponent kladný. Zjednodušeně lze číslo vyjádřit matematicky: $Mantisa \times 2^{Exponent}$

„Exponent“ je celé číslo kladné nebo záporné. Není vyjádřen ve dvojkovém doplňkovém kódu, jak by se dalo předpokládat, ale v kódu posunuté nuly. Je-li prostor pro uložení exponentu 8bitový (Single Precision), je k zapisovanému číslu přičtena hodnota 7Fh (127). Potom nula bude zapsána jako 7Fh a jednička jako 80h atd. Je-li prostor pro uložení exponentu 11bitový (Double Precision), přičítá se 3FFh (1 023), a je-li exponent 15bitový, přičítá se 3FFFh (16 383). Při čtení čísla tohoto tvaru musíme výše uvedenou hodnotu odečíst.

Ačkoli se tento způsob ukládání čísel zdá být nevhodný, přináší jeden významný efekt, který spočívá ve snadném znaménkovém porovnávání dvou reálných čísel bez ohledu na jejich šířku. Obě reálná čísla začneme srovnávat od znaménkového bitu směrem k nejnižšímu bitu objektu. Jakmile narazíme na dvojici neshodujících se bitů, sdělí nám tyto dva bity, které z čísel je numericky větší.

Nabízíme několik příkladů, aby si čtenář mohl ověřit, že IEEE formát reálného čísla správně pochopil:

Číslo	S	Exponent	Mantisa
	31	30 23	22 0
12.5	0	1000 0010	1001 0000 ... 0
-12.5	1	1000 0010	1001 0000 ... 0
0.3125	0	0111 1101	0100 0000 ... 0
-0.3125	1	0111 1101	0100 0000 ... 0
1.0	0	0111 1111	0000 0000 ... 0

Číslo	S	Exponent	Mantisa
	79	78 64	63 0
12.5	0	100 0000 0000 0010	1100 1000 ... 0
-0.3125	1	011 1111 1111 1101	1010 0000 ... 0

3.2 Výsadní symboly

Nula (Zero) může být kladná nebo záporná. FPU běžně produkuje kladnou nulu.

Nekonečno (Infinity) může být opět zobrazeno jako kladné nebo jako záporné. Pokud bychom obě tato nekonečna srovnávali, je kladné nekonečno větší než záporné.

Číslo	S	Exponent	Mantisa
	31	30 23	22 0
Kladná nula	0	0000 0000	0000 0000 ... 0
Záporná nula	1	0000 0000	0000 0000 ... 0
Kladné nekonečno	0	1111 1111	0000 0000 ... 0
Záporné nekonečno	1	1111 1111	0000 0000 ... 0

Nenormalizované číslo se připouští pouze tehdy, potřebujeme-li zapsat menší číslo v absolutní hodnotě, než je exponent schopen zobrazit. Potom doplněním nevýznamných levostranných nul do mantisy můžeme číslo ještě (v absolutní hodnotě) zmenšit na úkor přesnosti. O číslu musí být známo, že je v nenormalizovaném tvaru, aby se nedoplňovala jednička do nejvyššího řádu mantisy.

Nečíselné hodnoty (Not a Number) se uvádějí pod zkratkou NaN. Poněvadž nejsou povoleny všechny kombinace bitů čísla tvaru IEEE, jsou nepovolené kombinace označeny NaN a FPU takové vstupní hodnoty odmítne. Nečíselné hodnoty jsou rozděleny do dvou skupin: neohlašované (Quiet) a ohlašované (Signaling). Do neohlašovaných patří čísla s mantisou v rozmezí od 100…01 do 111…11. Do ohlašovaných patří 100…01 až 101…11. Obě skupiny mají exponent 111…11.

Neurčitá hodnota (Indefinite) je určena pro každý typ čísla včetně celočíselných. Běžně je neurčitá hodnota představována největším záporným číslem v absolutní hodnotě. V reálných formátech je neurčitá hodnota tvořena exponentem obsahujícím samé jedničky a mantisou obsahující 110…00.

3.3 Výjimky FPU

Detekuje-li FPU chybu při výpočtu, procesor na tento stav odpoví generováním výjimky 16. FPU umí detekovat několik různých chyb. Konkrétní druh chyby se oznamuje ve stavovém registru FPU. Možné chyby jsou následující:

I - Invalid operation - Chybná operace je detekována rozpoznáním NaN, nepodporovaného formátu čísla, nepovolenou operací (např. $0 \times \infty$, $0/0$, $(+\infty) + (-\infty)$) nebo přeplněním zásobníku (potom je také nastaveno SF). Je-li toto přerušení maskováno (viz dále), vrací se jako výsledek neohlašované NaN, nedefinované celé číslo nebo nedefinované BCD.

D - Denormalized - Nenormalizovaný výsledek je zapříčiněn hodnotou výsledku, která je v absolutní hodnotě menší než číslo zobrazitelné nejmenším záporným normalizovaným číslem. Je-li přerušení maskováno, výpočet pokračuje normálně dál.

Z - Divide by zero - Dělení nulou nastane tehdy, je-li dělitel nula a dělenec je různý od ∞ a 0. Je-li přerušení maskováno, je výsledek ∞.

O - Numeric overflow - Přeplnění (Přetečení) je detekováno tehdy, je-li výsledek větší než maximální zobrazitelné číslo v použitém formátu dat. Je-li přerušení maskováno, je výsledek největší možné konečné číslo nebo ∞.

U - Numeric underflow - Nenaplnění (Podtečení) nastane při nenulovém výsledku operace, který je tak malý, že jej nelze v daném formátu dat zobrazit. Je-li přerušení maskováno, je výsledek nenormalizované číslo nebo nula.

P - Precision - Nepřesný výsledek nastane tehdy, nebyla-li FPU schopna vypočítat výsledek v zadané přesnosti. Výsledek je zaokrouhlen podle řídicího registru. Je-li přerušení maskováno, výpočet pokračuje dál.

3.4 Registry FPU

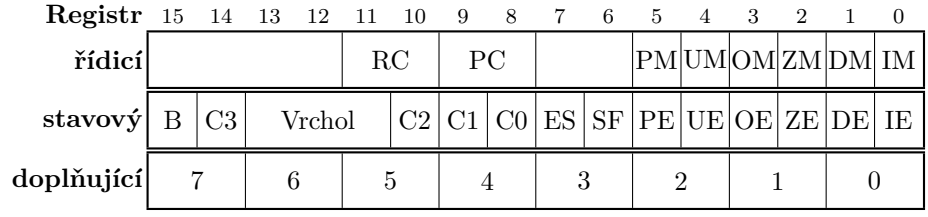
FPU obsahuje 8 datových registrů, které jsou 80bitové, a tři 16bitové pomocné registry: řídicí, stavový a doplňující.

Datové registry jsou očíslovány 0 až 7. Každý má kapacitu 80 bitů a slouží pro uložení zpracovávaného čísla v libovolném formátu. Skupina registrů může být použita buď jako zásobník, nebo jako jednotlivě adresované registry.

Používá-li se skupina registrů jako zásobník, potom je ve stavovém registru v položce „Vrchol“ číslo registru, který je právě na vrcholu zásobníku – tj. číslo registru, do kterého byla operací vložení do zásobníku zapsána poslední hodnota. Operace vložení do zásobníku nejprve sníží hodnotu „Vrchol“ o jedničku a potom do registru číslo „Vrchol“ vloží hodnotu. Operace výběru ze zásobníku nejprve vybere hodnotu z registru číslo „Vrchol“ a potom zvýší „Vrchol“ o jedničku.

Registr, který je právě na vrcholu zásobníku, je označován ST(0) nebo jenom ST. Ukazuje-li „Vrchol“ např. na registr číslo 2, potom ST(0) znázorňuje registr číslo 2, ST(1) registr číslo 3, ST(2) registr číslo 4, ST(5) registr číslo 7, ST(6) registr číslo 0 a ST(7) registr číslo 1.

Osm datových registrů je skutečně implementováno jako zásobník, nikoli jako kruhová fronta. Proto je-li do zásobníku vloženo více než 8 operandů, generuje FPU přerušení chybná operace (ve stavovém registru je nastaveno SF – chyba zásobníku a C1 – přeplnění). Přerušení se generuje také v tom případě, pokud je ze zásobníku vybráno více než 8 položek.



Obr. 3.2 Registry FPU

Řídicí registr, jehož tvar je na obr. 3.2, nastavuje některé parametry pro výpočty v FPU:

RC (Rounding Control) řídí zaokrouhlování výsledků. Možné kombinace jsou tyto:

00 zaokrouhlení na nejbližší číslo,
01 zaokrouhlení dolů (směrem k zápornému nekonečnu),
10 zaokrouhlení nahoru (směrem ke kladnému nekonečnu),
11 oseknutí.

Implicitně je RC nastaveno na 00.

PC (Precision Control) stanovuje přesnost výsledku. Přesnost může být snížena proto, aby byl zrychlen výpočet. Možné kombinace jsou tyto:

00 24 bitů (Single Precision),
01 nepoužito,
10 53 bitů (Double Precision),
11 64 bitů (Extended Precision).

Implicitně je PC nastaveno na 11.

PM (Precision Mask) jedničkové maskuje přerušení vyvolané nepřesným výsledkem.

UM (Underflow Mask) jedničkové maskuje přerušení vyvolané nenaplněním.

OM (Overflow Mask) jedničkové maskuje přerušení vyvolané přeplněním.

ZM (Divide-by-Zero Mask) jedničkové maskuje přerušení vyvolané dělením nulou.

DM (Denormalized Mask) jedničkové maskuje přerušení vyvolané nenormalizovaným výsledkem.

IM (Invalid Operation Mask) jedničkové maskuje přerušení vyvolané chybnou operací.

Stavový registr obsahuje stav FPU. Rovněž je v něm zaznamenána informace o vzniklé chybě tak, aby rutina obsluhující INT 16 mohla blíže upřesnit příčinu přerušení.

B (Busy) je zachováván pro kompatibilitu s koprocesorem 8087. Neodráží aktuální stav FPU, obsahuje totéž co bit ES.

C3, **C2**, **C1**, **C0** (Condition C*i*) jsou čtyři bity příznaků nastavovaných podle výsledku provedené operace.

Vrchol (Top) je 3bitová hodnota uchovávající číslo datového registru, který je právě na vrcholu zásobníku.

ES (Error Summary) je programovým duplikátem signálu $\overline{\text{FERR}}$. Sděluje, že nastalo přerušení, které bylo vyvoláno jednou z nemaskovaných chybových příčin.

SF (Stack Fault) upřesňuje, že chybná operace byla vyvolána chybou v zásobníku. Pokud je tento bit nastaven, je C1=1 při přeplnění zásobníku nebo C1=0 při nenaplnění zásobníku.

PE, **UE**, **OE**, **ZE**, **DE**, **IE** Každý z těchto bitů oznamuje, že přerušení nastalo po výskytu příslušné chyby. Jednotlivé bity odpovídají bitům řídicího registru, které na rozdíl od stavového registru jednotlivá přerušení maskují.

Doplňující registr obsahuje doplňující informaci o každém z datových registrů. Každému z 8 datových registrů jsou zde vyhrazeny 2 bity (viz obr. 3.2). Jsou čtyři možné kombinace:

00 správný obsah registru,
01 nula,
10 nekonečno, NaN, nenormalizováno nebo chybný formát,
11 prázdný.

3.5 Komunikace procesoru a FPU

Instrukce určené FPU (stejně jako koprocesoru ve starších typech procesorů) jsou v instrukčním toku společně s instrukcemi pro procesor. Jakmile procesor rozpozná, že instrukce je určena FPU (prvních 5 bitů instrukce je 11011, viz též ESC), předá ji na V/V bránu určenou pro předávání příkazů FPU (8000 00F8h). Potom procesor normálně pokračuje ve zpracovávání instrukčního toku. Pokud FPU požaduje více informací (tj. další slabiku operačního kódu nebo slabiku operandu), nastaví signál PEREQ (Processor Extension Request) do aktivní úrovně (tak je tomu u Intel386, ve vyšších typech procesorů není tento signál vyveden vně procesoru). Na základě toho si procesor přečte V/V bránu pro předávání příkazů FPU, aby zjistil, co FPU potřebuje. Pokud FPU vyžaduje přenést data, předá je procesor prostřednictvím datové V/V brány (8000 00FCh).

Po dobu vykonávání instrukce je nastavena aktivní úroveň signálu $\overline{\text{BUSY}}$ (v Intel486 a vyšších není opět vyveden vně procesoru), který informuje procesor o tom, že FPU není schopno přijímat další příkazy. V takovém případě procesor čeká na dokončení činnosti FPU.

4. Reálný režim

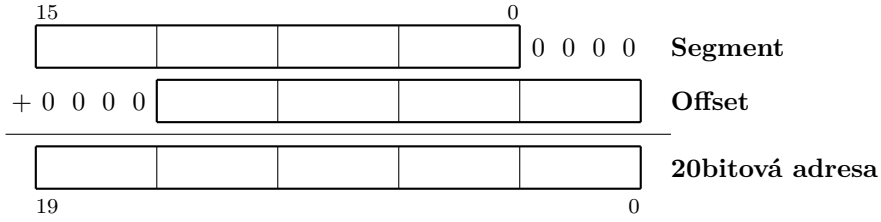
Reálný režim procesoru Pentium je určen na spouštění programů napsaných pro procesory 8086, 8088, 80186 a 80188 nebo pro reálný režim procesorů Intel286, Intel386 a Intel486.

Architektura procesoru Pentium běžícího v tomto režimu je redukována na architekturu 8086 – Pentium je tímto degradováno na rychlou 8086 s několika rozšířeními instrukčního repertoáru a s možností pracovat ve 32bitové aritmetice.

4.1 Adresace paměti

V reálném režimu se adresa zadaná obsahem segmentového registru a offsetem překládá přímo na adresu fyzickou. Obsah segmentového registru není interpretován jako selektor ukazující do některé z tabulek, nýbrž obsah segmentového registru je součástí výsledné fyzické adresy.

V procesoru 8086 byla maximální adresovatelná kapacita 1 MB, čemuž odpovídala potřeba 20bitové adresy. Protože však procesor zpracovával pouze 16bitové údaje, byl zde technicky implementován mechanismus výpočtu fyzické adresy ze dvou složek. První složka, tvořící nižší bity fyzické adresy, je **offset** (efektivní adresa) uložený na 16 bitech a druhá složka, tvořící vyšší bity, se nazývá **segment**. Fyzická adresa se vytvoří sečtením těchto dvou částí vzájemně posunutých o 4 bity (viz obr. 4.1).



Obr. 4.1 Vytváření 20bitové fyzické adresy v procesoru 8086

Předpokládejme, že počítáme fyzické adresy instrukcí a že hodnota segmentu je nastavena při zahájení programu a hodnota offsetu se mění. Při spuštění se offset nastaví na nulu (tak jej vytvořil překladač a sestavovací program). Potom program může být v paměti umístěn pouze od adres dělitelných 16 a velikost programu smí být maximálně 64 KB (adresovatelná kapacita 16 bitů offsetu). Je-li potřeba zpracovávat programy větší než 64 KB, musí se za běhu procesu měnit hodnota segmentu.

Stránkování v reálném režimu dostupné není, proto adresový prostor reálného režimu začíná vždy od začátku operační paměti. V procesoru Intel286 je adresová aritmetika 24bitová a od Intel386 je 32bitová. Při sčítání segmentu a offsetu podle obr. 4.1 v prostředí takové vícebitové aritmetiky může součet být až 10FFEFh (segment 0FFFF0h plus offset 0FFFFh), což je 1 M plus necelých 64 K a takovou adresovou kapacitu lze v reálném režimu procesorů od Intel286 používat. Ve 20bitové aritmetice (tj. v procesoru 8086) ukazují adresy, u kterých součet složek je 1 M a vyšší, na začátek paměti (např. výše zmíněný součet se upraví na hodnotu 0FFEFh – přenos z nejvyššího bitu se ignoruje). Procesory Intel486 a Pentium mají signál $\overline{\text{A20M}}$, kterým se v reálném režimu zapíná maskování podle pravidel 8086.

Procesory od Intel386 mohou i v reálném režimu při použití prefixu změny velikosti adresy generovat 32bitovou efektivní adresu s hodnotou vyšší než 0FFFFh. V reálném režimu však efektivní adresa být vyšší než 0FFFFh nesmí a při pokusu použít takovou adresu se generuje výjimka 12 nebo 13 bez chybového kódu.

4.2 Registry a instrukce

V reálném režimu lze používat všechny všeobecné registry (vč. 32bitových), všechny segmentové registry (vč. FS a GS), registry ladicího systému, řídicí registry, testovací registry a registry FPU. Dostupné jsou též nové instrukce pracující s těmito registry, prefixy nastavující použití nových segmentových registrů, prefixy měnící velikost adresy a operandu, nové techniky pro vyčíslování efektivních adres a s nimi spojená možnost použít i jiné registry pro index a bázi. Dostupné jsou rovněž všechny instrukce, které byly do instrukčních repertoárů přidávány u vyšších typů následujících za procesorem 8086. V reálném režimu nejsou úrovně oprávnění, a proto lze používat všechny instrukce z chráněného režimu.

Naopak výjimku „chybný operační znak“ v reálném režimu generují ty instrukce určené pro chráněný režim, které jako svůj vstup požadují popisovač některé z tabulek, tj. VERR, VERW, LAR, LSL, LTR, STR, LLDT a SLDT.

4.3 Přerušení a výjimky

Přerušení a výjimky vznikají a obsluhují se v reálném režimu téměř shodně s procesorem 8086. Rozlišuje se 256 různých přerušení a výjimek. Pro každé přerušení nebo výjimku je v paměti uloženo 32 bitů adresy (přerušovací vektor) začátku obslužné programové rutiny. Adresy jsou zapsány v **tabulce přerušovacích vektorů** (viz obr. 4.2). Tabulka má velikost 1 KB a je implicitně uložena na začátku paměti od adresy 0000:0000.

31	0	Adresa	Číslo přer. vektoru
segment	*offset*	0:03FC	INT 0FFh
	⋮	⋮	
segment	*offset*	0:000C	INT 3
segment	*offset*	0:0008	INT 2
segment	*offset*	0:0004	INT 1
segment	*offset*	0:0000	INT 0

Obr. 4.2 Tabulka přerušovacích vektorů reálného režimu

Po přijetí žádosti o přerušení provádí procesor v reálném režimu tyto akce:

1. do zásobníku se uloží registr příznaků (FLAGS),
2. vynulují se příznaky IF a TF,
3. do zásobníku se uloží registr CS,
4. registr CS se naplní 16bitovým obsahem adresy $n \times 4 + 2$,
5. do zásobníku se uloží registr IP ukazující na neprovedenou instrukci,
6. registr IP se naplní 16bitovým obsahem adresy $n \times 4$.

Výjimky v reálném režimu nevracejí chybový kód. Návrat do přerušeného procesu a jeho pokračování zajistí instrukce IRET, která provede činnosti v tomto pořadí:

1. ze zásobníku obnoví registr IP,
2. ze zásobníku obnoví registr CS,
3. ze zásobníku obnoví příznakový registr (FLAGS).

Procesory od Intel386 nemusí mít v reálném režimu tabulku přerušovacích vektorů uloženu od adresy 0000:0000. To je hlavní rozdíl mezi obsluhou přerušení v 8086 a typech od Intel386. I když jde o reálný režim, bere se ve vyšších typech procesorů v úvahu obsah registru IDTR. Tohoto faktu si však programátoři v reálném režimu

nemusejí všímat, protože po inicializaci procesoru, je báze v IDTR nastavena na 0 a limit na 3FFh, což jsou hodnoty kompatibilní s 8086. Obsah IDTR lze v reálném režimu změnit instrukcí LIDT. Nastane-li přerušení požadující vektor, který je mimo limit nastavený v IDTR, generuje se výjimka „dvojnásobný výpadek“.

Některé výjimky procesor oznamuje v reálném režimu jinak než v režimu chráněném. Podrobnosti jsou v tabulce na obr. 4.3. Podrobný popis výjimek chráněného režimu je uveden od str. 95. Přerušení v reálném režimu **nepředávají** chybové slovo (viz kapitola 5). Jiný význam mají tyto výjimky:

INT 8 Příliš malý limit (Interrupt Table Limit Too Small)

Tato výjimka se může v reálném režimu vyskytnout tehdy, byl-li instrukcí LIDT změněn limit tabulky přerušovacích vektorů a adresa vypočtená z přerušovacího vektoru je větší než tento limit.

INT 13 Překročení segmentu (Segment Overrun)

Tato výjimka se vyskytne v reálném režimu tehdy, pokud nelze operand umístit do segmentu. Např. při pokusu umístit do segmentu slovo od offsetu 0FFFFh.

Poznámky k předchozím procesorům

Procesor 8086 generuje výjimky 0 až 4. V procesoru Intel286 se v reálném režimu mohou navíc vyskytnout tyto výjimky: 5, 6, 7, 8, 9, 13 a 16. Procesor Intel386 rozšiřuje interpretaci výjimky 1.

Ukazuje návratová adresa na instrukci, která přerušení/výjimku způsobila?			
Číslo vektoru	*Význam výjimky nebo přerušení*	*Výjimku nebo přerušení způsobuje*	
0	Dělení nulou	instrukce DIV a IDIV	ano
1	Ladicí systém	libovolná instrukce	*[1]
2	Nemaskovatelná přerušení (NMI)	nemaskovatelné přerušení	ano
3	Ladicí bod	instrukce INT 3	ne
4	Přeplnění	instrukce INTO	ne
5	Kontrola mezí	instrukce BOUND	ano
6	Chybný operační znak	vyhrazené operační znaky a nepovolené použití prefixu LOCK	ano
7	Zařízení nepřístupné	instrukce ESC nebo WAIT	ano
8	Dvojnásobný výpadek	limit přerušovací tabulky je malý, nastal výpadek při obsluze jiného výpadku	ano
9	Rezervováno		
10	Chybný TSS[3]	instrukce JMP, CALL, IRET, přerušení a výjimky	ano
11	Segment nepřístupný[3]	libovolná instrukce měnící segmenty	ano
12	Výjimka zásobníku	operace překročila limit (přes offset 0FFFFh)	ano
13	Překročení segmentu	operand velikosti slova překročil max. hodnotu offsetu 0FFFFh, pokus o provedení instrukce za koncem segmentu CS	ano
14	Výpadek stránky[3]	libovolná instrukce přistupující k paměti	ano
15	Rezervováno		
16	FP chyba	instrukce ESC nebo WAIT	ano[2]
17	Kontrola zarovnání[3]	libovolný odkaz na data	ne
18-31	Rezervováno firmou Intel		
0-255	Programová přerušení	instrukce INT *n*	ne
32-255	Maskovatelná přerušení		

Obr. 4.3 Přerušení a výjimky v reálném režimu (1)

Poznámky:

1. Některé výjimky ladicího systému ukazují na instrukci způsobující výjimku, jiné ukazují na instrukci následující. Obsluha výjimky musí testovat obsah registru DR6 na zjištění typu přerušení.
2. Chyby FPU hlásí první instrukce ESC nebo WAIT následující po instrukci, která přerušení způsobila.
3. Výjimky 10, 11, 14 a 17 nenastávají v reálném režimu, v režimu V86 nastat mohou.

Obr. 4.4 Přerušení a výjimky v reálném režimu (2)

5. Přehled architektury chráněného režimu

V této kapitole jsou popisovány principy chráněného režimu, který tvoří jádro architektury procesoru Pentium.

5.1 Správa paměti v chráněném režimu

Filozofie správy a adresování paměti v chráněném režimu je jiná než v režimu reálném. Správa paměti nabízí dva mechanismy: *segmentaci* a *stránkování*. Segmentováním získá každý program svůj vlastní, na ostatních nezávislý chráněný adresový prostor. Stránkováním jsme schopni vytvořit virtuální adresový prostor větší kapacity, než je kapacita paměti RAM.

Poznámky k předchozím procesorům

Stránkování je zavedeno od procesorů Intel386, chráněný režim spolu se segmentací od Intel286.

5.2 Segmentace paměti

Paměť je rozdělena na tzv. **segmenty**. Segment je souvislý prostor paměti velikosti až 4 GB, který může začínat na libovolné adrese. Každý segment je definován těmito parametry:

1. bází segmentu (adresou začátku segmentu),
2. limitem segmentu (délkou segmentu ve slabikách – 1),
3. přístupovými právy a typem segmentu.

Uvnitř jednoho segmentu se pohybujeme pomocí zadání 32bitového offsetu, který přičtením k bázi určí lineární adresu v paměti.

Základním pojmem, který se váže k organizaci paměti, je proces (Task). Adresový prostor procesu může být rozdělen mezi **lokální adresový prostor** (Local Address

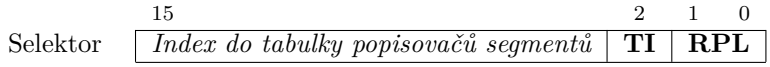
Space) a **globální adresový prostor** (Global Address Space). Do lokálního adresového prostoru proces umísťuje vlastní jedinečná data a proměnné. Jiné procesy sem nemají přístup. Obsahem globálního adresového prostoru může být např. kód programu, který je souběžně spuštěn více uživateli (např. překladač, jehož proměnné má každý proces ve vlastním lokálním prostoru). Tento prostor může být sdílen více procesy.

Poznámky k předchozím procesorům

Protože Intel286 byl 16bitový procesor, měl též offset velikost 16 bitů. Z toho důvodu byla maximální velikost segmentu 64 KB. To byl jeden z hlavních důvodů (spolu s neexistencí stránkování) nepoužitelnosti procesoru pro „skutečné“ operační systémy (UNIX apod.).

5.2.1 Logická adresa

Adresa v chráněném režimu, stejně jako v reálném, je složena ze dvou složek. Interpretace složky nazývané **offset** je stejná. Použití druhé složky je v chráněném režimu jiné. Tuto složku nazvěme **selektor segmentu** (Segment Selector). Kompletní adresu v *chráněném režimu* (selektor a offset) nazýváme **logická adresa**.



Obr. 5.1 Struktura selektoru

Selektor segmentu je 16bitové slovo obsahující 13 bitů (8 192 kombinací) **indexu** do tabulky popisovačů segmentů lokálního nebo globálního adresového prostoru a další 3 informační bity s tímto významem:

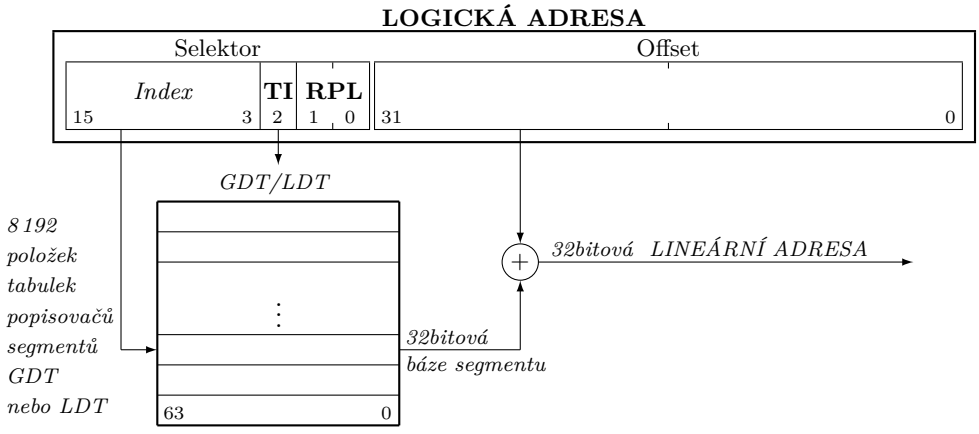
RPL (Requestor Privilege Level) představuje úroveň oprávnění, kterou proces nabízí při přístupu k tomuto segmentu. Proces si tímto může svoji úroveň oprávnění pouze snížit.

TI (Table Indicator) indikuje, ukazuje-li index do tabulky popisovačů segmentů lokálního adresovacího prostoru (TI=1) nebo globálního adresovacího prostoru (TI=0).

Kombinace Index=0 a zároveň TI=0 se nazývá **neplatný selektor** a má speciální význam (viz str. 75).

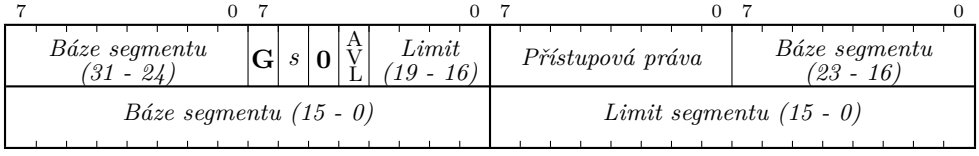
5.2.2 Tabulky popisovačů segmentů

Obraz globálního adresového prostoru udržuje tabulka popisovačů globálního adresového prostoru **GDT** (Global Descriptor Table). Pro lokální prostory jsou k dispozici tabulky **LDT** (Local Descriptor Table). Část logické adresy nazvaná **selektor** ukazuje do tabulky popisovačů segmentů buď globálního adresového prostoru (TI=0), nebo lokálního adresového prostoru (TI=1). Všechny tyto tabulky jsou uloženy v paměti a jejich prostřednictvím se transformují logické adresy na 32bitové **lineární adresy** (viz obr. 5.2).



Obr. 5.2 Transformace logické adresy na lineární pomocí tabulek popisovačů segmentů

Tabulky popisovačů segmentů mají nejvýše 8 192 položek. Každá položka má délku 8 slabik, které obsahují následující informace: **bázi segmentu** (adresu začátku segmentu) ve 32 bitech, **limit segmentu** (nejvyšší adresu uvnitř segmentu) ve 20 bitech, **přístupová práva AR** (Access Rights) v 8 bitech a další pomocné bity. Rozmístění jednotlivých informací v položce je patrné z obr. 5.3.



Obr. 5.3 Formát popisovače segmentu

Uveďme ještě několik podrobností k položkám popisovače:

Báze segmentu definuje začátek segmentu v rámci 4 GB prostoru lineárních adres. Procesor si bázi složí ze tří polí (viz obr. 5.3). Segment může začínat na libovolné adrese, žádoucí je však segmenty zahajovat na adresách dělitelných 16.

G (Granularity) nulová sděluje, že limit v popisovači je v jednotkách 1 B (potom segment může mít velikost max. 1 MB). Je-li G=1, jsou jednotkou limitu 4 KB (potom segment může mít velikost až 4 GB v násobcích 4 KB).

Limit segmentu určuje velikost segmentu zadáním maximální adresy. Offset potom může nabývat hodnot 0 až *limit*. Jiné hodnoty vyvolají výjimku. Jinak je tomu u segmentů „rostoucích dolů“ (např. segmenty pro uložení zásobníku), tyto naopak musejí mít offset mimo interval 0 až *limit*.

Hodnota položky *limit* se interpretuje podle obsahu bitu G (Granularity). Je-li G=1, může mít segment velikost až 4 GB. Je-li G=1, je maximální velikost 1 MB.

AVL (Available for Programmer Use) je bit s nespecifikovaným významem, který může programové vybavení (operační systém) používat podle svých potřeb.

s je bit, který má různý význam v závislosti na typu popisovače.

Typ popisovaného segmentu je definován obsahem slabiky **přístupová práva**. Podle typu segmentu rozlišujeme tyto tři základní třídy popisovačů:

1. popisovač segmentu obsahujícího data (datový segment),
2. popisovač segmentu obsahujícího instrukce (instrukční segment),
3. popisovače zvláštních typů označené jako systémové.

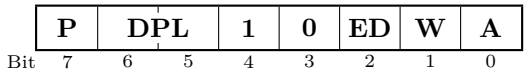
Poznámky k předchozím procesorům

Procesor Intel286 měl bázi 24bitovou a limit 16bitový. Velikost popisovače byla stejná s tím, že nejvyšší dvě slabiky se nepoužívaly. V dokumentaci procesoru se doporučovalo, aby programátor obsah těchto dvou slabik udržoval nulový. Pokud tak učinil, je jeho program bez úprav přenositelný i na vyšší typy procesorů.

5.2.3 Popisovač datového segmentu

Popisovač datového segmentu ukazuje na segment v paměti obsahující data určité aplikace nebo zásobník. Že jde o popisovač datového segmentu, procesor rozpozná podle obsahu slabiky přístupová práva (viz obr. 5.4). Význam má také bit, který byl na obr. 5.3 označen jako *s*.

Bity 4 a 3 sdělují, že jde o popisovač datového segmentu. Ostatní bity mají tento význam:



Obr. 5.4 Přístupová práva popisovače datového segmentu

P (Segment Present) je nastaven na jedničku tehdy, je-li segment je uložen v lineární paměti (v textu označujeme, že segment je „přítomný“). V tomto případě musejí být v popisovači uloženy údaje tak, jak je procesor požaduje. Je-li bit P nulový, není segment v paměti a operační systém může zbytek popisovače využít ke svým účelům.

Pokus o přístup k segmentu, který není přítomný, vyvolá výjimku 11 (Segment not Present) nebo 12 (Stack Exception), v závislosti na typu uložených dat.

DPL (Descriptor Privilege Level) určuje úroveň oprávnění přidělenou segmentu, který je adresován popisovačem.

ED (Expand Down) indikuje, kterým směrem se bude obsah segmentu rozšiřovat. Datové segmenty mohou obsahovat klasická data nebo zásobníky. Při požadavku na zvětšení klasických dat se bude obsah rozšiřovat směrem k vyšším adresám. U zásobníku tomu bude naopak, protože ten se plní od vyšších adres k nižším (viz str. 25).

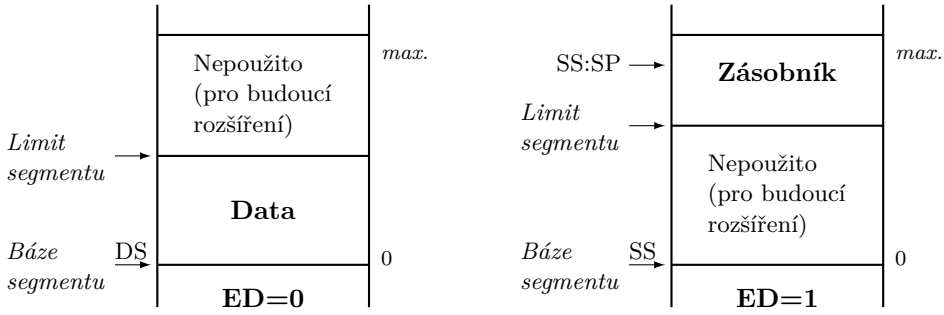
Je-li nastaveno ED=0 (rostoucí nahoru – data), bude se obsah segmentu rozšiřovat směrem k vyšším adresám. Data se ukládají od adresy 0 směrem k adrese 0FFFFFFFFh. Při požadavku na zvětšení obsahu se musí zvětšit hodnota limitu segmentu. Správná hodnota offsetu je v intervalu 0 až *limit* včetně. Hodnota offsetu mimo tento interval generuje výjimku.

Je-li nastaveno ED=1 (rostoucí dolů – zásobník), bude se obsah segmentu rozšiřovat směrem k nižším adresám. Položky zásobníku se ukládají od adresy 0FFFFFFFFh směrem k adrese 0. Při požadavku na zvětšení obsahu se musí zmenšit hodnota limitu segmentu (ten se totiž stále počítá od adresy 0). Pro srovnání viz obr. 5.5. Správná hodnota offsetu je v intervalu (*limit*+1) až 0FFFFFFFFh.

Jinými slovy lze říci, že při ED=0 se překročení limitu hlídá směrem od nižších adres a při ED=1 se překročení limitu hlídá směrem od vyšších adres.

W (Writable) je nastaven na 1, pokud je povoleno čtení i zápis do segmentu (v textu segment označujeme jako „zapisovatelný“). Nulová hodnota bitu W zakazuje zápis dat a je povoleno pouze čtení. Zásobník musí mít vždy W=1.

A (Accessed) nastavuje procesor na jedničku při každém přístupu k této položce v tabulce popisovačů segmentů (zavedení do segmentového registru nebo



Obr. 5.5 Srovnání datového segmentu rostoucího nahoru a dolů

použití instrukce testující selektor). Procesor tento příznak nenuluje. Je určen operačnímu systému ke sledování četnosti přístupů ke konkrétním segmentům.

Procesory Intel486 a Pentium nastavují příznak A pouze tehdy, je-li nulový. Tím se liší od předchozích typů, které příznak nastavovaly při každém přístupu k segmentu bez ohledu na hodnotu příznaku. U těchto starších procesorů dochází k potížím tehdy, je-li popisovač segmentu uložen v paměti ROM. Technické vybavení potom musí vynechávat zápisové cykly do této paměti určené pouze ke čtení. U procesorů od Intel486 stačí popisovač do ROM zapsat s hodnotou A=1, čímž se zabrání nastavování příznaku.

Bit *s* (viz obr. 5.3) je v popisovači datového segmentu označován B a má tento význam:

B (Big) má význam v datovém segmentu rozšiřujícím se směrem dolů (ED=1, segment pro uložení zásobníku). Je-li B=0, může mít segment kapacitu max. 64 KB a zaplňuje se od adresy max. FFFFh. Je-li B=1, může mít segment kapacitu až 4 GB (potom musí být G=1) a zaplňuje se od max. FFFFFFFFh k adrese *Limit*.

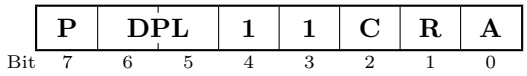
Hodnota bitu B má ještě jeden význam: je-li B=0, je implicitní velikost položky v zásobníku (vybírané instrukcí POP a zapisované instrukcí PUSH) 16bitové slovo, a je-li B=1, je implicitní velikost 32bitové dvojslovo.

Pro zachování kompatibility zdola je segment ED=1 a B=0 totožný se segmentem definovaným v procesoru Intel286.

5.2.4 Popisovač instrukčního segmentu

Instrukční segment obsahuje instrukce určené ke spuštění. V tabulce popisovačů segmentů se položka instrukčního segmentu liší od položky datového segmentu pouze

slabikou přístupových práv (viz obr. 5.6). Význam má také bit, který byl na obr. 5.3 označen jako *s*.



Obr. 5.6 Přístupová práva popisovače instrukčního segmentu

Bity 4 a 3 sdělují, že jde o popisovač instrukčního segmentu. Ostatní bity mají tento význam:

P a **DPL** viz popisovač datového segmentu.

C (Conforming) nulový sděluje, že podprogramy volané v tomto segmentu budou mít nastavenu úroveň oprávnění odpovídající úrovni segmentu, v němž se nacházejí. Je-li C=1, bude volanému podprogramu v tomto segmentu přidělena úroveň oprávnění segmentu, z něhož je volán (v textu segment označujeme jako „přizpůsobitelný“).

R (Readable) nulový zakazuje čtení obsahu segmentu. Je povoleno pouze obsah segmentu spustit. Jedničková hodnota bitu povoluje jak spuštění, tak i čtení segmentu (takový segment označujeme v textu jako „čitelný“).

A (Accessed) viz popisovač datového segmentu.

Bit *s* (viz obr. 5.3) je v popisovači datového segmentu označován D a má tento význam:

D (Default) sděluje implicitní velikost efektivních adres a operandů používaných v tomto segmentu. Je-li D=0, je implicitní velikost efektivních adres a operandů 16 bitů (odpovídá Intel286) a D=1 nastavuje implicitní velikost 32 bitů. V reálném režimu je vždy implicitní velikost 16 bitů.

Explicitní určení velikosti zajišťují instrukční prefixy 66h a 67h. Prefix **66h** mění implicitní velikost **operandu** a prefix **67h** mění implicitní velikost **adresy**. Jejich použití je patrné z obr. 5.7.

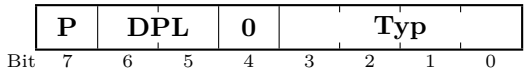
D=	0	0	0	0	1	1	1	1
Prefix 66h (vel. operandu)	ne	ne	ano	ano	ne	ne	ano	ano
Prefix 67h (vel. adresy)	ne	ano	ne	ano	ne	ano	ne	ano
Velikost operandu v bitech	16	16	32	32	32	32	16	16
Velikost adresy v bitech	16	32	16	32	32	16	32	16

Obr. 5.7 Použití instrukčních prefixů 66h a 67h v závislosti na bitu D

V reálném režimu není, ani po použití prefixu změny velikosti adresy, povoleno adresovat segmenty větší než 64 KB. Offset, který by překročil hodnotu FFFFh, způsobí výjimku 13.

5.2.5 Popisovač systémového segmentu

Popisovačů systémového segmentu je řada. Jsou to popisovače zvláštního určení. Jejich počet narostl zavedením 32bitových procesorů a udržováním kompatibility zdola. V tomto odstavci si typy pouze vyjmenujeme a budeme se k nim vracet později. Bit *s* (viz obr. 5.3) v popisovači systémového segmentu není použit.



Obr. 5.8 Přístupová práva popisovače systémového segmentu

Typ=0 … nepovolená hodnota,
1 … TSS neaktivního procesu Intel286 (str. 84),
2 … LDT všech procesorů (str. 51),
3 … TSS aktivního procesu Intel286 (str. 84),
4 … brána pro předání řízení Intel286 (str. 77),
5 … brána zpřístupňující TSS všech procesorů (str. 86),
6 … brána pro maskující přerušení Intel286 (str. 92),
7 … brána pro nemaskující přerušení Intel286 (str. 92),
8 … nepovolená hodnota,
9 … TSS neaktivního procesu od Intel386,
A … nepovolená hodnota,
B … TSS aktivního procesu od Intel386,
C … brána pro předání řízení od Intel386,
D … nepovolená hodnota,
E … brána pro maskující přerušení od Intel386,
F … brána pro nemaskující přerušení od Intel386.

Do systémového segmentu smí zapisovat pouze procesor. Programátor (operační systém) si pro účely modifikace obsahu musí přes systémový segment překrýt datový (viz odstavec 5.2.8).

Každá tabulka popisovačů segmentů je sama uložena v některém ze segmentů. Tabulek LDT může být v jednom okamžiku v paměti více. Tabulka popisovačů globálního adresového prostoru (GDT) je právě jedna. Pro uchování adresy jejího uložení je vyhrazen speciální registr **GDTR** (Global Descriptor Table Register).

Každá LDT je specifikována jedním z popisovačů v GDT. Poněvadž s právě aktivním procesem je svázána jedna LDT, je pro snadnější přístup k adrese uložení této LDT vyhrazen rovněž jeden speciální registr nazvaný **LDTR** (Local Descriptor Table Register). Obsah registru LDTR se změní vždy, když je přepnuto zpracování na jiný proces tak, aby v každém okamžiku byla v LDTR adresa LDT právě aktivního procesu.

5.2.6 Segmentové registry

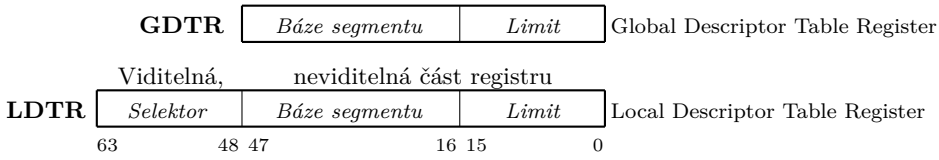
Segmentové registry CS, DS, ES, FS, GS a SS jsou v chráněném režimu složeny ze dvou částí: část programátorovi „viditelná“ obsahuje 16bitový selektor a část programátorovi „neviditelná“ obsahuje slabiku přístupových práv, bázi segmentu, limit segmentu apod.

Vždy při změně selektoru některého ze segmentových registrů použije procesor jeho hodnotu k výběru odpovídající položky z tabulky popisovačů a naplní neviditelnou část příslušnými hodnotami. Ta se potom použije při každém přístupu do paměti, protože se musí k bázi segmentu přičíst offset, zkontrolovat nepřekročení limitu a přístupová práva. Výhodné je mít tyto údaje umístěny v registrech proto, že doba přístupu do paměti (kde jsou tabulky popisovačů uloženy) je delší než do registru.

Obsah selektoru registrů SS, DS, ES, FS a GS nastavujeme instrukcemi typu MOV, LES, LDS atd. Obsah selektoru registru CS se plní instrukcemi JMP, CALL a RETF ve variantě vzdáleného skoku a volání. Při každé změně selektoru se také kontroluje úroveň oprávnění.

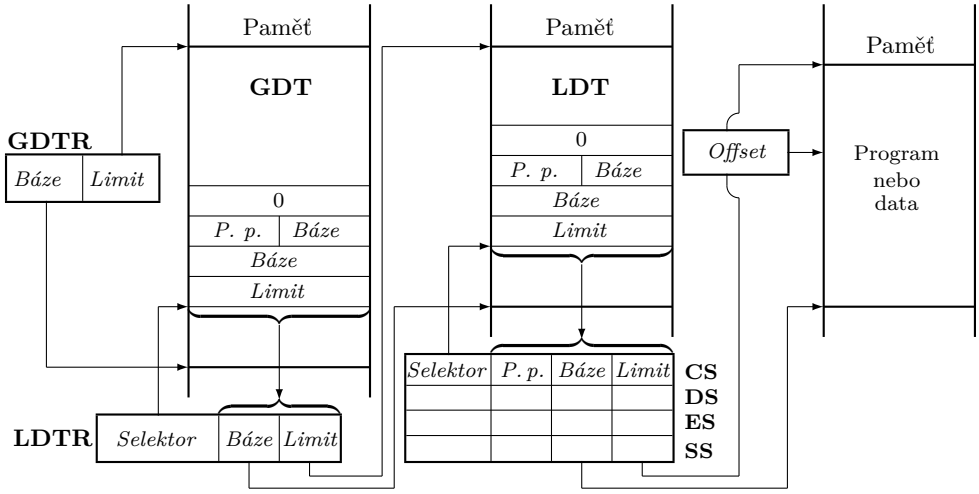
5.2.7 Registry GDTR a LDTR

Registr **GDTR** (Global Descriptor Table Register) uchovává adresu uložení tabulky popisovačů globálního adresového prostoru. GDTR obsahuje 4 slabiky báze segmentu s uloženou tabulkou a 2 slabiky limitu tohoto segmentu. Registr je přístupný pouze instrukcemi LGDT (Load GDT) a SGDT (Store GDT).



Obr. 5.9 Struktura registrů GDTR a LDTR

Registr **LDTR** (Local Descriptor Table Register) je složený ze dvou částí: z viditelného 16bitového selektoru a z neviditelné 48bitové části obsahující bázi segmentu s LDT a limit segmentu s LDT. Selektor LDTR (viditelná část) má stejný tvar jako v logické adrese a je přístupný pouze instrukcemi LLDT (Load LDT) a SLDT (Store LDT). Ukazuje na položku specifikující LDT v globální tabulce popisovačů. Při každé změně jeho obsahu procesor kontroluje, ukazuje-li selektor na popisovač systémového segmentu s LDT, a nastaví podle tabulky popisovačů nové hodnoty neviditelné části LDTR. Obsah celého registru LDTR se musí změnit také při každém přepnutí na jiný zpracovávaný proces. Použití GDTR, LDTR a segmentových registrů je ukázáno na obr. 5.10.



Obr. 5.10 Použití GDTR, LDTR a segmentových registrů

Registry GDTR, LDTR a dalšími systémovými ovlivňujeme prostředí pro běh aplikačních programů, a proto je přístup k těmto registrům z aplikačních programů nežádoucí. Z toho důvodu jsou v procesoru implementovány prostředky ochrany, kterými se kontroluje chování aplikací. Při pokusu aplikace o nedovolenou činnost se generuje výjimka.

5.2.8 Sdílení jednoho segmentu více popisovači

Na závěr části věnované segmentům a jejich popisovačům poznamenejme, že více popisovačů může v jednom okamžiku „ukazovat“ na jeden segment v paměti. Uveďme si dva příklady:

- Aplikační proces je uložen v instrukčním segmentu, který je v popisovači označen bitem R=0 (obsah segmentu se smí pouze spustit, nelze jej číst). Ladí-li programátor svůj aplikační program ladicím systémem, potřebuje tento systém instrukční segment číst i do něho zapisovat (označit ladicí body, atd.). Proto si ladicí systém vytvoří nový popisovač a segment označí jako datový s možností čtení i zápisu. Oba popisovače (instrukční pro aplikaci a datový pro ladicí systém) ukazují na stejnou oblast v lineární paměti.
- Jeden proces zapisující data do vyrovnávací paměti má v popisovači datového segmentu bit W=1 (do segmentu lze zapisovat). Jiný proces smí obsah této vyrovnávací paměti pouze číst, proto je mu nabídnut jiný popisovač s hodnotou W=0. Oba popisovače ukazují opět na stejnou datovou oblast.

Sdílení segmentu více popisovači používá také operační systém pro modifikaci svých systémových segmentů. Rovněž poznamenejme, že popisovače ukazující na stejné datové oblasti nemusejí mít stejné limity.

5.3 Systémové registry

V předchozích odstavcích jsme již popsali tyto registry: segmentové registry, GDTR a LDTR. Nyní si popíšeme příznaky registru EFLAGS, určené chráněnému režimu, a řídicí registry.

5.3.1 Příznakový registr s příznaky chráněného režimu

Příznakový registr **EFLAGS** obsahuje příznaky dvou základních skupin. Příznaky nastavované procesorem podle výsledku právě provedené operace byly již popsány na str. 23. V této kapitole nás budou zajímat příznaky určené pro řízení procesoru. Jde o tyto příznaky:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	**ID**	**VIP**	**VIF**	**AC**	**VM**	**RF**
0	**NT**	**IOPL**		**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	0	**AF**	0	**PF**	1	**CF**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Obr. 5.11 Příznakový registr EFLAGS procesoru Pentium

TF (Trap Flag) uvádí procesor do krokovacího režimu, ve kterém je po provedení první instrukce generována výjimka. Příznak nastavují různé ladicí systémy,

které tímto režimem po jednotlivých instrukcích krokují laděný program. Ladicí systém se aktivuje generovanou výjimkou a potom zjišťuje, co se během provádění instrukce změnilo. Příznak lze nastavit pouze přes zásobník instrukcí IRET.

IF (Interrupt Enable Flag) vynulovaný instrukcí CLI zabrání uplatnění vnějších maskovatelných přerušení (generovaných signálem INTR). Nastavení příznaku na jedničku (instrukcí STI) přerušení povoluje. Maskovat lze přerušení od vnějších zařízení (klávesnice, tiskárna atd.), nikoli výjimky, programová přerušení (INT) a NMI (přerušení ze skupiny nemaskovatelných přerušení). Hodnoty CPL a IOPL určují, zda lze tento příznak měnit instrukcemi CLI, STI, POPF, POPFD a IRET.

IOPL (I/O Privilege Level) určuje úroveň oprávnění, při které může proces ještě provádět V/V instrukce. Vyšší hodnota představuje nižší úroveň oprávnění.

NT (Nested Task) určuje režim práce instrukce IRET. Je-li NT=0, provádí IRET klasický návrat z přerušení. Je-li NT=1, přepne se při provádění IRET proces podle zpětného ukazatele právě aktivního TSS. Příznak se nastavuje instrukcemi POPF, POPFD a IRET. Chybné nastavení příznaku vede k neočekávaným výskytům výjimek v aplikačních programech (viz dále v této kapitole).

RF (Resume Flag) maskuje opakování ladicí výjimky. Příznakem se dočasně vypíná generování ladicí výjimky na to, aby se mohla instrukce po obsluze výjimky provést bez opakovaného přerušení. Ladicí program tento příznak nastavuje instrukcí IRETD. Příznak se nenastavuje instrukcemi POPF, POPFD a IRET. Viz kapitola 7.

VM (Virtual 8086 Mode) zapíná režim *virtuální 8086* pro proces, jemuž obsah příznakového registru náleží. Příznak VM smí programátor nastavovat pouze v chráněném režimu, a to instrukcí IRET, a jenom na úrovni oprávnění 0. Příznak je také modifikován mechanismem přepnutí procesu. Podrobnosti viz v kapitole 10 od str. 139.

AC (Alignment Check) spolu s bitem AM v registru CR0 zapíná generování přerušení při odkazu na paměť, který není „zarovnán“ na hranici odpovídající délce zpřístupňovaného objektu. Je-li AC=1 a je-li uskutečněn pokus o čtení nebo zápis např. 16bitového slova na lichou adresu, vyvolá se výjimka. Tato kontrola se provádí pouze na úrovni oprávnění 3 (tj. na uživatelské úrovni). Viz též str. 102.

VIF (Virtual Interrupt Flag) je virtuální obraz příznaku IF pro daný V86 proces.

VIP (Virtual Interrupt Pending Flag) se používá spolu s příznakem VIF ve V86 procesech.

ID (Identification Flag) oznamuje, zda procesor má ve svém repertoáru instrukci CPUID. Procesor instrukci umí zpracovat tehdy, pokud má program možnost tento bit nastavovat a nulovat.

Poznámky k předchozím procesorům

V 8086 obsahuje registr FLAGS (nebo též F) tyto příznaky: CF, PF, AF, ZF, SF, TF, IF, DF a OF. V procesoru Intel286 přibývají příznaky IOPL a NT. Od Intel386 je registr 32bitový, jmenuje se EFLAGS a přibývají příznaky RF a VM. V Intel486 přibývá příznak AC. Příznaky VIF, VIP a ID jsou vlastní až Pentiu.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PG	**CD**	**NW**											**AM**		**WP**
										NE	**ET**	**TS**	**EM**	**MP**	**PE**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Obr. 5.12 Řídicí registr CR0

5.3.2 Řídicí registr CR0

Základním registrem pro řízení procesoru je registr **CR0**. Tímto registrem se mj. zapíná a vypíná chráněný režim procesoru a stránkování. Jednotlivé bity CR0 mají následující význam (viz obr. 5.12):

PE (Protected Mode Enable) zapíná *chráněný režim*. Po inicializaci procesoru (signálem RESET) je zapnut *reálný režim*. Nastavením tohoto příznaku na jedničku je procesor přepnut do *chráněného režimu*, vynulováním zpět do *reálného režimu*.

MP (Monitor coProcessor) indikuje fyzickou přítomnost koprocesoru (FPU) a řídí činnost instrukce WAIT. V Intel486 a Pentiu by měl být nastaven na 1. V Intel486 SX by měl být nulový. V Intel286 a Intel386 by měl být nastaven podle toho, je-li v počítači instalován koprocesor (MP=1), nebo není (MP=0).

EM (Emulate coProcessor) zapíná programovou emulaci koprocesoru tehdy, není-li koprocesor instalován (viz popis výjimky 7 na str. 98).

TS (Task Switch) se nastavuje každým přepnutím procesu a testuje se při pokusu o provedení instrukce pro koprocesor. Je určen na vyvolání operace uložení kontextu koprocesoru při změně kontextu procesoru. Viz též str. 98. Bit nuluje instrukce CLTS.

ET (Extension Type) je v procesoru Pentium rezervován. Jinak sděluje typ instalovaného matematického koprocesoru. Bit nastavuje procesor během inicializace (po přijetí signálu RESET). Detekuje-li Intel287, nastaví ET=0, pro typ Intel387 a FPU Intel486 nastaví ET=1. Na základě bitu ET procesor používá buď 16bitový, nebo 32bitový protokol komunikace s koprocesorem. Bit lze nastavovat i programově.

NE (Numeric Error) sděluje, jak se mají procesoru oznamovat chyby zjištěné v koprocesoru. Je-li NE=1, bude se generovat výjimka 16. Je-li NE=0 a vstupní signál $\overline{\text{IGNNE}}$ je aktivní, potom se chyba ignoruje. Je-li NE=0 a signál $\overline{\text{IGNNE}}$ je neaktivní, potom chyba zastaví procesor, který bude čekat na přerušení. O přerušení procesor současně žádá výstupním signálem $\overline{\text{FERR}}$ připojeným k řadiči přerušení (signál $\overline{\text{FERR}}$ emuluje signál $\overline{\text{ERROR}}$ koprocesorů Intel287 a Intel387). Stav NE=0 je kompatibilní s předcházejícími typy koprocesorů a operačním systémem MS-DOS, který chyby přijímal přes vnější přerušení. Viz též popis výjimky 16 na str. 101.

WP (Write Protect) je-li jedničkový, zakazuje zápis do stránek označených W=0 i procesům na úrovni oprávnění CPL<3. Procesor Intel386, bylo-li W=0, zakazoval zápis pouze procesu s CPL=3 a procesy s CPL<3 měly zápis povolen. Kompatibilita vyšších typů s Intel386 je zachována při WP=0. Je-li WP=1, nemůže do stránky označené W=0 zapisovat žádný proces.

AM (Alignment Mask) maskuje náhodné nastavení příznaku AC v příznakovém registru (viz str. 60). Je-li AM=0, není funkce AC zapnuta ani tehdy, je-li AC=1. Je-li AM=1, záleží na hodnotě AC.

Maskování bitu AC je zavedeno proto, že programy přenesené z procesoru Intel386 by mohly do tohoto příznaku zavádět různé nedefinované hodnoty. Nastavením AC=0 zabráníme výskytům výjimek 17.

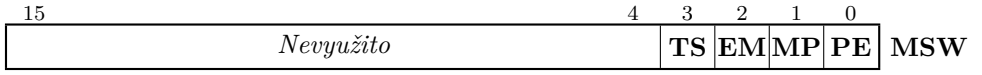
NW (Not Write-Through) je-li nulový, potom se všechny zápisy do paměti, jejichž položka je v interní vyrovnávací paměti, zapisují jak do IVP, tak do fyzické paměti. Ty zápisy, jejichž položka v IVP není, se provádějí pouze do fyzické paměti. Je-li NW=1, není zápisem do paměti změněn obsah IVP ani tehdy, máli adresa zapisovaného objektu svoji položku v IVP. Údaj se ukládá pouze do fyzické paměti. Po inicializaci procesoru signálem RESET je nastaveno NW=1.

CD (Cache Disable) zapíná nebo vypíná IVP. Je-li CD=1, je IVP vypnuta tak, že položky, které při čtení nebyly ve vyrovnávací paměti nalezeny, se do ní nezapisují. IVP je zapnuta, jsou-li současně splněny tyto podmínky: CD=0, $\overline{\text{KEN}}$=0 a PCD=0 (PCD je bit z CR3 nebo ze stránkové tabulky). Po inicializaci procesoru signálem RESET je nastaveno CD=1.

PG (Paging) zapíná stránkovou jednotku určenou k transformaci lineárních na fyzické adresy.

Poznámky k předchozím procesorům

V procesoru Intel286 byl na místě registru CR0 16bitový registr **MSW** (Machine Status Word – viz obr. 5.13). Ve vyšších typech procesorů se pod názvem MSW rozumí dolních 16 bitů registru CR0. V procesoru Intel286 nebylo možné nulovat bit PE. Jedinou možností, jak přepnout procesor Intel286 z chráněného režimu zpět do reálného, je inicializace (RESET) procesoru.



Obr. 5.13 Registr MSW procesoru 80286

Procesor Intel386 má v registru CR0 bity PE, MP, EM, TS, ET a PG. Procesor Intel486 má registr CR0 totožný s Pentiem.

5.3.3 Řídicí registr CR2

Registr **CR1** je rezervován pro využití budoucími typy procesorů. Registr **CR2**, je-li nastaven bit PG v CR0, obsahuje lineární adresu, která způsobila výpadek stránky detekovaný stránkovací jednotkou. Výpadek stránky má za následek generování výjimky 14. Registr je určen pouze ke čtení.

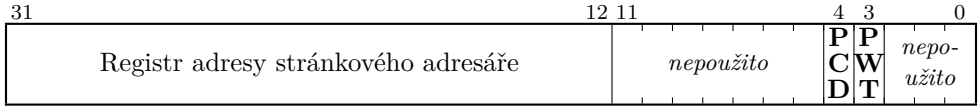
Poznámky k předchozím procesorům

Registry od CR2 jsou zavedeny od procesorů Intel386.

5.3.4 Řídicí registr CR3

Registr **CR3** je rovněž využit pouze při zapnuté stránkovací jednotce. Obsahuje fyzickou adresu stránkového adresáře právě aktivního procesu (Directory Base Address – DBA). Dolních 12 bitů se při zápisu do tohoto registru ignoruje, protože stránkový adresář smí začínat pouze na hranici 4 KB stránky. Z dolních 12 bitů registru CR3 jsou využity bity 3 a 4 (viz obr. 5.14).

Bity **PWT** (Page Write-Through) a **PCD** (Page Cache Disable) slouží k řízení vyrovnávacích pamětí (viz též str. 67). Hodnota těchto bitů se přenáší mimo procesor po stejnojmenných signálech tehdy, není-li zapnuto stránkování nebo se stránkování z nějaké příčiny obchází. V jiných případech se na výstup z procesoru předávají hodnoty bitů PWT a PCD ze stránkovacích tabulek odpovídající zpracovávané stránce.

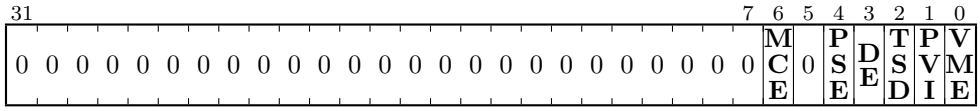


Obr. 5.14 Řídicí registr CR3

Poznámky k předchozím procesorům

Bity PWT a PCD byly do registru CR3 zavedeny v procesoru Intel486.

5.3.5 Řídicí registr CR4



Obr. 5.15 Řídicí registr CR4

Jednotlivé bity registru CR4 mají tento význam (viz též obr. 5.15):

MCE (Machine Check Enable) je-li nastaven na jedničku, je povoleno generování výjimky 18 (Machine Check Exception). Touto výjimkou se v procesoru Pentium oznamují chyby parity a jiné chyby technického vybavení. Viz též str. 103.

PSE (Page Size Extensions) zapíná používání stránek velikosti 4 MB při nastavení na 1. Viz odstavce 5.4.

DE (Debugging Extensions) je-li nastaven na 1, jsou povoleny ladicí body sledující přístup k V/V branám (viz str. 115).

TSD (Time Stamp Disable) nastavením tohoto příznaku se instrukce RDTSC (Read from Time Stamp Counter) stává privilegovanou. Instrukce je popsána ve zvláštních informačních dodatcích vydávaných firmou Intel.

PVI (Protected-Mode Virtual Interrupts) je-li nastaven na 1, umožňuje používat virtuální IF v chráněném režimu. Touto vlastností se některým programům určeným pro úroveň oprávnění 0 dovoluje běh i na úrovni oprávnění 3.

VME (Virtual-8086 Mode Extensions) je-li nastaven na jedničku, je povoleno používat virtuální obraz příznaku IF v režimu V86. Tato vlastnost zvyšuje rychlost provádění V86 aplikací eliminací řady přerušení aktivujících monitor V86 procesu.

Poznámky k předchozím procesorům

Registr CR4 zavádí až procesor Pentium.

5.4 Stránkování

Procesor Pentium poskytuje operačnímu systému dvě úrovně transformací adres. V první, vyšší úrovni je **logická adresa** mechanismem segmentační jednotky transformována na **lineární adresu** (viz obr. 5.2 na str. 51). Ve druhé úrovni je lineární adresa stránkováním transformována na **fyzickou adresu**. Fyzická adresa ukazuje již přímo do fyzické paměti.

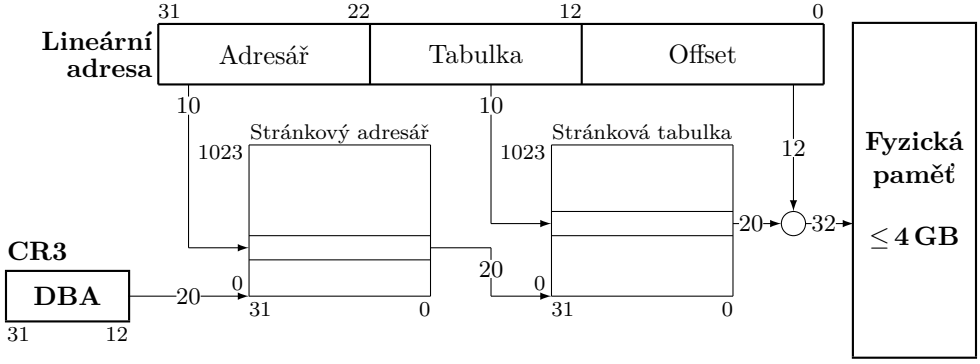
Stránkovací jednotka v procesoru realizuje mechanismus virtuální paměti, který dovoluje programům používat větší kapacitu paměti, než je skutečně instalovaná fyzická paměť. Stránkování rovněž dovoluje procesům mít vlastní adresový prostor. Celá kapacita virtuální paměti je uložena po **stránkách** v externím zařízení (např. na disku). Fyzická paměť se po **rámcích** propůjčuje na dočasné uložení stránek virtuální paměti. Kapacita stránky a rámce je stejná.

Propůjčování rámců fyzické paměti stránkám paměti virtuální provádí stránkovací jednotka v kombinaci s programovou podporou. Stránkovací jednotka provádí přepočet lineární adresy na číslo rámce, kde je momentálně stránka ve fyzické paměti umístěna. Pokud požadovaná stránka není v žádném z rámců, nastává tzv. **výpadek stránky**. Při rozpoznání výpadku stránky se generuje výjimka „Page Fault“, která aktivuje programovou rutinu zajišťující uvolnění rámce a přenos požadované stránky do uvolněného rámce. Stránkovací jednotka dále zajišťuje stránkovou ochranu na základě přístupových práv.

Segmentování je povinnou částí zpracování adresy, naopak stránkování je volitelné. Stránkování je zapnuto jenom tehdy, je-li bit PG v CR0 nastaven na jedničku. Stránkování lze zapnout pouze v rámci chráněného režimu. Zapnuté musí být tehdy, potřebujeme-li realizovat virtuální paměť pomocí stránkování, spouštět více než jeden virtuální 8086 proces a provádět stránkově orientovanou ochranu paměti. Mechanismus transformace lineární adresy na fyzickou je na obr. 5.16.

Pro účely stránkování dělíme vstupní 32bitovou lineární adresu do tří částí. První část (10 bitů nejvyšších řádů) nazvěme **adresář**, protože ukazuje do tabulky pojmenované **stránkový adresář** (Page Directory). Stránkový adresář je v daném okamžiku v systému právě jeden a smí začínat na adrese dělitelné 4 K. Fyzická adresa začátku stránkového adresáře (DBA) je uložena v registru **CR3**.

V CR3 není lineární adresa, ale fyzická, protože tento registr ukazuje na stránkový adresář, a tudíž jím nemůže být mapován. Stránka, ve které je umístěn stránkový adresář, může být mechanismem virtualizace paměti odložena do vnější paměti až



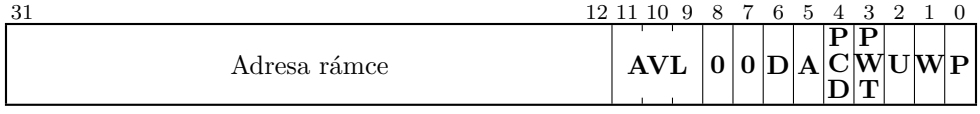
Obr. 5.16 Transformace lineární adresy na fyzickou pomocí stránkového adresáře a stránkové tabulky

tehdy, není-li proces, který adresář používá, aktivní. Každý proces má vlastní stránkový adresář (CR3 je uloženo v TSS). Operační systém musí před aktivací procesu zajistit, aby stránkový adresář, který bude proces používat, byl ve fyzické paměti. CR3 totiž neobsahuje bit P (Present), jehož nulová hodnota by vyvolala výjimku výpadku stránky.

Vybraná položka stránkového adresáře ukazuje na začátek **stránkové tabulky** (Page Table), kterých může být až 1 024. Stránkové tabulky opět musí být uloženy od adres dělitelných 4 K. Položka stránkové tabulky, vybraná částí lineární adresy nazvanou **tabulka**, ukazuje na začátek 4 KB rámce ve fyzické paměti. Ukazatelem do vybraného rámce je poslední 12bitová část lineární adresy **offset**.

Fyzická paměť je tedy rozdělena na 4 KB rámce, které začínají na adresách dělitelných 4 K a které se propůjčují pro uložení 4 KB stránek.

Stránkový adresář a **stránková tabulka** jsou pole obsahující 1 024 32bitových specifikátorů. Kapacita každé z těchto tabulek je právě stránka. Formát specifikátorů obou tabulek je na obr. 5.17. Jednotlivé bity mají tento význam:



Obr. 5.17 Tvar specifikátoru stránkového adresáře a stránkové tabulky

Adresa rámce je horních 20 bitů fyzické adresy rámce, na který položka ukazuje. Dolních 12 bitů se doplní nulami. Ve stránkovém adresáři ukazuje na stránkovou tabulku, ve stránkové tabulce ukazuje na 4KB stránku s požadovaným objektem v paměti.

AVL (Available) jsou bity určené pro volné použití operačním systémem.

D (Dirty) nastavuje procesor na jedničku před změnou obsahu rámce, na který specifikátor právě ukazuje. Ve stránkovém adresáři je tento bit nedefinován, procesor jej nenastavuje. Hodnotu bitu používá operační systém tehdy, potřebuje-li tento rámec vyprázdnit. Je-li D=1, musí se obsah rámce zapsat zpět do externí paměti (na disk).

A (Accessed) nastavuje procesor na jedničku před každým použití tohoto specifikátoru. Bit má stejný význam jako v popisovači segmentů (viz str. 53).

U (User Accesible) je určen pro stránkovou ochranu paměti. Pracuje-li proces na úrovni oprávnění CPL=3, smí k této stránce přistupovat pouze tehdy, je-li U=1. Procesy s CPL<3 smějí přistupovat ke všem stránkám bez ohledu na hodnotu bitu U.

W (Writeable) je určen rovněž pro stránkovou ochranu paměti. Pracuje-li proces na úrovni CPL=3, smí do této stránky zapisovat jenom tehdy, je-li W=1. Procesy s CPL<3 smějí zapisovat do všech stránek bez ohledu na hodnotu bitu W. Viz též popis bitu WP registru CR0 na str. 62.

P (Present) jedničkový označuje, že obsah specifikátoru je platný a položku lze použít ke transformaci adresy. Je-li P=0, není obsah odpovídající stránky ve fyzické paměti. Potom se položka nepoužívá a s jejím obsahem (vyjma bitu P) může operační systém disponovat na jiné účely (např. uchovat do ní informaci o tom, kde na disku je stránka momentálně uložena). Pokus o přístup ke stránce, která má nastaveno P=0 (ve stránkovém adresáři nebo stránkové tabulce), je výpadek stránky a vyvolá výjimku 14.

PWT (Page Write-Through) určuje způsob práce externí vyrovnávací paměti týkající se této stránky. Je-li PWT=1, provádí se zápis metodou „Write-Through“. Jde o způsob, ve kterém se zapisovaná slabika současně ukládá jak do vyrovnávací paměti, tak i do paměti. Je-li PWT=0, provádí se zápis metodou „Write-Back“, ve které se slabiky zapisují pouze do vyrovnávací paměti. Změněné položky se z vyrovnávací paměti přepíší do paměti až při jejím vyprázdňování.

PCD (Page Cache Disable) vypíná činnost interní VP pro danou stránku. Je-li PCD=0, je splněna jedna z podmínek zapínajících IVP. Další podmínky tvoří signál $\overline{\text{KEN}}$ a bity CD a NW v registru CR0. Je-li PCD=1, je IVP vypnuta bez ohledu na ostatní podmínky.

Stav bitů PWT a PCD je při přístupu k paměti přenášen mimo procesor signály PWT a PCD. Není-li stránkování zapnuto (PG=0) nebo je obcházeno z jiné příčiny, přenášejí se po signálech PWT a PCD stejnojmenné bity z registru CR3. Je-li stránkování zapnuto (PG=1) a je-li právě plněn stránkový adresář, čtou se bity PWT a PCD z CR3. Bity PWT a PCD konkrétního specifikátoru ze stránkového adresáře se čtou při plnění stránkové tabulky. V ostatních případech (tj. při přístupech ke všem ostatním stránkám fyzické paměti) se použijí bity PWT a PCD ze specifikátoru stránkové tabulky.

Pracuje-li proces na úrovni oprávnění CPL<3, může každou stránku číst a do každé stránky zapisovat bez ohledu na nastavení bitů U a W v případě, že bit WP registru CR0 (viz str. 62) je nulový. Jde-li o uživatelský proces (CPL=3), smí číst obsah stránek označených U=1 a zapisovat do stránek označených U=1 a zároveň W=1. Vyhodnocení stránkových ochran se provádí až na druhém místě, tj. až po zkontrolování legálnosti přístupu k segmentu. Kompletní schéma vyhodnocení je uvedeno na obr. 5.18. Zopakujme ještě, že bit WP existuje až od procesoru Intel486.

U	**W**	**WP**	Proces CPL=3	Proces CPL<3
0	0	0	nepřístupná	čtení, zápis, provedení
0	1	0	nepřístupná	čtení, zápis, provedení
1	0	0	čtení, provedení	čtení, zápis, provedení
1	1	0	čtení, zápis, provedení	čtení, zápis, provedení
0	0	1	nepřístupná	čtení, provedení
0	1	1	nepřístupná	čtení, zápis, provedení
1	0	1	čtení, provedení	čtení, provedení
1	1	1	čtení, zápis, provedení	čtení, zápis, provedení

Obr. 5.18 Kombinace bitů stránkové ochrany

Bity U a W v položce stránkové tabulky se vztahují na stránku, na kterou tato položka stránkové tabulky ukazuje. Bity U a W v položce stránkového adresáře platí pro všechny stránky adresované stránkovou tabulkou, na kterou tato položka stránkového adresáře ukazuje (nevztahují se tedy na stránkovou tabulku jakožto na stránku v paměti). Poněvadž jsou pro každou stránku ve fyzické paměti vyhrazeny dvě dvojice bitů U a W, je stanoveno pravidlo, podle něhož se použije typ ochrany mající nižší numerickou hodnotu. Pro stanovení numerické hodnoty se bity U a W chápou jako dvě binární číslice UW.
Příklad: Je-li U a W ve stránkovém adresáři 10 (CPL=3 smí číst a provádět) a ve stránkové tabulce 01 (pro CPL=3 nepřístupné), vybere se varianta U=0 a W=1.

Na závěr poznamenejme, že není žádný vztah mezi hranicemi stránek a hranicemi segmentů. Stránka má velikost právě 4 KB a začíná na adrese dělitelné 4 K. Segment

začíná na libovolné adrese a má velikost max. 4 GB. Je věcí operačního systému, jak bude informace v paměti organizovat.

Procesor Pentium zavádí možnost používání **4MB stránek**. Tato funkce se zapíná bitem PSE v registru CR4 (viz str. 64). Zapnutím 4MB stránek se vyřadí stránková tabulka a v činnosti zůstává pouze stránkový adresář. Dolních 22 bitů lineární adresy (viz obr. 66) je offsetem do 4MB stránky. Význam bitů položky stránkové tabulky se přesouvá do stránkového adresáře.

Poznámky k předchozím procesorům

Stránkování je zavedeno od procesoru Intel386.

5.4.1 TLB

Protože při transformaci lineární adresy na fyzickou se musí přistupovat ke dvěma tabulkám umístěným ve fyzické paměti (a tento přístup trvá poměrně dlouho), má procesor implementovánu **TLB** (Translation Look-aside Buffer), která je vyrovnávací pamětí pro posledně transformované adresy včetně příznaků.

V procesoru Pentium jsou integrovány dvě TLB. Jedna je součástí instrukční vyrovnávací paměti a druhá je uvnitř datové vyrovnávací paměti. TLB v datové VP je dvouportová. To jí dovoluje současně transformovat adresy ze dvou zdrojů. Instrukční TLB je jednoportová.

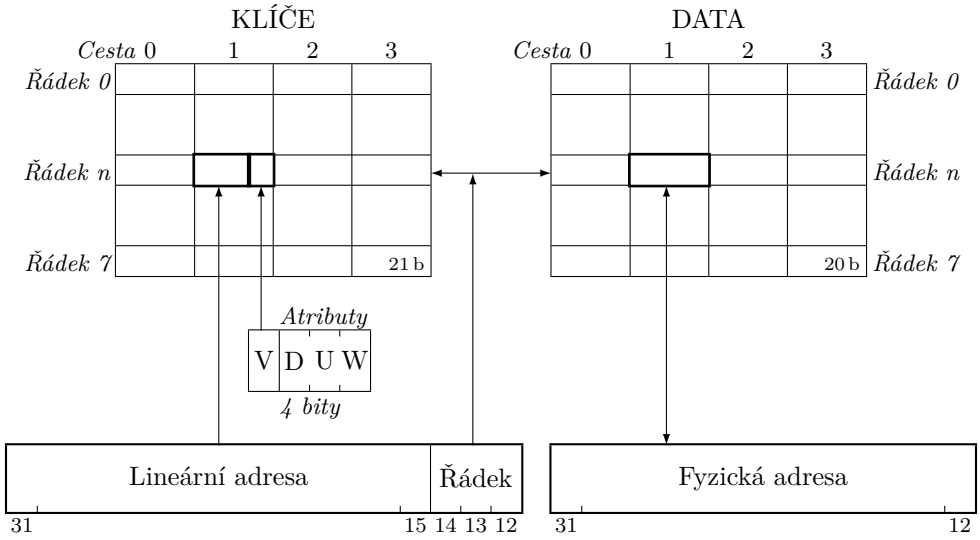
Datová TLB je 4cestná asociativní paměť o 64 položkách pro 4KB stránky a 4cestná asociativní paměť o 8 položkách pro 4MB stránky. Instrukční TLB je 4cestná asociativní paměť o 32 položkách pro 4KB stránky a pro 4MB stránky ve 4KB násobcích. Četnost přístupů k jednotlivým položkám se sleduje 3bitovým algoritmem LRU implementovaným ve vyrovnávacích pamětech procesoru Intel486 (viz str. 126). Obsahy TLB jsou chráněny paritou.

Poznámky k předchozím procesorům

Procesory Intel386 a Intel486 mají pouze jednu TLB a poměrně podrobně dokumentovanou. Proto si některé podrobnosti uveďme:

TLB Intel386 a Intel486 je 4cestná asociativní paměť obsahující 4×8 klíčů. 21bitový klíč zahrnuje lineární adresu a atributy. Ke každému klíči je přiřazeno 20 bitů obsahujících fyzickou adresu. Poněvadž se stránkovací jednotkou transformují adresy 4 KB stránek, je nejnižších 12 bitů nevýznamných. Struktura TLB Intel386 a Intel486 je patrná z obr. 5.19.

Lineární adresa je při prohledávání TLB rozdělena na tři části. První částí je 12 nevýznamných bitů nejnižších řádů. Druhou částí jsou bity 14 až 12, podle kterých se vybere jeden z řádků 0 až 7. Ve vybraném řádku se potom testuje zbytek lineární



Obr. 5.19 Struktura TLB Intel386 a Intel486

adresy na shodu s obsahem jedné ze čtyř cest. Při testování se uplatní rovněž atributy D, U, W a V=1. Je-li lineární adresa nalezena, předá se obsah datové cesty, která svou polohou odpovídá klíči.

Při zápisu originálu a transformované adresy do TLB se podle lineární adresy (tj. originálu) vybere řádek a na tomto řádku se testuje, má-li některá cesta nastaveno V=0. Pokud ano, zapíše se nová hodnota klíče (tj. lineární adresa, atributy D, U, W a V=1) a jemu příslušející data (tj. fyzická adresa). Pokud jsou všechny cesty na daném řádku obsazeny (všechny mají V=1), vybere se jedna položka (algoritmus výběru pro Intel386 výrobce nesděluje) a ta se vyřadí.

TLB je běžně obsluhována stránkovací jednotkou bez účasti programátora. Dále popsané **testovací registry** jsou určeny pouze pro testovací účely. Jde o příkazový registr **TR6** a datový registr **TR7**. S testovacími registry lze pracovat speciální variantou instrukce MOV v reálném i chráněném režimu. Pokud se TLB testuje v chráněném režimu, musí být vypnuto stránkování (PG=0) a proces, který pracuje s testovacími registry, musí mít CPL=0. Formát testovacích registrů je na obr. 5.20.

Pomocí registrů lze provádět operace: plnění TLB (lineární adresa z TR6 a fyzická adresa z TR7) a prohledávání TLB (k lineární adrese v TR6 se do TR7 zapíše fyzická adresa).

31 — 12	11	10	9	8	7	6	5	4	3 — 2	1	0	
Fyzická adresa	0	0	0	0	0	0	0	**H**	**RP**	0	0	**TR7**
Lineární adresa	**V**	**D**	$\overline{\mathbf{D}}$	**U**	$\overline{\mathbf{U}}$	**W**	$\overline{\mathbf{W}}$	0	0	0	**C**	**TR6**

Obr. 5.20 Testovací registry TR6 a TR7 procesoru Intel386

Registr TR6 obsahuje tato pole:

C (Command) je-li nulové, jde o plnění TLB lineární adresou z TR6 a fyzickou z TR7. Je-li jedničkové, jde o prohledávání TLB (hledá se lineární adresa podle TR6).

W, **U**, **D** viz popis stránkového adresáře a stránkové tabulky. Hodnoty $\overline{\text{W}}$, $\overline{\text{U}}$, $\overline{\text{D}}$ jsou inverzní k W, U, D.

V (Valid) jedničkové znamená, že obsah této položky je platný (při inicializaci nebo vyprázdnění TLB je všem položkám nastaveno V nulové).

Lineární adresa je vyšších 20 bitů lineární adresy začátku 4 KB stránky.

Registr TR7 obsahuje pole:

Fyzická adresa je vyšších 20 bitů fyzické adresy začátku 4 KB stránky. Při zápisu se tato hodnota spolu s lineární adresou v TR6 uloží do TLB. Při prohledávání je sem uložena platná fyzická adresa, je-li H=1. Pokud se lineární adresa z TR6 nenašla, je H=0 a obsah položky „fyzická adresa“ je nedefinován.

H (Hit) při prohledávání se H nastaví podle výsledku operace: H=1 při nalezení lineární adresy, H=0 při nenalezení. Při zápisu H sděluje, zda má být zapisovaná informace uložena do cesty TLB vybrané procesorem (H=0), nebo programátorem (H=1). Poněvadž Intel386 neumí při zápisu testovacími registry sám vybírat cestu, **musí** být nastaveno H=1 a v RP číslo cesty.

RP při prohledávání sděluje, ve které ze čtyř cest je informace uložena. Při zápisu musí být RP nastaveno na číslo cesty.

Při používání TLB je vhodné dodržet následující algoritmy. Pro zápis informace jsou doporučeny tyto kroky:

1. V pomocném registru (např. v EAX) vytvoříme obsah registru TR7: zapíšeme fyzickou adresu, do bitu H nastavíme jedničku a do pole RP číslo cesty TLB. Obsah pomocného registru opíšeme do registru TR7 instrukcí `MOV TR7,EAX`.
2. Naplnění TR6. Zapíšeme lineární adresu a nastavíme V=1 (zapisujeme-li platný obsah), nastavíme bity D, U, W podle typu zpřístupňované stránky a nastavíme bity $\overline{\text{D}}$, $\overline{\text{U}}$ a $\overline{\text{W}}$ na hodnoty inverzní k D, U, W. Nastavíme C=0.

3. Tím byla zapsána nová položka do TLB. Zápis lze opakovat libovolněkrát.

Je-li zapsáno několik položek do TLB, můžeme TLB prohledávat. Čtení TLB by mělo proběhnout v těchto krocích:

1. Naplnění TR6. Do registru TR6 uložíme horních 20 bitů hledané adresy stránky. Bit V by měl být nastaven na jedničku, pokud hledáme platnou položku TLB. Bit C musí být nastaven na jedničku. Ostatní atributy (D, $\overline{\text{D}}$, U, $\overline{\text{U}}$, W, $\overline{\text{W}}$) musí být nastaveny na hodnoty odpovídající hledané položce, protože jsou spolu s lineární adresou a bitem V součástí prohledávací masky.

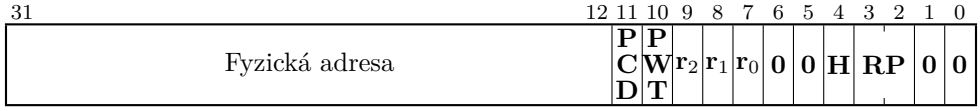
   Existence přímých a inverzních hodnot bitů D, U, W má ten význam, že můžeme do prohledání zahrnout i předem neznámou hodnotu některého z těchto bitů. Jsou-li např. hodnoty D a $\overline{\text{D}}$ obě jedničkové, potom se při hledání na obsah bitu D nebere ohled (všechny položky budou mít bit D vyhovující masce). Jsou-li obě hodnoty D a $\overline{\text{D}}$ nulové, nebude podmínka splněna nikdy a nenajde se žádná vyhovující položka (tj. prohledávání TLB skončí vždy neúspěšně). Stejným způsobem TLB pracuje s bity U a W.
2. Přečtení TR7. Byl-li logikou TLB nastaven bit H, bylo prohledávání TLB úspěšné. Potom je v TR7 uloženo 20 bitů fyzické adresy a v poli RP číslo cesty TLB, ve které je tato položka zapsána. Je-li H nulové, položka se nenalezla a zbytek registru TR7 má nedefinovaný obsah.

Pokud se na jednom řádku vyskytnou alespoň dvě stejné lineární adresy, potom prohledávání TLB s takovou lineární adresou je neúspěšné.

Při každé změně obsahu registru CR3 (přepnutím procesu nebo přímým naplněním CR3) je TLB automaticky vyprázdněno. Po vyprázdnění mají všechny položky nastaveno V=0. TLB musíme vyprázdnit i tehdy, provedeme-li změny uvnitř tabulek. Násilně TLB vyprázdníme tak, že znovu naplníme (i když stejnou hodnotou) registr CR3. Další důvod k vyprázdnění TLB je nastavení P=0 u některé z položek stránkovacích tabulek, tj. při odložení obsahu některého rámce na disk. Pokud by zůstal v TLB obsah podle původních tabulek, docházelo by k chybným transformacím adres.

Mechanismus TLB byl v procesoru Intel486 rozšířen následovně: Spolu s adresami a dalšími informacemi o stránce jsou uloženy i bity PCD a PWT. Je implementován algoritmus pseudo-LRU na vylučování nejdéle nepoužité položky ze zaplněné paměti tehdy, je-li potřeba zavést položku další. Algoritmus pseudo-LRU pro TLB je shodný s algoritmem pseudo-LRU pro IVP, který je popsán na str. 127. Schéma TLB Intel486 je shodné se schématem TLB Intel386 (viz obr. 5.19 na str. 70) s tou výjimkou, že byly pro každý řádek přidány 3 rozhodovací bity r_0, r_1 a r_2 (podobně jako na obr. 8.3). TLB 80486 má stále 8 řádků a 4 cesty.

Oproti Intel386 je změněn datový testovací registr **TR7**. Byly přidány bity PCD, PWT (popis viz str. 67) a 3 rozhodovací bity LRU označené r_0, r_1 a r_2 (viz obr. 5.21).



Obr. 5.21 Testovací registr TR7 procesoru Intel486

Při zápisu do TLB se v položce nastavují bity PCD a PWT tak, jak jsou v TR7. Na bity r_0 až r_2 se při zápisu nebere ohled. Při úspěšném prohledání TLB jsou vráceny všechny hodnoty PCD, PWT, r_2, r_1 a r_0.

Bit H po prohledávání TLB sděluje, byla-li položka nalezena, či nikoli. Oproti Intel386 má bit význam i při zápisu (v Intel386 musel být jedničkový). Je-li při zápisu H=1, je položka zapsána do cesty určené polem RP v TR7. Je-li při zápisu H=0, je obsah RP nevýznamný a cesta pro umístění položky se vybere algoritmem pseudo-LRU.

Pro selektivní rušení položek TLB byla do instrukčního repertoáru Intel486 zavedena instrukce INVLPG (Invalidate TLB Entry), která zruší (nastaví V=0) položku, jejíž klíč odpovídá operandu.

Na rozdíl od Intel386 při testování TLB není nutno v chráněném režimu vypínat stránkování. Potom však musíme poznamenat, že každé testovací čtení nebo zápis mají vliv na nastavování bitů LRU, stejně jako při normálním používání TLB. To znamená, že při každém testovacím přístupu k určité položce TLB se nastavují rozhodovací bity na řádku tak, aby signalizovaly, že tato položka byla právě použita.

5.5 Systém ochran

Pro činnost víceúlohového operačního systému je velmi důležitý dokonalý systém ochran. Ten v procesorech Intel mj. zajišťuje: izolaci systémového od uživatelského programového vybavení, vzájemnou izolaci jednotlivých uživatelů (procesů) a kontrolu typů dat a jejich použití (data nemohou být spuštěna, program nelze modifikovat atd.).

5.5.1 Úrovně oprávnění

Úroveň oprávnění (Privilege Level) vyjadřuje kvantitu „důvěry“ poskytnuté určitému procesu. Např. procesy s nižší úrovní oprávnění nesmějí mít možnost zasahovat

do procesů nebo dat s vyšší úrovní oprávnění. Procesor poskytuje **4 úrovně oprávnění**. Nejvyšší úroveň oprávnění (tj. nejvíce „důvěry“) má úroveň číslo 0. Nejmenší úroveň oprávnění má úroveň číslo 3. Rozdělení těchto čtyř úrovní mezi jednotlivé vrstvy operačního systému může být následující:

úroveň 0 … jádro operačního systému (řízení procesoru, V/V operací),

úroveň 1 … služby poskytované operačním systémem (plánování procesů, organizace V/V, přidělování prostředků),

úroveň 2 … systémové programy a podprogramy z knihoven (systém obsluhy souborů, správa knihoven),

úroveň 3 … uživatelské aplikace.

Proces je sestaven z instrukcí a dat, které jsou uloženy v instrukčním a datovém segmentu. Každému segmentu je přiřazena jistá úroveň oprávnění vztahující se na jeho obsah, tedy na program nebo data v něm uložená. Úroveň oprávnění přidělená segmentu je uložena v popisovači segmentu.

V dalším textu se setkáme se čtyřmi druhy indikátorů, obsahujících úrovně oprávnění:

DPL (Descriptor Privilege Level) je uložen ve dvou bitech slabiky **přístupová práva** popisovače segmentu. Obsahuje úroveň oprávnění přidělenou obsahu segmentu.

CPL (Current Privilege Level) je zapsán ve dvou nejnižších bitech **selektoru CS** (tj. v poli označeném RPL). Představuje momentální úroveň oprávnění přidělenou právě prováděnému procesu.

RPL (Requestor Privilege Level) je uložen v bitech 0 a 1 **selektoru segmentového registru** a obsahuje úroveň oprávnění, kterou proces nabízí při přístupu k segmentu, jehož popisovač je adresován právě tímto selektorem. Procesor použije numerické maximum z čísel RPL, CPL. Z toho plyne, že proces může nabídnout pouze nižší úroveň oprávnění, než je jeho uložená v CPL.

EPL (Effective Privilege Level) je numerické maximum CPL a RPL (tedy hodnota nižší úrovně oprávnění).

5.5.2 Zpřístupnění datového segmentu

Procesu se přístup k datům umístěným v datovém segmentu povolí tehdy, je-li úroveň oprávnění procesu rovna nebo vyšší než úroveň oprávnění zpřístupňovaného datového segmentu. Numericky musí platit vztah $\mathbf{CPL} \leq \mathbf{DPL}$. Dále musí být úroveň oprávnění v RPL větší nebo rovna úrovni zpřístupňovaného segmentu ($\mathbf{RPL} \leq \mathbf{DPL}$). Obě podmínky spojíme v jednu

Max(CPL,RPL) ≤ DPL.

Využijeme-li pojmu EPL, můžeme zapsat **EPL ≤ DPL**. Kontrola oprávněnosti přístupu (EPL ≤ DPL) se provede vždy, když je naplněn segmentový registr např. instrukcí `MOV DS,AX`. Pomocí indikátoru RPL (který byl před provedením instrukce uložen do AX) lze uměle snížit úroveň oprávnění, kterou proces nabízí při přístupu k tomuto datovému segmentu.

První položku GDT procesor nepoužívá. Selektor, který má index a TI nulové, se nazývá **neplatný selektor** (Null Selector). Lze jím naplnit segmentový registr (mimo CS a SS), aniž by procesor generoval výjimku. Procesor bude tuto výjimku generovat až při pokusu o použití segmentového registru, který obsahuje neplatný selektor. Této vlastnosti se využívá v okamžiku, kdy tabulky popisovačů nejsou ještě kompletně sestaveny, nebo tehdy, potřebujeme-li, aby obsah segmentového registru byl neplatný.

5.5.3 Předání řízení do instrukčního segmentu

Proces smí předat řízení pouze do segmentu se stejnou úrovní oprávnění, musí tedy platit **CPL = DPL**, tj. úroveň oprávnění běžícího procesu (CPL) se rovná úrovni oprávnění segmentu, do kterého se předává řízení (DPL).

Zvláštní případ je předávání řízení do segmentu, který je přizpůsobitelný (ve slabice přístupových práv má nastaven bit C na jedničku). Tento segment musí mít stejnou nebo vyšší úroveň oprávnění, než má běžící proces, tj. CPL ≥ DPL. Přizpůsobitelný segment se potom provádí na úrovni oprávnění volajícího procesu, tj. CPL. Hodnota CPL v registru CS zůstane zachována, i když je DPL menší.

Bez použití dalších prostředků (volání podprogramů na jiné úrovni oprávnění pomocí brány) lze instrukcemi CALL, JMP a RETF předávat řízení přímo pouze do:

- segmentu se stejnou úrovní oprávnění, jako je CPL;
- přizpůsobitelného (C=1) segmentu na stejné nebo vyšší úrovni oprávnění, tj. DPL ≤ CPL.

5.5.4 Předání řízení do instrukčního segmentu pomocí brány

Brány (Gate, Call Gate) lze použít při volání podprogramu umístěného v segmentu s vyšší úrovní oprávnění, než jakou má volající segment. Volání podprogramu pomocí brány lze uskutečnit jenom tehdy, platí-li numerický vztah **CPL ≥ DPL** volaného podprogramu. Z toho vyplývá, že v žádném případě nelze předávat řízení do segmentu majícího nižší úroveň oprávnění než volající (tedy ani pomocí brány).

Tato podmínka musí být dodržena proto, aby např. jádro operačního systému případným voláním podprogramu nižší úrovně oprávnění, do něhož mohlo být neautorizovaně zasáhnuto, nezpůsobilo zhroucení celého systému.

Shrňme si čtyři možnosti předávání řízení:

1. uvnitř jednoho segmentu (krátký a blízký JMP, blízké CALL a RET), kontroluje se pouze nepřekročení limitu segmentu;
2. mezi segmenty na stejné úrovni oprávnění (vzdálený JMP, CALL a RETF),
3. mezi segmenty na stejné úrovni oprávnění (vzdálený JMP, CALL a RETF) přes bránu,
4. mezi segmenty na různých úrovních oprávnění (vzdálené CALL a RETF) přes bránu.

5.5.5 Brány

Brána[1] (Gate) je popisovač uložený v tabulce popisovačů segmentů čtyř možných významů:

1. brána pro předání řízení do segmentu vyšší úrovně oprávnění (Call Gate),
2. brána pro nemaskující přerušení (Trap Gate),
3. brána pro maskující přerušení (Interrupt Gate) – obě popsány na str. 92,
4. brána zpřístupňující segment stavu procesu (Task Gate) – popsána na str. 85.

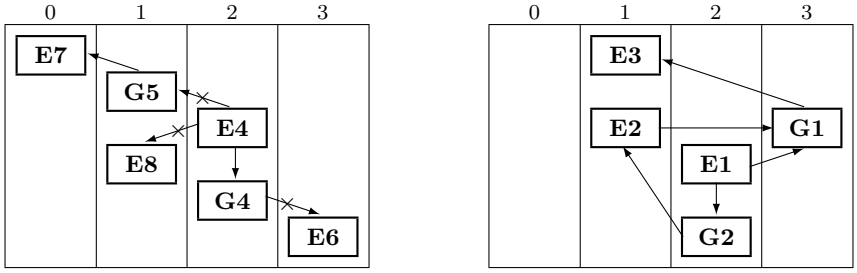
5.5.6 Brána pro předání řízení

Brána pro předání řízení do segmentu vyšší (nebo stejné) úrovně oprávnění (Call Gate) může být buď v GDT, nebo LDT. Jakmile se proces na tuto bránu odkáže, procesor zkontroluje, má-li proces dostatečnou úroveň oprávnění pro volání této brány. Při odkazu procesu na bránu platí stejná pravidla jako při odkazech procesu na data, tj. proces musí mít vyšší nebo stejnou úroveň oprávnění jako brána. Numericky tento vztah zapíšeme CPL $\leq$ DPL brány. Zároveň pole RPL popisovače odkazujícího se na bránu musí indikovat vyšší úroveň oprávnění než volaný segment, tj. RPL $\leq$ DPL brány. Obě pravidla spojíme do podmínky **MAX(CPL,RPL) $\leq$ DPL brány**.

Ovšem (podle odstavce 5.5.4) musí také platit, že úroveň oprávnění volaného podprogramu musí být stejná nebo vyšší než volajícího, tj. **CPL $\geq$ DPL podprogramu**. Instrukcí JMP lze bránou předat řízení pouze do segmentu na stejné úrovni oprávnění nebo do přizpůsobitelného segmentu na vyšší úrovni oprávnění. Instrukcí

[1] Pozor na záměnu pojmů V/V brána (Port) a brána pro předávání řízení do segmentů na jiné úrovni oprávnění (Call Gate).

CALL lze bránou předat řízení do libovolného segmentu na vyšší úrovni oprávnění. U obou instrukcí musí být navíc splněna podmínka MAX(CPL,RPL) ≤ DPL brány.

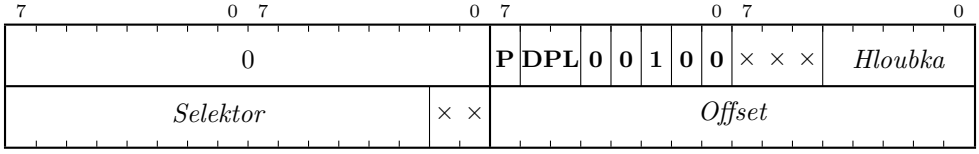


Obr. 5.22 Příklad předávání řízení pomocí bran

Na obr. 5.22 je několik příkladů volání bran a instrukčních segmentů prostřednictvím těchto bran. Objekt označený „**G**“ představuje bránu a „**E**“ instrukční segment. Čísla v horní řadě označují oblasti jednotlivých úrovní oprávnění. V tomto odstavci byla vysvětlena správnost volání bran E1-G2, E1-G1, E2-G1 a E4-G4. Rovněž bylo vysvětleno, proč je volání E4-G5 nesprávné. V odstavci 5.5.4 bylo vysvětleno volání segmentu bránou. Bylo uvedeno, že správné volání je takové volání, kde CPL ≥ DPL volaného segmentu. Z toho vyplývá správnost volání G2-E2, G1-E3 a G5-E7. Nesprávné je volání G4-E6.

Jako typický příklad použití brány pro volání podprogramu na vyšší úrovni oprávnění uveďme: E1→G2→E2. Správné, i když asi ne typické, jsou postupy: E1→G1→E3 a E2→G1→E3.

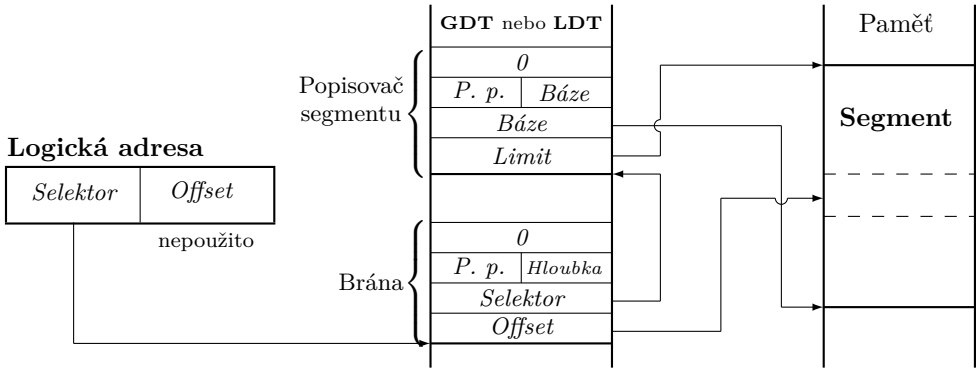
Při provádění instrukce RETF (vzdálený návrat do podprogramu) se kontroluje, je-li návrat veden do segmentu se stejnou nebo nižší úrovní oprávnění.



Obr. 5.23 Formát popisovače brány pro předání řízení v GDT nebo LDT

Při volání brány (rozuměj popisovače brány v tabulce popisovačů LDT nebo GDT) se z logické adresy uplatní pouze část selektor. Offset je ignorován. Z popisovače brány (viz obr. 5.23, na kterém bity 0 a 1 slabiky přístupová práva sdělují,

že jde o bránu určenou pro předávání řízení) se vybere selektor a offset. Tato nová hodnota selektoru adresuje popisovač, ze kterého se převezme báze volaného segmentu. K této bázi se přičte hodnota offsetu z popisovače brány, čímž procesor obdrží adresu vstupního bodu volaného podprogramu (viz obr. 5.24).



Obr. 5.24 Použití brány k předávání řízení instrukčnímu segmentu

5.5.7 Shrnutí pravidel pro předávání řízení

- Předávat řízení na jinou úroveň oprávnění lze pouze pomocí brány.
- Skok (JMP) se smí provést při C=0 pouze do segmentu se stejnou úrovní oprávnění, při C=1 do segmentu se stejnou nebo vyšší úrovní oprávnění.
- Při C=0 se smí volat podprogram (CALL) pouze takový, který je umístěn v segmentu se stejnou úrovní oprávnění. Při volání podprogramu v segmentu vyšší úrovně oprávnění se musí použít brána.
- Pro předávání řízení obsluze přerušení platí stejná pravidla jako pro volání podprogramu.
- Segmenty označené C=1 lze volat ze stejné nebo nižší úrovně oprávnění.
- CPL musí být vždy menší nebo rovno DPL zpřístupňované brány (lze přistupovat pouze k bráně na stejné nebo nižší úrovni oprávnění).
- Instrukční segment, na který ukazuje brána, musí mít DPL≤CPL procesu, který přístup provádí.
- Instrukce návratu z podprogramu (RETF) musí provést návrat pouze do segmentu se stejnou nebo nižší úrovní oprávnění.

- Přepnutí procesu lze zajistit instrukcemi CALL, JMP a INT, které se odkazují na bránu zpřístupňující TSS nebo přímo na TSS, který má DPL≥CPL procesu, který přepnutí vyvolal.

Každé předání řízení uvnitř segmentu, které změní CPL, způsobí i změnu zásobníku tak, že podle nového CPL se naplní SS:ESP z TSS aktivního procesu hodnotami náležejícími momentální úrovni oprávnění. Původní hodnota SS:ESP se uloží do zásobníku nové úrovně oprávnění. Při návratu (RETF) se tato hodnota vybere a použije pro nastavení SS:ESP.

5.5.8 Přepínání zásobníků

Pro zajištění vzájemné izolace procesů nabízí chráněný režim všem úrovním oprávnění každého procesu vlastní zásobníky. SS:ESP obsahuje vždy ukazatel do zásobníku úrovně oprávnění, na které proces právě pracuje. Systém limitů v datových segmentech hlídá případné přeplnění obsahu zásobníku. V okamžiku přeplnění by totiž došlo k zničení dat uložených bezprostředně „pod“ zásobníkem.

Zásobník je používán i pro předávání parametrů mezi volajícím modulem a volaným podprogramem tak, že volající před provedením instrukce CALL do zásobníku předávané parametry vloží např. touto posloupností instrukcí:

```
PUSH    Par1
PUSH    Par2
CALL    Podprogram
```

Brána pro předávání řízení zkopíruje parametrem **hloubka** (WC – Word Count, viz obr. 5.23) zadaný počet 32bitových dvojslov ze zásobníku úrovně oprávnění volajícího modulu do zásobníku úrovně oprávnění volaného modulu. Maximální hodnota parametru *hloubka* je 31. Potřebujeme-li předat více než 31 parametrů, může být jeden z nich ukazatel na datovou strukturu obsahující další parametry. Detailně popíšeme činnost brány při předávání řízení těmito kroky:

1. Zkontroluje se, zda zásobník volané úrovně oprávnění je dost velký na uložení parametrů a ukládaných registrů. Pokud ne, generuje se výjimka.
2. Ukazatel vrcholu zásobníku (SS:ESP) volajícího modulu (starý zásobník) se uloží do zásobníku volaného podprogramu (nový zásobník). SS:ESP se naplní obsahem ukazujícím do zásobníku úrovně oprávnění odpovídající volanému podprogramu.
3. Ze starého zásobníku se zkopíruje *hloubka* slov do nového zásobníku.
4. Do nového zásobníku se vloží jako návratová adresa (CS:EIP) adresa této brány. Tím může zásobník být použit volaným podprogramem.

Počáteční hodnota SS:ESP je nastavena při spuštění procesu. Další podrobnosti o zásobnících na jiných úrovních oprávnění budou uvedeny v části věnované přepínání procesů.

Používáním bran pro předávání řízení zeslabíme izolaci datových segmentů od procesů běžících na nižší úrovni oprávnění. Diskutovaná situace nastane, když aplikační proces na úrovni 3 volá pomocí brány službu operačního systému (např. provedení V/V operace) na úrovni 0. Potom z úrovně 0 může proces, normálně běžící na úrovni 3 (např. uvedením nesprávné adresy), zničit obsahy datových segmentů úrovní 0, 1 a 2. Tomu lze zabránit dvěma způsoby: pomocí bitu C (Conforming) sdělíme, že volaným podprogramům se má přiřadit úroveň oprávnění volajícího modulu, a pomocí indikátorů EPL a RPL.

Je-li v popisovači instrukčního segmentu nastaven bit **C (Conforming)** na jedničku, přiřadí se všem podprogramům spuštěným „zvnějšku“ tohoto modulu (prostřednictvím brány) úroveň oprávnění odpovídající úrovni oprávnění volajícího modulu. Podle výše uvedeného příkladu potom nemůže dojít ke zničení obsahu datových segmentů na úrovních 0, 1 a 2, protože podprogram (jemuž byla dočasně přiřazena úroveň 3) do těchto segmentů podle pravidel přístupu k datovým segmentům nemůže nic zapisovat, ani z nich číst. Takový podprogram má přístup k datům pouze na úrovni číslo 3.

Druhá cesta, jak se vyhnout nežádoucímu přístupu k datovým segmentům na vyšší úrovni oprávnění, je stanovení **EPL** a nastavení RPL instrukcí ARPL.

Bity 0 a 1 každého selektoru specifikují RPL (jde-li o selektor z registru CS, hovoříme o CPL). Přístup k datovému segmentu procesor povolí tehdy, platí-li: CPL≤DPL a zároveň RPL≤DPL. Při použití EPL můžeme zapsat pouze EPL≤DPL.

Hodnoty EPL vypočítává instrukce ARPL (Adjust RPL), která testuje bity 0 a 1 svých dvou operandů a přiřadí prvnímu vyšší hodnotu. Je-li RPL prvního operandu numericky menší než RPL druhého operandu, přiřadí se do RPL prvního operandu hodnota RPL druhého operandu. Instrukci ARPL používají systémové programy na vyšších úrovních oprávnění. Jako druhý operand se vkládá selektor CS s CPL volajícího modulu.

5.5.9 Privilegované instrukce

Procesor v chráněném režimu rozlišuje tva typy privilegovaných instrukcí. Do první skupiny patří instrukce, které lze provést pouze s úrovní oprávnění 0 (na nejvyšší úrovni oprávnění). Druhá skupina zahrnuje instrukce, které lze provádět pouze od jisté úrovně oprávnění.

Pozn.: Originální terminologie firmy Intel označuje instrukce první skupiny jako „privileged“ a instrukce druhé skupiny jako „trusted“. V tomto textu zůstaneme u opisujících názvů.

Instrukce, které lze **provést pouze na nejvyšší úrovni oprávnění** (CPL=0), patří do první skupiny privilegovaných instrukcí. Pokus o provedení těchto instrukcí na jiné úrovni oprávnění způsobí výjimku 13. Do této skupiny patří instrukce modifikující obsah registru GDTR, LDTR, TR, IDTR, CR*i* nebo DR*i*; instrukce spravující vyrovnávací paměti, instrukce zastavení procesoru a instrukce návratu z SMM (viz kapitola 9). Jsou to tyto instrukce:

LGDT	naplnění registru GDTR,
LIDT	naplnění registru IDTR (bude popsán dále),
LLDT	naplnění registru LDTR,
LTR	naplnění registru TR (bude popsán dále),
MOV	naplnění registru CR*i* nebo DR*i*,
CLTS	nulování bitu TS (Task Switch) v registru CR0,
INVD	vyprázdnění IVP bez uložení,
WBINVD	vyprázdnění IVP s uložením,
INVLPG	vyprázdnění TLB,
RSM	návrat z SMM,
HLT	zastavení procesoru.

Instrukce POPF a IRET nejsou sice privilegovanými instrukcemi, ale na jiné úrovni oprávnění než nejvyšší (CPL=0) nesmějí měnit obsah bitů IOPL v příznakovém registru.

Do druhé skupiny zahrneme ty instrukce, které se nesmějí provádět na úrovni oprávnění nižší, než je úroveň oprávnění zadaná bity **IOPL** (Input/Output Privilege Level) v příznakovém registru. Pravidlo pro povolení provedení takové instrukce je **CPL≤IOPL**. Do této skupiny patří instrukce:

IN, INS	čtení ze V/V brány,
OUT, OUTS	zápis na V/V bránu,
STI, CLI	změna příznaku IF.

Hodnota IOPL má rovněž vliv na nastavování příznaku IF jinými prostředky než instrukcemi STI a CLI. Je-li CPL>IOPL, nelze hodnotu IF v příznakovém registru změnit ani jinou instrukcí plnící registr EFLAGS (např. POPF). Po provedení takové instrukce zůstává vždy příznak IF nezměněn a není generováno žádné přerušení.

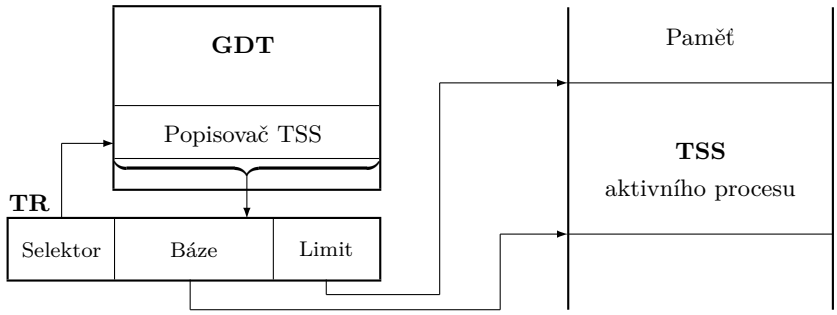
5.6 Přepínání procesů

Jedním z hlavních rysů chráněného režimu procesoru je možnost mít v paměti naráz uloženo více procesů a přepínat mezi nimi tak, že v daném okamžiku se provádí právě jeden. Přepínání procesů zajišťuje procesor ve vlastní režii, aniž by pro přepnutí bylo nutné provedení speciálních instrukcí.

5.6.1 Segment stavu procesu a registr TR

O každém procesu (aktivním i neaktivním) jsou v paměti ve speciálním segmentu uloženy všechny potřebné informace. Tento segment se nazývá **segment stavu procesu** (TSS – Task State Segment).

Stav procesu je dán obsahem všech registrů procesoru. Je-li proces momentálně neaktivní, je jeho stav zapsán v TSS tak, aby po aktivaci byly procesu zajištěny stejné podmínky, jaké měl před přerušením činnosti. Každý popisovač TSS je vlastně popisovačem systémového segmentu a smí být uložen pouze v GDT. Adresa popisovače TSS právě aktivního procesu je uložena ve speciálním registru **TR (Task Register)** tak, jak je to uvedeno na obr. 5.25. Na každý TSS ukazuje právě jeden popisovač TSS (TSS není reentrantní).



Obr. 5.25 Zpřístupnění TSS aktivního procesu pomocí registru TR

Registr **TR** je složen ze dvou částí: z viditelné 16bitové části, obsahující selektor ukazující do GDT, a neviditelné části, obsahující bázi segmentu a limit ukazující na skutečný TSS.

Tvar TSS je na obr. 5.26. Velikost segmentu (limit) není obecně stanovena. Maximální délka TSS je 4 GB a minimální je 104 slabik. V TSS jsou uloženy tyto položky:

31	0	
Offset mapy V/V bran	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **T**	100
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor LDT	96
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor GS	92
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor FS	88
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor DS	84
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor SS	80
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor CS	76
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selektor ES	72
EDI		68
ESI		64
EBP		60
ESP		56
EBX		52
EDX		48
ECX		44
EAX		40
EFLAGS		36
EIP		32
CR3 (DBA)		28
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS pro úroveň 2	24
ESP pro úroveň 2		20
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS pro úroveň 1	16
ESP pro úroveň 1		12
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS pro úroveň 0	8
ESP pro úroveň 0		4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Zpětný ukazatel	0

Obr. 5.26 Tvar TSS

Zpětný ukazatel obsahuje selektor TSS přerušeného procesu. Ukazatel se použije při obnovení přerušeného procesu instrukcí IRET.

Ukazatele zásobníků zahrnují momentální stavy ukazatelů zásobníků pro úrovně oprávnění 0, 1 a 2.

Řídicí registry vztahující se k procesu - registr CR3 s bází stránkového adresáře, registr EIP čítače instrukcí a příznakový registr EFLAGS. Nejsou zde registry vztahující se k procesoru (CR0 apod.).

Všeobecné registry procesoru.

Segmentové registry procesoru.

Selektor LDT obsahuje selektor položky GDT, který specifikuje LDT náležející procesu.

T (Trap) je bit, který, je-li nastaven na jedničku, způsobí v okamžiku přepnutí na tento proces ladicí výjimku 1.

Offset mapy přístupných V/V bran ukazuje na začátek bitové mapy uvnitř tohoto segmentu stavu procesu. Báze segmentu ukazuje na položku *Zpětný ukazatel* a offset musí být minimálně 104 a maximálně DFFFh.

Protože je TSS v systémovém segmentu, nelze do něj přímo zapisovat, ani jej přímo číst. Jedinou formou zpřístupnění je předání řízení na popisovač TSS (instrukcí JMP nebo CALL). Tím je provedeno přepnutí na proces, jehož TSS byl instrukcí zpřístupněn. Pravidlo pro zpřístupnění TSS je **EPL≤DPL**, tj. lze přepnout na proces, který je na stejné nebo nižší úrovni oprávnění.

Segment stavu procesu může být umístěn kdekoli v lineárním adresovém prostoru. Není vhodné jej při zapnutém stránkování umístit přes hranici stránky. V případě, že by při přepínání procesů (v okamžiku čtení TSS) nebyla druhá stránka v paměti (P=0), nastalo by přerušení pro výpadek stránky, což komplikuje operaci přepínání procesů. Proto se doporučuje TSS přes hranici stránky neumísťovat.

Poznámky k předchozím procesorům

Šestnáctibitový procesor Intel286 měl jinou strukturu TSS, která odpovídala jeho architektuře. Vyšší typy procesorů pro zachování kompatibility programového vybavení umějí pracovat i s TSS typu Intel286 (viz též obsah popisovače systémového segmentu na str. 56). Formát TSS Intel286 je na obr. 5.27.

5.6.2 Popisovač TSS

Popisovač TSS je jedním z typů popisovače systémového segmentu (viz str. 56). Typ se určuje nejnižšími 4 bity slabiky přístupových práv. Popisovač TSS blíže určují tyto hodnoty:

15 **TSS** 0	
...	
Selektor LDT	42
Selektor DS	40
Selektor SS	38
Selektor CS	36
Selektor ES	34
DI	32
SI	30
BP	28
SP	26
BX	24
DX	22
CX	20
AX	18
F	16
IP	14
SS pro úroveň 2	12
SP pro úroveň 2	10
SS pro úroveň 1	8
SP pro úroveň 1	6
SS pro úroveň 0	4
SP pro úroveň 0	2
Zpětný ukazatel	0

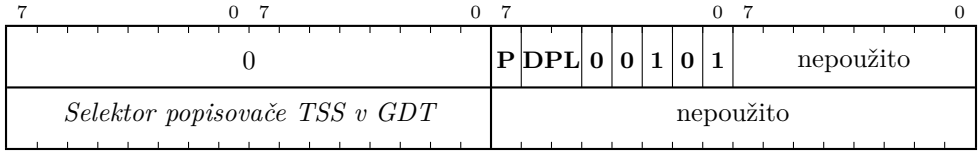
Obr. 5.27 Tvar TSS Intel286

1 … TSS neaktivního procesu Intel286,
3 … TSS aktivního procesu Intel286,
9 … TSS neaktivního procesu od Intel386,
B … TSS aktivního procesu od Intel386,

Aktivní proces je ten, který právě procesor provádí, nebo čeká na přidělení procesoru. Toto rozlišení je důležité proto, že procesy nejsou reentrantní. Přepnout lze pouze na proces neaktivní. Při přepnutí instrukcí JMP a IRET přestává být původní proces aktivní, při přepnutí instrukcí CALL nebo přerušením původní proces zůstává aktivní. Vracet z přerušení či z volání CALL se smí pouze do aktivního procesu.

5.6.3 Brána zpřístupňující segment stavu procesu

Brána zpřístupňující TSS (Task Gate) může být umístěna v GDT, LDT nebo IDT (bude popsáno později). Protože popisovač TSS smí být umístěn pouze v GDT, zprostředkovává brána přepínání procesů i pomocí LDT. Formát popisovače brány zpřístupňující TSS je na obr. 5.28.



Obr. 5.28 Formát popisovače brány zpřístupňující TSS v GDT, LDT nebo IDT

Ačkoli na každý TSS smí ukazovat právě jeden popisovač TSS, může více bran zpřístupňovat jeden popisovač TSS. Tato situace je správná proto, že příznak aktivního TSS je uložen v jeho popisovači TSS.

Při použití brány se kontroluje, zda RPL a CPL jsou numericky menší nebo rovny DPL popisovače brány. Procesy na nižší úrovni oprávnění mají tak zabráněno přepínat na tento proces.

5.6.4 Přepínání procesů

Operace přepnutí procesu sestává z těchto elementárních akcí:

1. Současný stav procesu (registry procesoru) se uloží do TSS, jehož adresa je v TR.
2. Procesor naplní TR selektorem popisovače TSS nového procesu.
3. Procesor naplní všechny své registry obsahem TSS, na který ukazuje TR.
4. Procesor předá řízení novému procesu.

Přepnutí procesu může být vyvoláno:

vzdáleným JMP nebo CALL, jehož selektor ukazuje na popisovač TSS nového procesu v GDT. Offsetová část adresy se ignoruje.

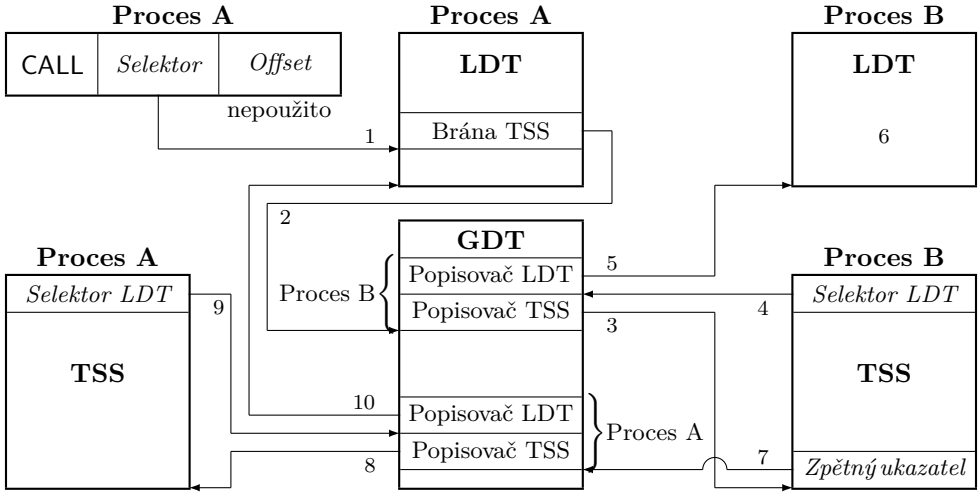
vzdáleným JMP nebo CALL, jehož selektor ukazuje na bránu (zpřístupňující TSS nového procesu) v GDT, LDT nebo IDT. Offsetová část adresy se ignoruje.

IRET s nastaveným NT=1. Instrukce IRET vyvolá přepnutí procesu pouze tehdy, je-li nastaven příznak NT (Nested Task) v příznakovém registru. Proces, na který se má v tomto případě přepnout, je zadán zpětným ukazatelem v TSS právě aktivního procesu.

přerušením, jehož přerušovací vektor ukazuje na bránu (zpřístupňující TSS nového procesu) v IDT.

Na obr. 5.29 je rámcově znázorněno přepnutí procesu instrukcí CALL pomocí brány (není diskutováno ukládání stavu procesu). Předpokládejme, že chceme přepnout

provádění z procesu A na proces B. Přepnutí aktivizujeme v procesu A instrukcí vzdáleného volání CALL, ve které je offsetová část adresy nevýznamná. Další kroky jsou popsány v bodech:



Obr. 5.29 Přepnutí procesů vyvolané instrukcí CALL pomocí brány

1. Selektor z instrukce CALL ukazuje do LDT na bránu zpřístupňující TSS procesu B.
2. Z brány zpřístupňující TSS byl vybrán selektor popisovače TSS procesu B v GDT.
3. Popisovač TSS procesu B v GDT obsahuje bázi a limit TSS procesu B.
4. V TSS procesu B je zapsán stav procesu B (obsah všech registrů procesoru). Selektor LDT ukazuje do GDT na popisovač LDT procesu B.
5. V GDT je uložen popisovač LDT procesu B, obsahující bázi a limit LDT procesu B.
6. Předá se řízení procesu B.

V následujících krocích je proveden návrat do přerušeného procesu A. Návrat se vyvolá instrukcí IRET s nastaveným příznakem NT.

7. Zpětný ukazatel TSS právě aktivního procesu (t.č. procesu B) obsahuje selektor ukazující do GDT na popisovač TSS procesu, na který se má přepnout (proces A).

Kroky 8, 9 a 10 jsou analogické s kroky 3, 4 a 5.

5.6.5 Detailní popis přepínání procesů

Přepnutí procesu instrukcí vzdáleného skoku JMP *Selektor* (*Offset* je ignorován):

1. *Selektor* ukazuje na bránu zpřístupňující TSS nového procesu.
   (a) Kontrola oprávněnosti přístupu k bráně: CPL≤DPL brány.
   (b) *Selektor* ukazuje na bránu v GDT, LDT nebo IDT.
   (c) Brána obsahuje selektor ukazující na popisovač TSS v GDT.

   Selektor ukazuje na popisovač TSS nového procesu v GDT.
   (a) Kontrola oprávněnosti přístupu k TSS: CPL≤DPL popisovače.
2. Kontrola popisovače TSS: nový proces musí být neaktivní (Typ=1), popisovač musí mít P=1 a správný limit.
3. Uložení stavu procesoru do TSS podle TR.
4. Deaktivace starého procesu aktualizací popisovače TSS (Typ:=1).
5. Naplnění obsahu TR (selektor a neviditelná část) ukazatelem na TSS nového procesu.
6. Aktivace nového procesu aktualizací popisovače TSS (Typ:=3) a nastavení TS=1 (bit Task Switch v CR0).
7. Nulování příznaku NT nového příznakového registru (IRET neprovádí návrat).
8. Zavedení nového obsahu všech registrů procesoru podle TSS, na který ukazuje TR.

Přepnutí procesu instrukcí vzdáleného volání CALL *Selektor* (*Offset* je ignorován)**, nebo přerušením**:

1. *Selektor* ukazuje na bránu zpřístupňující TSS nového procesu.
   (a) Kontrola oprávněnosti přístupu k bráně: CPL≤DPL brány.
   (b) *Selektor* ukazuje na bránu v GDT, LDT nebo IDT.
   (c) Brána obsahuje selektor ukazující na popisovač TSS v GDT.

   Selektor ukazuje na popisovač TSS nového procesu v GDT.
   (a) Kontrola oprávněnosti přístupu k TSS: CPL≤DPL popisovače.

   Přepnutí je vyvolané přerušením.

   (a) Přerušovací vektor ukazuje na bránu v IDT.
   (b) Brána obsahuje selektor ukazující na popisovač TSS v GDT.
2. Kontrola popisovače TSS: nový proces musí být neaktivní (Typ=1), popisovač musí mít P=1 a správný limit.
3. Uložení stavu procesoru do TSS podle TR.
4. Starý proces zůstává aktivní (Typ je stále 3).
5. Dočasné uschování původního obsahu TR a následné naplnění TR (selektor a neviditelná část) ukazatelem na TSS nového procesu.
6. Aktivace nového procesu aktualizací popisovače TSS (Typ:=3) a nastavení TS=1 (bit Task Switch v CR0).
7. Jedničkování příznaku NT nového příznakového registru (nejbližší IRET způsobí přepnutí).
8. Naplnění zpětného ukazatele TSS, jehož adresa je uložena v TR, dočasně uschovanou hodnotou původního TR (ukazatel na proces, který inicioval přepnutí).
9. Zavedení nového obsahu všech registrů procesoru podle TSS, na který ukazuje TR.

Přepnutí procesu instrukcí IRET s NT=1 (je-li NT=0, jde o klasický návrat z přerušení):

1. Obsah TR ukazuje na TSS právě aktivního procesu.
2. Deaktivace tohoto procesu aktualizací popisovače TSS (Typ:=1).
3. Nulování NT (NT:=0).
4. Uložení stavu procesoru do TSS podle TR.
5. Naplnění obsahu TR (selektor) zpětným ukazatelem z TSS podle TR.
6. Kontrola nového TSS – musí být aktivní (Typ=3).
7. Zavedení nového obsahu všech registrů procesoru podle TSS, na který ukazuje TR.

5.6.6 Brány zpřístupňující TSS versus přerušení

Obsluha přerušení pomocí brány zpřístupňující TSS má několik výhod oproti klasické metodě obsluhy přerušení:

- V přerušení obslouženém bránou se automaticky uchovávají všechny registry procesoru, zatímco klasické přerušení uloží pouze CS:EIP a EFLAGS.

- Přerušený proces a proces přerušením aktivovaný jsou od sebe plně izolovány.
- Při klasické obsluze přerušení se rutině pro obsluhu přerušení předává řízení pouze jedním vstupním bodem. Při použití brány zpřístupňující TSS může být přerušovací rutina aktivována v místě, kde předchozí obsluha skončila. Touto vlastností lze např. při ovládání periferních zařízení snadno oddělit obsluhu po inicializaci přenosu dat a obsluhu vlastního přenosu dat. Nevýhodou ovšem je, že obsluha přerušení branou zpřístupňující TSS trvá déle než obsluha klasického přerušení.

5.6.7 Mapa přístupných V/V bran

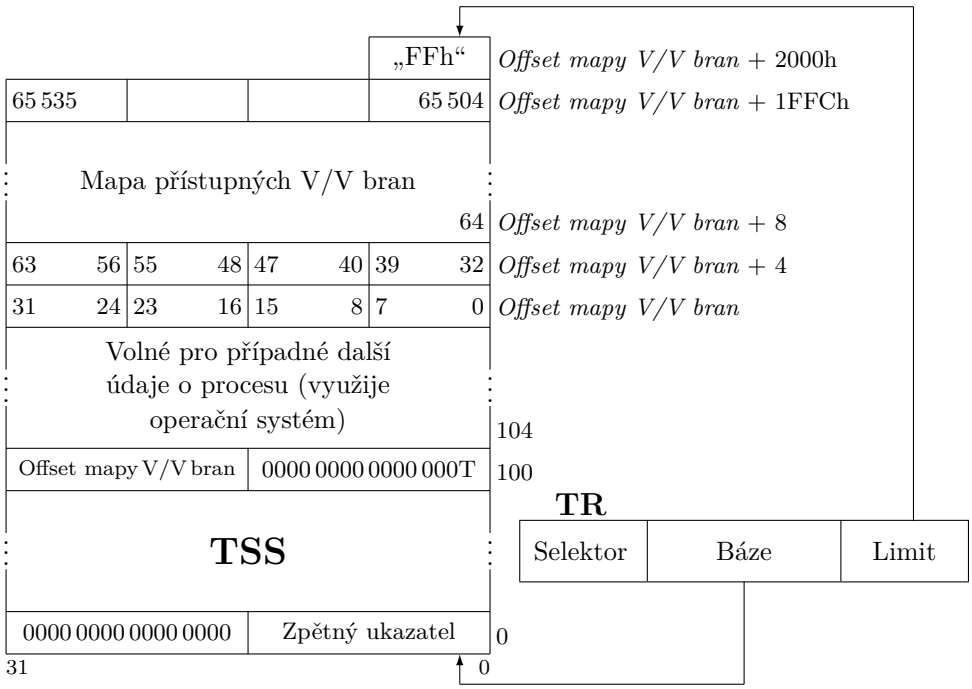
Mapa přístupných V/V bran vymezuje množinu V/V bran přístupných procesu. Tato mapa se vztahuje pouze na separátní adresový prostor V/V bran, nikoli na V/V brány mapované do adresového prostoru fyzické paměti.

Mapa je uložena v TSS. Její umístění a velikost může být proces od procesu různé. Začátek mapy vymezuje položka TSS nazývaná „offset mapy přístupných V/V bran“ (viz obr. 5.26 na str. 83). Hodnota ukazuje na začátek bitové mapy uvnitř tohoto segmentu stavu procesu. Báze segmentu ukazuje na položku *Zpětný ukazatel* a offset musí být minimálně 104 a maximálně DFFFh. Každý bit této mapy odpovídá jedné 8bitové V/V bráně tak, že např. brána 41 má zpřístupňující bit na adrese *offset mapy*+5 (bit č. 1).

Podle pravidel uvedených na str. 81 se mapa přístupných V/V bran používá pro kontrolování V/V instrukcí pouze tehdy, je-li CPL>IOPL nebo proces je V86. Podle čísla V/V brány použitého ve V/V instrukci se testuje bit mapy a má-li hodnotu 0, je V/V instrukce provedena. Je-li testovaný bit jedničkový, generuje se výjimka 13. Pracuje-li V/V instrukce se slovem nebo dvojslovem, testují se dva nebo čtyři bity na sousedících adresách. V/V instrukce se potom povolí jenom tehdy, jsou-li všechny testované bity nulové.

Mapa nemusí obsahovat obraz všech 64 K V/V bran. Velikost mapy je určena hodnotou *Limit*. Všechny v mapě neuložené V/V adresy jsou automaticky pro V/V operace nepřístupné. Mezi poslední slabikou mapy a adresou, na kterou ukazuje *Limit*, **musí** být vždy alespoň jedna slabika obsahující samé jedničky (FFh). Umístění mapy přístupných V/V bran je patrné též z obr. 5.30.

V reálném režimu nejsou žádné TSS, a proto V/V instrukce mohou pracovat se všemi V/V bránami. V chráněném režimu se mapa použije pouze tehdy, je-li CPL>IOPL. Je-li CPL≤IOPL, mohou V/V instrukce adresovat všechny V/V brány bez ohledu na obsah mapy. V režimu virtuální 8086 je TSS a není IOPL, a proto se mapa použije při každé V/V instrukci bez ohledu na CPL.



Obr. 5.30 Umístění mapy přístupných V/V bran v TSS

Poznámky k předchozím procesorům

Mapu přístupných V/V bran zavádí procesor Intel386.

5.7 Výjimky a přerušení

Výjimky a přerušení jsou v chráněném režimu obsluhovány diametrálně odlišně než v reálném režimu. Zásadní změnou je jiný formát tabulky přerušovacích vektorů a zavedení nové struktury, podobající se GDT.

5.7.1 Tabulka popisovačů segmentů obsluhy přerušení

Tabulka popisovačů segmentů obsluhy přerušení **IDT** (Interrupt Descriptor Table) obsahuje až 256 popisovačů. Adresa IDT je uložena ve speciálním registru **IDTR** (Interrupt Descriptor Table Register). Registr obsahuje 32bitovou bázi a 16bitový

limit IDT. Plní se privilegovanou instrukcí LIDT (lze ji provést pouze na úrovni oprávnění 0). V IDT se vyskytují pouze tyto tři typy popisovačů:

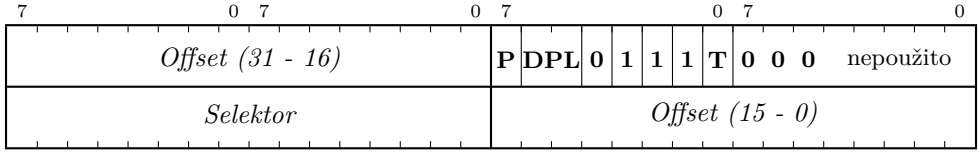
- brána zpřístupňující TSS (popsána na str. 85),
- brána pro maskující přerušení (Interrupt Gate),
- brána pro nemaskující přerušení (Trap Gate).

Odkaz na bránu zpřístupňující TSS způsobí přepnutí procesů, zatímco obsluha dalších dvou typů je v rámci stejného procesu bez přepnutí.

V okamžiku přerušení je vybrán jeden z 256 popisovačů IDT podle obsahu přerušovacího vektoru (v intervalu 0 až 255). V IDT nemusí být využito všech 256 položek. Nepoužité položky mají ve slabice *přístupová práva* nulu (P=0) nebo limit IDT je menší než 256 položek.

5.7.2 Brány pro přerušení

Brána pro maskující přerušení (Interrupt Gate) se od brány pro nemaskující přerušení (Trap Gate) liší pouze zacházením s příznakem IF (Interrupt Flag – příznak povolující maskovatelná přerušení). Obsluhuje-li procesor přerušení bránou pro maskující přerušení, zakáže další přerušení vynulováním příznaku IF. Provádí-li totéž bránou pro nemaskující přerušení, příznak IF nenuluje.

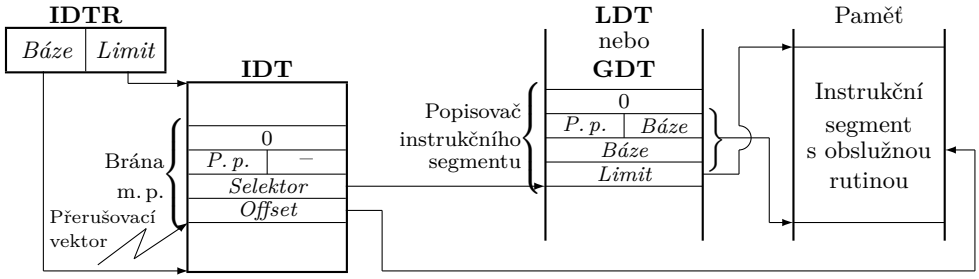


Obr. 5.31 Formát popisovače brány přerušení v IDT

Formát brány pro maskující i nemaskující přerušení je na obr. 5.31. Je-li bit **T=1**, jde o bránu pro nemaskující přerušení (Trap Gate), je-li **T=0**, jde o bránu pro maskující přerušení (Interrupt Gate). Položka *Selektor* ukazuje na popisovač instrukčního segmentu v GDT nebo LDT. V tomto instrukčním segmentu je od relativní adresy *Offset* uložena rutina pro obsluhu právě vyvolaného přerušení (viz obr. 5.32).

Při povolování přístupu k rutině obsluhující přerušení platí stejná pravidla jako pro zpřístupňování instrukčního segmentu branou předávající řízení: **CPL≤DPL brány** přerušení a zároveň **CPL≥DPL rutiny** obsluhující přerušení. Přerušením tedy nelze předat řízení na nižší úrovni oprávnění.

Někdy je vhodné pro uložení kódu obsluhujícího výjimku použít přizpůsobitelný segment (C=1), např. při obsluze dělení nulou. Rutina potom musí používat data



Obr. 5.32 Obsluha výjimky a přerušení branou v IDT

pouze ze zásobníku nebo z datového segmentu, majícího odpovídající úroveň oprávnění.

Návrat z obsluhy přerušení provádí instrukce IRET. Na rozdíl od instrukce RET ze zásobníku vyzvedne ještě obraz registru EFLAGS. Pole IOPL v registru EFLAGS obrazem přepíše pouze tehdy, je-li CPL nulové. Příznak IF přepíše pouze při CPL≤IOPL.

5.7.3 Plnění zásobníku při přerušení

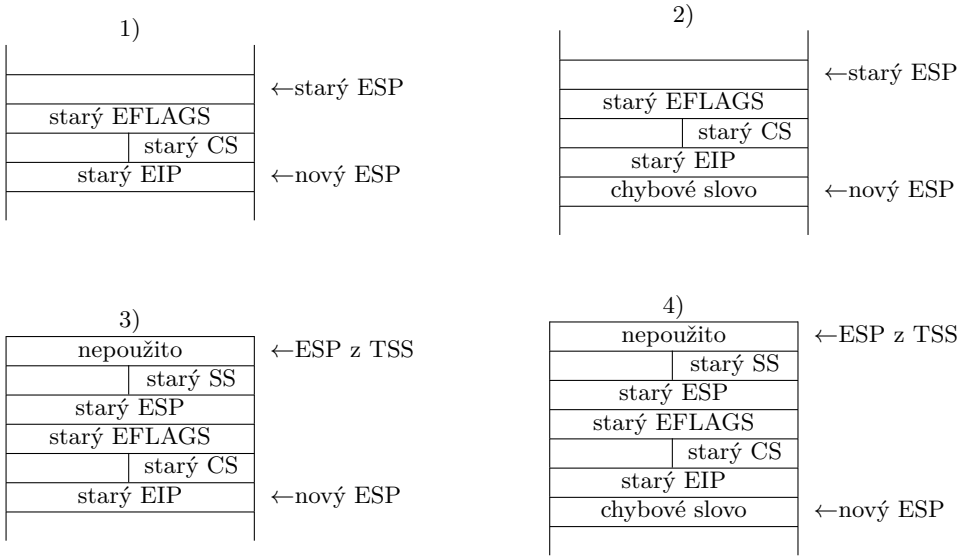
Z hlediska informací ukládaných do zásobníku rozlišujeme čtyři typy přerušení – viz obr. 5.33. Rutina obsluhující přerušení pracuje na stejné úrovni oprávnění jako přerušovaný proces (případ 1 a 2), nebo na jiné (případ 3 a 4). Přerušení či výjimka předávají chybové slovo (případ 2 a 4), nebo nepředávají (případ 1 a 3).

5.7.4 Chybové slovo

Chybové slovo se generuje zpravidla tehdy, týká-li se přerušení konkrétního segmentu. Obsah chybového slova se podobá selektoru s tím, že nejnižší dva bity mají jiný význam (viz obr. 5.34). Chybové slovo se pro dodržení kompatibility typů do zásobníku ukládá na 4 slabikách. Horní dvě slabiky jsou proto nevyužity. V některých případech je chybové slovo prázdné, potom jsou dolní dvě slabiky nulové.

IDT indikuje, že index ukazuje do IDT (nikoli do GDT nebo LDT podle TI).

EXT (External) je nastaven tehdy, bylo-li přerušení způsobeno vnější událostí bez zavinění procesu (např. přerušení 10: vnější přerušení přes bránu zpřístupňující TSS vyvolalo pokus o přepnutí na proces, který má TSS s chybným obsahem).



1) bez změny úrovně oprávnění, bez chybového slova
2) bez změny úrovně oprávnění, s chybovým slovem
3) se změnou úrovně oprávnění, bez chybového slova
4) se změnou úrovně oprávnění, s chybovým slovem

Obr. 5.33 Možné obsahy zásobníku při aktivaci obsluhy přerušení

16	15	2	1	0
Nepoužito	*Index*	**TI**	**IDT**	**EXT**

Obr. 5.34 Tvar chybového slova

5.7.5 Rezervované výjimky

Výjimky generované procesorem dělíme do tří kategorií:

Fault do zásobníku uloží CS:EIP ukazující **na instrukci**, která způsobila přerušení.

Trap do zásobníku uloží CS:EIP ukazující **za instrukci** (na následující instrukci), která přerušení způsobila. Pokud instrukce, která způsobila výjimku, je instrukcí předávání řízení, potom CS:EIP uložené v zásobníku ukazuje na cílové místo (nikoli na následující instrukci za instrukcí způsobující výjimku).

Abort způsobí, že v procesu nelze pokračovat a musí být násilně ukončen. Výjimka tohoto typu nastává při chybách systémových tabulek, či při chybách technického vybavení.

Číslo vektoru	*Určení vektoru*	*Typ výjimky*	*Chybové slovo?*
0	Chyba dělení	Fault	ne
1	Ladicí výjimka	Trap/Fault	ne
3	Ladicí bod	Trap	ne
4	Přeplnění	Trap	ne
5	Kontrola mezí	Fault	ne
6	Chybný operační kód	Fault	ne
7	Nedostupnost koprocesoru	Fault	ne
8	Dvojnásobný výpadek segmentu	Abort	ano (=0)
10	Chybný TSS	Fault	ano
11	Výpadek segmentu	Fault	ano
12	Výpadek segmentu se zásobníkem	Fault	ano
13	Obecná chyba ochrany	Fault	ano
14	Výpadek stránky	Fault	ano
16	Chyba koprocesoru	Fault	ne
17	Kontrola zarovnání	Fault	ano (=0)
18	Chyba v procesoru	-	
0 až 255	Programové přerušení INT	Trap	ne

Obr. 5.35 Výjimky generované procesorem

INT 0 Chyba dělení (Divide Error)

Výjimka 0 nastane při dělení nulou v instrukcích DIV a IDIV.

Poznámky k předchozím procesorům

Obsah CS:IP uložený do zásobníku procesoru 8086 ukazoval **za** instrukci, která přerušení způsobila. Při přenosu programového vybavení z 8086 na vyšší typy může proto při obsluze výjimky 0 dojít k zacyklení.

INT 1 Ladicí výjimky (Debug Exceptions)

Ladicí výjimku generuje ladicí systém procesoru. Existuje pět příčin, pro které procesor tuto výjimku generuje:

1. při čtení/zápisu z/do paměti byl detekován ladicí bod (Trap),
2. při výběru instrukce byl detekován ladicí bod (Fault),
3. po provedení instrukce v krokovacím režimu (Trap),
4. při přepnutí na proces mající v TSS T=1 (Trap),
5. nedovoleným přístupem k ladicím registrům při GD=1 (Fault).

Rutina, která výjimku obsluhuje, zjistí příčinu čtením ladicího registru DR6. Podrobnosti jsou uvedeny spolu s popisem ladicího systému procesoru v kapitole 7 od str. 113.

Poznámky k předchozím procesorům

Ladicí systém je zaveden od procesoru Intel386. Ve předchozích typech byl implementován pouze krokovací režim následovně:

Je-li ladicím programem nastaven příznak TF=1 (Trap Flag), vygeneruje se výjimka 1 po provedení první instrukce. Výjimkou se registr příznaků s nastaveným TF uloží do zásobníku a TF dostupné z obslužné rutiny se nuluje, aby obslužná rutina mohla vůbec proběhnout. Obslužná rutina aktivuje ladicí program a ten zjistí, co se během provedení instrukce změnilo, a oznámí to uživateli. Pokud uživatel zadá příkaz na provedení další instrukce z takto krokovaného programu, obslužná rutina prostě skončí instrukcí IRET. Ta obnoví ze zásobníku registr příznaků (včetně TF=1) a adresy, na které byl laděný program přerušen.

Takto lze snadno procházet krokovaným programem. Jedinou možností, jak provést první instrukci laděného programu, je uložit do zásobníku instrukcí PUSH hodnoty tak, jak by byly uloženy přerušením. To znamená uložit registr F (s TF=1) a CS:IP ukazující na první instrukci. Poté hodnoty aktivovat instrukcí IRET.

Výjimka 1 se negeneruje po instrukcích MOV a POP plnících některý ze segmentových registrů.

INT 3 Ladicí bod (Breakpoint)

Výjimka 3 se používá společně s výjimkou 1 v ladicích programech. Výjimka číslo 3 se vygeneruje po dekódování speciální jednoslabikové instrukce INT 3 (s operačním kódem 0CCh). Tuto instrukci ladicí program vloží na to místo, na kterém se má běžící program zastavit. Výjimka uloží do zásobníku obsah CS:EIP ukazující na slabiku bezprostředně za touto instrukcí.

Ladicí program uloží instrukci 0CCh na místo operačního znaku (resp. instrukčního prefixu) té instrukce, před kterou se má provádění programu zastavit. Ladicí program si musí zapamatovat adresu a původní obsah přepsaného paměťového místa ze dvou důvodů: za prvé, aby po aktivaci obsluhy výjimky 3 rozpoznal, která z instrukcí 0CCh přerušení způsobila (ladicích bodů může být v laděném programu více), a za druhé, aby systém dokázal instrukci restaurovat tak, aby laděný program mohl pokračovat.

INT 4 Přeplnění (Overflow)

Je-li nastaven příznak OF=1 (Overflow Flag) a potom je provedena instrukce INTO (Interrupt on Overflow), generuje se výjimka 4. Příznak OF je nastaven během některých instrukcí ze skupiny aritmetických operací při zjištění přeplnění (výsledek je mimo rozsah zobrazení).

INT 5 Kontrola mezí (Bound Check)

Výjimku vyvolá instrukce BOUND, je-li její operand mimo zadané meze (viz popis instrukce BOUND na str. 225). Instrukce IRET umístěná v přerušovací rutině způsobí opakování BOUND, protože CS:EIP uložené do zásobníku ukazuje na instrukci, která přerušení způsobila.

Poznámky k předchozím procesorům

Tato a další výjimky jsou zavedeny od procesoru Intel286 (není-li uvedeno jinak).

INT 6 Chybný operační kód (Invalid Opcode)

Tuto výjimku generuje prováděcí jednotka procesoru, nedokázala-li rozpoznat operační znak instrukce, nalezla-li chybnou kombinaci operačního znaku a operandu (např. použití registru v instrukci vzdáleného skoku), nebo při použití prefixu LOCK před instrukcí, se kterou se prefix použít nesmí. Chybný operační kod se pozná až při provádění instrukce, nikoli už při jejím výběru. CS:EIP uložené do zásobníku ukazuje na instrukci, která přerušení způsobila.

Firmou Intel jsou vyhrazeny operační znaky D6 a F1. Tyto jsou nedefinovány, ale výjimku 6 negenerují.

INT 7 Nedostupnost koprocesoru (Processor Extension Not Available)

Výjimku mohou vyvolat instrukce obsluhující koprocesor v závislosti na indikátorech zaznamenaných v registru CR0 (viz obr. 5.12 na str. 61):

- Je-li nastaven bit TS=1 a zároveň MP=1, znamená to, že byl přepnut proces v procesoru, ale nikoli v koprocesoru. Pokus o provedení instrukce WAIT nebo instrukce určené koprocesoru (ESC) způsobí výjimku 7. Bit TS je nutno vynulovat (po uložení stavu koprocesoru) instrukcí CLTS.
- Je-li EM=1, generuje se toto přerušení při každém výskytu instrukce určené koprocesoru (ESC). Tato funkce dovoluje programově emulovat fyzicky neinstalovaný koprocesor.

INT 8 Dvojnásobný výpadek segmentu (Double Fault)

Výjimka se generuje tehdy, vyskytne-li se výjimka v okamžiku předávání řízení obslužné rutině předchozího přerušení (např. je-li instrukční segment obsluhující přerušení označen P=0, tak se při pokusu o zpřístupnění generuje výjimka 11, a je-li segment s rutinou obsluhující výjimku 11 označen rovněž P=0, generuje se výjimka 8.

Vyskytne-li se výjimka i během provádění obsluhy výjimky 8, procesor se zastaví (z tohoto stavu ho vyvede buď RESET, nebo NMI). Výjimka ukládá do zásobníku nulové chybové slovo.

INT 9 Rezervováno

Tato výjimka se v procesorech Pentium a Intel486 nepoužívá.

Poznámky k předchozím procesorům

V procesoru Intel386 bylo definováno jako „Překročení segmentu koprocesorem“ (Processor Extension Segment Overrun). Bylo vyvoláno, překročil-li koprocesor limit segmentu při čtení nebo zápisu operandu. Obsluha tohoto přerušení musí respektovat, že v době mezi zahájením instrukce koprocesoru a jejím dokončením mohlo dojít k přepnutí procesu v procesoru. Nedošlo-li k přepnutí procesu, je pravděpodobné, že v okamžiku vzniku tohoto přerušení bude procesor již zpracovávat několikátou instrukci za instrukcí, která přerušení vyvolala. V procesoru Intel486 a Pentium na tuto situaci reaguje výjimkou 13.

INT 10 Chybný TSS (Invalid Task State Segment)

Výjimka nastává při pokusu o přepnutí na proces mající chybný TSS. Výjimka se generuje při splnění jedné z následujících podmínek (v pravém sloupci je obsah chybového slova):

Podmínka	*Index chyb. slova*
• limit TSS < 67h	TSS
• LDT je chybná nebo není přítomná	LDT
• selektor zásobníkového segmentu překročil limit tabulky	SS
• zásobníkový segment není zapisovatelný	SS
• DPL zásobníkového segmentu je různé od CPL	SS
• RPL SS je různé od CPL	SS
• selektor instrukčního segmentu překročil limit tabulky	CS
• instrukční segment není proveditelný	CS
• DPL nepřizpůsobitelného instrukčního segmentu je různé od CPL	CS
• DPL přizpůsobitelného instrukčního segmentu je větší než CPL	CS
• selektor datového segmentu překročil limit tabulky	DS
• datový segment není čitelný	DS

Tato výjimka do zásobníku ukládá chybové slovo. Není-li splněna první podmínka (velikost TSS), provede se přepnutí procesu. Z toho plyne, že při splnění první podmínky se výjimka 10 obsluhuje v rámci starého procesu. Není-li první podmínka splněna a je-li splněna některá z dalších, obsluhuje se výjimka v rámci nového procesu.

INT 11 Výpadek segmentu (Segment Not Present)

Výjimka nastane tehdy, je-li detekováno P=0 v těchto okamžicích:

- při plnění registrů CS, DS, ES, FS nebo GS,
- při plnění registru LDTR instrukcí LLDT,
- při přístupu k bráně.

V chybovém slově je selektor segmentu, který vyvolal přerušení.

INT 12 Výpadek segmentu se zásobníkem (Stack Fault)

Výjimka je generována v těchto případech:

- pokud některá z instrukcí, odkazujících se na registr SS (POP, PUSH, ENTER, LEAVE, CALL atd.), způsobí překročení limitu segmentu se zásobníkem,

- pokud při plnění registru SS je selektor popisovače označen P=0. Tento stav může nastat při:

  - přepnutí procesu,
  - vzdáleném volání podprogramu instrukcí CALL,
  - vzdáleném návratu z podprogramu instrukcí RETF,
  - instrukcemi MOV a POP modifikujícími SS.

INT 13 Obecná chyba ochrany (General Protection Fault)

Tato výjimka se vyvolá při chybě detekované systémem ochran v těch případech, ve kterých nepatří pod žádnou z dosud diskutovaných výjimek. Může nastat při těchto pokusech:

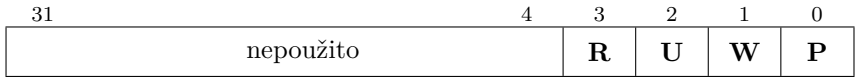
- překročení limitu segmentu,
- předání řízení (skok) do datového segmentu,
- zápis do segmentu určeného pouze ke čtení nebo ke spuštění,
- čtení segmentu určeného pouze ke spuštění,
- použití neplatného selektoru,
- přístup k popisovači, který je za limitem příslušné tabulky,
- přepnutí na aktivní úlohu,
- provedení privilegované instrukce při CPL>0,
- překročení limitu délky instrukce, který je 15 slabik. Tato situace může nastat pouze při použití redundantních prefixů,
- nastavení PG=1 při PE=0,
- zápis jedničky do vyhrazených bitů CR4,
- jakékoli jiné porušení pravidel systému ochrany.

INT 14 Výpadek stránky (Page Fault)

Výjimku generuje stránkovací jednotka (je-li zapnuta nastavením bitu PG=1 v CR0), pokud při transformaci lineární na fyzickou adresu nastala jedna z těchto událostí:

1. právě aktivní proces neměl dostatečnou úroveň oprávnění pro přístup k požadované stránce,
2. položka jedné z transformačních tabulek, použitá pro přepočet adresy, signalizovala, že požadovaná stránka není momentálně v paměti (má nastaveno P=0).

Nastala-li jedna z těchto událostí, je v CR2 uložena lineární adresa, která vyvolala přerušení, a v zásobníku je uloženo chybové slovo. Na rozdíl od výjimek 10 až 13 má zde chybové slovo jiný tvar. Indikuje, vznikla-li výjimka na základě výpadku stránky nebo porušením přístupových práv, zda chybná operace byla čtení nebo zápis a zda šlo o uživatelský nebo privilegovaný přístup (viz obr. 5.36). V chybovém slově výjimky 14 jsou využity pouze čtyři bity nejnižších řádů:



Obr. 5.36 Formát chybového slova předávaného výjimkou 14

P (Present) je výsledkem logického součinu bitů P obou transformačních tabulek, použitých k výpočtu této fyzické adresy. Je-li nulový, měla jedna z položek P=0.

W (Write) indikuje, o jakou operaci byl procesor požádán. V okamžiku vzniku přerušení se prováděl zápis (W=1) nebo čtení (W=0).

U (User Level) je-li nastaven, bylo požádáno o přístup ke stránce s úrovní oprávnění CPL=3. Je-li bit U nulový, bylo požádáno o přístup ke stránce s CPL<3.

R (Reserved) je-li nastaven, byla výjimka generována detekcí jedničky v některém z rezervovaných bitů položky stránkového adresáře nebo stránkové tabulky, která je označena jako přítomná.

Poznámky k předchozím procesorům

Tato výjimka je zavedena od procesoru Intel386. Bit R je definován od procesoru Pentium.

INT 16 Chyba koprocesoru (Processor Extension Error)

Generování této výjimky závisí na nastavení bitu NE v registru CR0 (viz též str. 62) a signálu $\overline{\text{IGNNE}}$. Pokud je NE=1, generuje se při vzniku odmaskované výjimky v FPU výjimka procesoru 16 bezprostředně před provedením následující jiné než řídicí instrukce FPU nebo instrukce WAIT.

Pokud je NE=0 a signál $\overline{\text{IGNNE}}$ je neaktivní, způsobí výskyt odmaskované výjimky v FPU zmrazení procesoru bezprostředně před provedením následující jiné než řídicí instrukce FPU nebo instrukce WAIT. Zmrazený procesor čeká na vnější přerušení, které by mělo přijít od externího řadiče přerušení jako odpověď na aktivní hodnotu výstupního signálu $\overline{\text{FERR}}$ procesoru Pentium nebo Intel486 (signál

má podobný význam jako $\overline{\text{ERROR}}$ v Intel387). Signál $\overline{\text{FERR}}$ se aktivuje při vzniku výjimky v FPU bez ohledu na nastavení bitu NE. Tato kombinace je určena pro kompatibilitu se staršími typy procesorů, jimž byla chyba koprocesoru oznamována externím přerušením.

Pokud je NE=0 a signál $\overline{\text{IGNNE}}$ je aktivní, procesor žádosti o přerušení ignoruje a výjimka nenastane.

Poznámky k předchozím procesorům

Koprocesory 8087, Intel287 a Intel387 transportovaly žádost o přerušení do procesoru prostřednictvím řadiče přerušení. V procesorech Intel486 a Pentium tomu tak již není. Pro kompatibilitu slouží signál $\overline{\text{FERR}}$.

Interpretace pro Intel286 a Intel386: Je-li v době čekání na dokončení operace koprocesoru detekována chyba aktivní hodnotou signálu $\overline{\text{ERROR}}$, vyvolá procesor tuto výjimku. Chyba vznikla během výpočtu v koprocesoru (např. u matematického koprocesoru: chybná operace, přeplnění, dělení nulou atd.).

INT 17 Kontrola zarovnání (Alignment Check)

Výjimka je vyvolána při současném splnění čtyř podmínek:

1. Přerušení musí být povoleno nastavením bitu AM=1 v registru CR0.
2. Přerušení musí být povoleno nastavením příznaku AC=1 v příznakovém registru EFLAGS (viz str. 60).
3. Proces běží na nejnižší úrovni oprávnění (CPL=3).
4. Procesor rozpoznal pokus o přístup k operandu v paměti, který nezačíná na adrese dělitelné délkou zpřístupňovaného operandu. Tabulka délek operandů je na obr. 5.37.

Kontrola při AC=1 se provádí pouze pro ty procesy, které pracují na úrovni oprávnění CPL=3. Má-li proces CPL<3, je nastavení AC ignorováno a výjimka 17 není v žádném případě generována. Výjimka 17 předává 32bitové chybové slovo s nulovým obsahem.

Poznámky k předchozím procesorům

Výjimka je zavedena od procesoru Intel486.

Objekt	*Adresa musí být dělitelná*
Slovo	2
Dvojslovo	4
Reálné číslo v jednoduché přesnosti (Short Real)	4
Reálné číslo v dvojnásobné přesnosti (Long Real)	8
Reálné číslo v rozšířené přesnosti (Temp Real)	8
Selektor	2
48bitový ukazatel (16b segment, 32b offset)	4
32bitový offset	4
Segment v 32bitovém tvaru	2
48bitový pseudo-popisovač	4
Ukládací oblast FSTENV/FLDENV	4 nebo 2
Ukládací oblast FSAVE/FRSTOR	4 nebo 2
Bitový řetězec	4

Obr. 5.37 Hranice zarovnání objektů v paměti

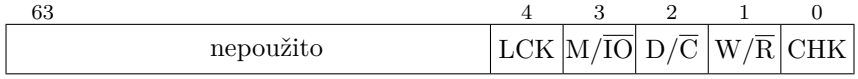
INT 18 Chyba v procesoru (Machine Check Exception)

Výjimka je dostupná pouze na procesorech Pentium a nemusí být v novějších typech. Oznamují se jí chyby parity a jiné chybové stavy technického vybavení. Generování výjimky je nutné zapnout bitem MCE v registru CR4 (viz str. 64).

Výjimka Machine Check Exception je typu Abort, tj. po vzniku této výjimky nelze již v přerušené instrukci pokračovat. Výjimka se generuje ze dvou příčin: při detekci chyby parity během čtení s nastaveným signálem $\overline{\text{PEN}}$ a při rozpoznání aktivní úrovně na signálu $\overline{\text{BUSCHK}}$.

Pro bližší určení chyby slouží dva registry: **Machine Check Address Register** a **Machine Check Type Register**. Existence obou těchto registrů závisí na konkrétním modelu procesoru. Oba registry jsou 64bitové a jsou určeny pouze ke čtení. Lze je číst instrukcí RDMSR (viz str. 284).

Registr Machine Check Address obsahuje fyzickou adresu místa, na kterém se chyba rozpoznala. Registr Machine Check Type místo s chybou blíže popisuje. Formát tohoto registru je na obr. 5.38. Je-li bit CHK nulový, je ostatní obsah obou registrů nedefinovaný. Přečtením registru Machine Check Type (instrukcí RDMSR) se bit CHK nuluje. Proto se musí nejdříve číst registr Machine Check Address a teprve potom Machine Check Type.



Obr. 5.38 Formát registru Machine Check Type

Jednotlivé bity registru Machine Check Type mají tento význam:

CHK nastavuje na 1 procesor při naplnění obsahu registru. Nuluje se čtením registru instrukcí RDMSR.

W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$ jsou zkopírované stavy stejnojmenných signálů pro bližší určení operace, která výjimku způsobila.

LCK nastavuje procesor na 1 při rozpoznání aktivní úrovně na signálu $\overline{\text{LOCK}}$.

Poznámky k předchozím procesorům

Výjimka je zavedena od procesoru Pentium.

6. Počáteční nastavení a přepínání režimů procesoru

Po rozpoznání aktivní úrovně na signálu RESET se procesor nastavuje do počátečního stavu. Instrukční ukazatel se naplní níže popsaným obsahem a ukazuje tak na instrukci, která se v dalším kroku provede. V tomto stavu procesor také předává informace o typu a verzi procesoru, ze kterých lze zjistit, jaké funkce procesor nabízí. Také další registry procesoru se naplní známým obsahem, vyprázdní se vnitřní vyrovnávací paměti, TLB a další vnitřní paměti procesoru.

Procesor Pentium má vedle signálu RESET také signál INIT. Význam signálu INIT je podobný významu signálu RESET s tím, že INIT nevyvolá operaci vyprázdnění vnitřní vyrovnávací paměti, registrů MSR (Model Specific Registers) a stavu FPU. Signál se používá na přepnutí z chráněného režimu do reálného se zachováním obsahu vyrovnávacích pamětí. TLB se při rozpoznání aktivní úrovně signálu INIT vyprázdní. Shrnutí významu signálů RESET a INIT poskytuje tabulka na obr. 6.1.

RESET	INIT	Spustí se BIST?	Efekt pro IVP	Efekt pro registry FPU	Efekt pro BTB a TLB
0	0	-	-	-	-
0	1	ne	žádný	žádný	vyprázdnění
1	0	ne	vyprázdnění	vyprázdnění	vyprázdnění
1	1	ano	vyprázdnění	vyprázdnění	vyprázdnění

Obr. 6.1 Významy signálů RESET a INIT

6.1 Test procesoru (BIST)

Po zapnutí napájení procesoru se může spustit vestavěný test (Built-In Self Test BIST). Test se spouští současným nastavením signálů RESET a INIT. Tento test trvá asi 2^{19} hodinových cyklů (počet se může měnit podle typu procesoru a výrobce si vyhrazuje právo dobu měnit bez oznámení změny). Registr EAX se nuluje

Registr	RESET bez BIST	INIT
EFLAGS[1]	0000 0002h	0000 0002h
EIP	0000 FFF0h	0000 FFF0h
CR0	6000 0010h	pozn. 2
CR2/CR3/CR4	0	0
CS	selektor=0F000h báze=0FFFF0000h limit=0FFFFh př. práva=přítomný, čtení i zápis, zpřístupněn	selektor=0F000h báze=0FFFF0000h limit=0FFFFh př. práva=přítomný, čtení i zápis, zpřístupněn
SS, DS, ES, FS, GS	selektor=0 báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis, zpřístupněn	selektor=0 báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis, zpřístupněn
EDX	0000 05xxh	0000 05xxh
EAX	0[3]	0
EBX, ECX, ESI, EDI, EBP, ESP	0	0
LDTR	selektor=0 báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis	selektor=0 báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis
GDTR, IDTR	báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis	báze=0 limit=0FFFFh př. práva=přítomný, čtení i zápis
DR0, DR1, DR2, DR3	0	0
DR6	FFFF 0FF0h	FFFF 0FF0h
DR7	0000 0400h	0000 0400h
Time Stamp Counter	0	nezměněn
Control and Event Select	0	nezměněn
TR12	0	nezměněn
Ostatní MSR (Model Specific Registers)	nedefinovány	nezměněny
TLB	vyprázdněny	vyprázdněny

Obr. 6.2 Obsahy registrů po inicializaci procesoru (1)

[1] Horních 10 bitů registru EFLAGS není po zapnutí definováno.
[2] Nezměněny jsou CD a NW, bit 4 je nastaven na 1, ostatní bity jsou nulové.
[3] Pokud je aktivován vestavěný test, je EAX nulové pouze při bezchybném ukončení testu.

Obr. 6.3 Obsahy registrů po inicializaci procesoru (2)

při úspěšném ukončení testu. Nenulový obsah EAX po dokončení testu signalizuje nalezenou závadu. Neprováděl-li se test, je EAX vždy nulové.

6.2 Obsah registrů

Do registru EDX se po ukončení testu ukládá informace o typu a verzi procesoru. Tuto vlastnost procesory Intel poskytují od typu Intel386. Intel386 vrací DH=3, Intel486 vrací DH=4 a Pentium vrací DH=5. Různé verze těchto procesorů mohou obsahovat jiné hodnoty (např. Intel386SX obsahuje 23h). V registru DL je uloženo číslo verze procesoru. Horní slovo registru EDX je vyhrazeno.

Registr CR0 má po nastavení procesoru do počátečních podmínek tento následující obsah: PG=0 (stránkování je vypnuté), CD=1 (IVP je vypnutá), NW=1 (nezapisuje se do IVP), AM=0 (kontrola zarovnání je vypnutá), WP=0 (režim kompatibility stránkové ochrany), NE=0 (vnější hlášení chyb FPU), TS=0 (nebylo přepnutí procesu), EM=0 (nesleduje se použití ESC), MP=0 (nesleduje se použití WAIT), PE=0 (reálný režim).

6.3 První provedená instrukce

Při nastavení procesoru do počátečních podmínek se nastavuje reálný režim procesoru. Podle pravidel tohoto režimu se také vyčíslí adresa první instrukce. V reálném režimu se při každé změně segmentového registru okamžitě mění obsah báze. Nové hodnotě báze se automaticky přiřadí šestnáctinásobek segmentového registru. Výjimkou je stav po inicializaci procesoru: selektor CS obsahuje 0F000h a báze CS obsahuje 0FFFF 0000h. Při první změně obsahu selektoru CS se báze automaticky vyčíslí podle standardního pravidla (tj. šestnáctinásobek selektoru CS). Výsledkem inicializace procesoru je provedení instrukce na fyzické adrese: CS.báze+EIP=0FFFF FFF0h. Tato adresa ukazuje o 16 slabik níže, než je horní konec fyzické paměti. Zde musí být permanentní paměť (EPROM, PROM apod.) s inicializační instrukcí.

První instrukce vzdáleného skoku nebo vzdáleného volání podprogramu naplní selektor CS. Pokud se toto provede v reálném režimu, naplní se dolních 20 bitů báze

novou hodnotou. Zbylých horních 12 bitů báze je nulových. Z toho plyne, že cílová adresa je v intervalu 0 až 1 M−1.

Báze ostatních segmentových registrů jsou po inicializaci procesoru nastaveny na nulu, tj. na začátek fyzické paměti. Zde by měla být paměť RAM.

6.4 Inicializace FPU

Programové rutiny spuštěné po inicializaci procesoru musí rozpoznat přítomnost či absenci FPU v systému a podle toho nastavit příznaky v registru CR0.

Stav FPU po zpracování signálu RESET se liší od stavu, ve kterém se FPU zanechává po provedení instrukce FNINIT nebo FINIT. Stav po FNINIT/FINIT je na obr. 6.4.

Pole	Hodnota	Interpretace
Řídicí slovo	037Fh	
(Nekonečno)*	0	
Zaokrouhlování	00	Zaokrouhlení na nejbližší
Přesnost	11	64 bitů (Extended)
Maskování výjimek	111111	Výjimky maskovány
Stavové slovo	0000h	
(Obsazen)	0	–
Kód podmínky	0000	–
Vrchol zásobníku	000	Na vrcholu je reg. 0
Přehled výjimek	0	Žádné výjimky
Příznak zásobníku	0	–
Příznaky výjimek	000000	Žádné výjimky
Doplňující slovo	FFFFh	
Registry	11	Prázdné
Registry	Nezměněny	Nezměněny
Ukazatele výjimek		
Kód instrukce	0	Vynulován
Adresa instrukce	0	Vynulován
Adresa operandu	0	Vynulován

* Procesory Pentium, Intel486 a Intel386 neřídí zpracovávání nekonečna.

Obr. 6.4 Stav FPU po provedení FINIT nebo FNINIT

Po zpracování signálu RESET jsou registry ST0 až ST7 nulové s nastaveným příznakem nuly (01). Signál INIT stav FPU nemění. Nastavování bitů MP a EM je popsáno na str. 61 a 98.

6.5 Přepnutí do chráněného režimu

Chráněný režim se sice zapne nastavením bitu PE v registru CR0 na jedničku, ale ještě předtím je nutno připravit celou řadu podmínek. Nastavení bitu PE bez předchozí přípravy vede s největší pravděpodobností k havárii systému.

Nejprve, ještě v reálném režimu, je potřeba připravit tabulky GDT a IDT. Tabulky LDT a TSS vytváříme zpravidla až v chráněném režimu, protože registry LDTR a TR nejsou v reálném režimu přístupny. Do registru GDTR zavedeme lineární adresu (bázi a limit) GDT a stejně tak do IDTR údaje o IDT.

Změna obsahu IDTR může způsobit problémy, protože IDTR je funkční i v reálném režimu. Změnou obsahu IDTR se znepřístupní tabulka přerušovacích vektorů a případné přerušení vygenerované po změně IDTR způsobí zhroucení systému. Proto je vhodné zakázat přerušení (IF:=0) a odložit změnu IDTR na okamžik těsně před zapnutím chráněného režimu. Ani tak není tato metoda bezpečná, protože nulová hodnota příznaku IF nezakazuje uplatnění NMI, výjimek (a instrukce INT *n*, ale tu tam snad programátor nedá).

Nyní lze **zapnout chráněný režim** instrukcí MOV CR0,*operand* s takovým operandem, který nastaví bit PE na jedničku. Lze použít i instrukci LMSW, ta ovšem zapisuje pouze do dolních 16 bitů a nemůže ovlivnit nastavení bitu PG (viz dále). Následující instrukcí by měl být blízký skok (např. na následující instrukci), kterým se vyprázdní fronta předvybraných instrukcí (musí se plnit znovu instrukcemi vybranými už podle pravidel chráněného režimu).

Má-li počítač instalovánu vnější vyrovnávací paměť (Cache), může někdy docházet k problémům v okamžiku přepínání do/z chráněného režimu. Vyrovnávací paměť musí být po přepnutí vyprázdněna (její obsah zneplatněn). Některé typy pamětí nereagují na přepnutí procesoru do jiného režimu tím, že by se automaticky vyprázdnily, a proto je musí vyprázdnit programátor. Tato operace by měla být popsána v dokumentaci příslušné vyrovnávací paměti.

V tomto okamžiku obsahuje dosud všech 6 segmentových registrů hodnoty, které byly nastaveny v reálném režimu. Po přepnutí do chráněného režimu však stále ukazují na stejné 64 KB segmenty, na které ukazovaly v reálném režimu. Toto platí až do okamžiku změny jejich obsahu. Jinými slovy, obsah všech segmentových registrů se po přepnutí do chráněného režimu interpretuje stejně jako v reálném režimu až do jejich změny.

Jakmile je segmentový registr v chráněném režimu změněn, je jeho neviditelná část naplněna z GDT (nebo LDT) a jím segmentované adresy jsou již podle pravidel chráněného režimu. Programátor by měl nastavit nové obsahy všem segmentovým registrům co nejdříve po přepnutí do chráněného režimu. Pokud by před změnou obsahu segmentových registrů nastalo přerušení, byl by jejich obsah (minimálně obsah CS) uložen do zásobníku. Problém nastane v okamžiku jejich obnovení ze zásobníku, protože nová hodnota by se chápala již podle pravidel adresace v chráněném režimu.

Segmentový registr CS lze změnit instrukcí nepodmíněného vzdáleného skoku (např. na následující instrukci). Souběžně se změnou segmentového registru SS je nutné nastavit také obsah ESP, jinak se bude stále používat zásobník reálného režimu. Pro tento účel je vhodné použít instrukci LSS, která plní oba registry zároveň. Doporučená posloupnost kroků po přepnutí do chráněného režimu je na obr. 6.5.

- Blízký skok (vyprázdní instrukční frontu).
- Naplnění DS, ES, FS, GS.
- Naplnění SS, ESP.
- Vzdálený skok (naplní CS).
- Naplnění TR.
- Naplnění LDTR.

Obr. 6.5 Doporučené činnosti po přepnutí do chráněného režimu

6.5.1 Víceúlohové zpracování

Budou-li se v chráněném režimu přepínat procesy, musí se připravit TSS. TSS lze nachystat už v reálném režimu nebo až v chráněném režimu. Registr TR lze naplnit až po přepnutí do chráněného režimu. Registr musí být naplněn před prvním přepnutím, protože procesor do tohoto „inicializačního“ TSS uloží stav právě aktivního procesu.

6.5.2 Zapnutí stránkování

Stránkování se zapíná nastavením jedničky do bitu PG v registru CR0. Může být zapnuto po přepnutí do chráněného režimu nebo souběžně s přepnutím (při jednom plnění registru CR0). Doporučuje se stránkování zapínat až v chráněném režimu. Před zapnutím stránkování musí být vytvořen stránkový adresář a stránkové tabulky. Potom se musí naplnit obsah registru CR3 adresou stránkového adresáře.

Při zapínání stránkování by si programátor měl být vědom toho, že instrukce, které stránkování zapínají, musí po zapnutí stránkování ležet ve stejném fyzickém ad-

resovém prostoru jako před zapnutím. Jinými slovy, právě procesorem zpracovávaná stránka musí před i po zapnutí stránkování být mapována do stejného prostoru.

Vždy po zapnutí stránkování musí programátor provést instrukci nepodmíněného blízkého skoku, čímž vyprázdní frontu předvybraných instrukcí a zajistí odpovídající transformaci adres stránkovací jednotkou.

6.6 Přepnutí do reálného režimu

Procesory od Intel386 dovolují přepnout z chráněného režimu zpět do reálného i jinak než signálem RESET a inicializací procesoru. Vlastní přepnutí se provede vynulováním bitu PE v registru CR0. Předtím se však musí připravit vše potřebné pro správný chod reálného režimu.

Je-li zapnuta stránkovací jednotka, musí se tato vypnout nejdříve. Před vypnutím je potřeba zajistit, aby se právě zpracovávaná stránka mapovala i po vypnutí stránkování stejně. Stránkování se vypne vynulováním bitu PG v registru CR0. Je vhodné vynulovat registr CR3, aby se zrušil obsah TLB stránkovací jednotky.

Dále se musí nastavit obsahy segmentových registrů tak, aby ukazovaly na popisovače, které adresují datové segmenty podobné segmentům reálného režimu. Tj. Limit=0FFFFh, G=0, ED=0, W=1, P=1. Ukazatelem na takové popisovače naplníme segmentové registry DS, ES, FS, GS a SS. Instrukcí vzdáleného skoku naplníme registr CS ukazatelem na popisovač instrukčního segmentu majícího podobné parametry jako ve výše uvedeném případě. Zakažte přerušení vynulováním příznaku IF a vynulujte bit PE v registru CR0.

Po vynulování bitu PE je žádoucí provést instrukci vzdáleného skoku pro vyprázdnění instrukční fronty a pro správné nastavení registru CS. Tato operace není v procesoru Pentium nutná, pro kompatibilitu s nižšími typy je však vhodné ji provádět.

Ihned po přepnutí do reálného režimu se musí naplnit registr IDTR správným obsahem (např. viz obr. 6.2). Potom povolte přerušení. Doporučený postup při přepínání do reálného režimu je na obr. 6.6.

- Je-li PG=1, proveď vnořené kroky.
  - Zajištění stejného mapování stránky i po vypnutí.
  - Nastavení PG=0.
  - Nastavení CR3=0.
- Naplnění DS, ES, FS, GS a SS.
- Vzdálený skok pro naplnění CS.
- Nastavení PE=0.
- Vzdálený skok.
- Naplnění IDTR.

Obr. 6.6 Doporučené činnosti při přepínání do reálného režimu

7. Ladicí nástroje

Procesory Intel mají od typu Intel386 do detailů propracované ladicí nástroje. Z procesoru 8086 známe dva ladicí prostředky: krokovací režim (INT 1) a ladicí bod (INT 3). Procesor Intel286 ladicí možnosti rozšířil v chráněném režimu o kontrolu dodržování limitů segmentů, respektování typové kompatibility operací nad segmenty a dodržování pravidel systému ochran. Procesor Intel386 zavádí v chráněném režimu volitelné sledování přepínání procesů. Navíc ve všech režimech lze nastavovat ladicí body beze změny operačního kódu a sledovat přístupy k vybraným datům v paměti. Na podporu posledně vyjmenovaných kontrol je v Intel386 instalována sada ladicích registrů DRi. Procesor Pentium ladicí možnosti rozšiřuje o sledování přístupů k V/V branám.

Do ladicích registrů se nastavují lineární adresy sledovaných míst. Výhodou je, že se nemusí přistupovat k vlastnímu kódu. V chráněném režimu by se v takovém případě musel definovat další datový segment, který by překryl instrukční, aby bylo možné vůbec do programu něco zapisovat. Díky registrům lze také sledovat i programy umístěné v pamětech ROM.

Krokovací režim byl popsán v rámci popisu výjimky 1 na str. 96 a **ladicí bod** v rámci popisu výjimky 3 tamtéž.

Chráněný režim od procesoru Intel286 poskytuje ladicímu systému možnost kontroly limitů segmentů, typové kompatibility operací nad segmenty a dodržování pravidel přístupů k segmentům převážně prostřednictvím výjimky 13 a ve speciálních případech prostřednictvím výjimek 10 až 12 (viz str. 99).

7.1 Sledování přepínání procesů

Možnost sledovat přepínání procesů je výhodná zvláště při ladění víceúlohových systémů. Sledování se zapne nastavením bitu **T** (Trap) v TSS (viz str. 84). Jakmile procesor při přepnutí procesu rozpozná, že nový proces má v TSS nastaven bit T=1, vygeneruje ladicí výjimku 1 ještě dříve, než se provede první instrukce nového procesu. Z toho plyne, že výjimka 1 se již obslouží v kontextu nového procesu za předpokladu, že výjimka se neobsluhuje vlastní branou zpřístupňující jiný TSS. Je-

li ladicí výjimka obsluhována tímto způsobem (přepnutím procesu), je logické, že takový obslužný proces nesmí mít T=1.

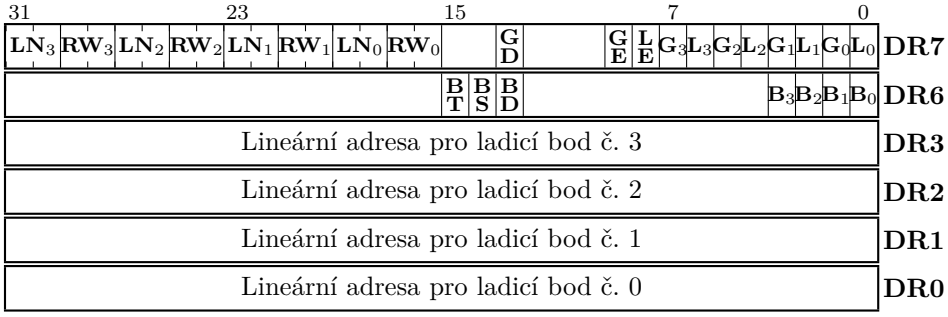
Samotné nastavení bitu T na jedničku nevyvolá žádnou aktivitu ladicích nástrojů. Jeho hodnota se čte pouze při přepnutí na tento proces. Bit T není procesorem nulován. Pokud již ladicí program nepotřebuje sledovat aktivaci tohoto procesu, musí T vynulovat sám.

V okamžiku vygenerování výjimky 1 ukazuje obsah zásobníku (EFLAGS, CS, EIP) na první instrukci nového procesu – na tu instrukci, která dosud nebyla provedena. EFLAGS, CS, EIP (v zásobníku) v tomto případě odpovídají hodnotám uloženým v TSS nového procesu.

7.2 Ladicí registry

Procesor Pentium disponuje šesti 32bitovými **ladicími registry**. V registrech **DR0 až DR3** mohou být uloženy lineární adresy ladicích bodů. Je-li zapnuto sledování a procesor má zpřístupnit obsah jedné z těchto adres, je generována ladicí výjimka 1.

Registr **DR7** je **příkazový registr** ladicího systému procesoru. Obsahuje tyto bity (viz obr. 7.1, i je v intervalu 0 až 3):



Obr. 7.1 Ladicí registry DR0 až DR3, DR6 a DR7

L$_i$ (Local Enable) je-li nastaven na jedničku, je adresa v registru DRi kontrolována pouze v rámci právě aktivního procesu. Přepnutím procesu je tento bit vynulován a musí být obnoven programově.

G$_i$ (Global Enable) je-li nastaven na jedničku, je adresa v registru DRi kontrolována ve všech procesech bez ohledu na jejich přepínání. Tento bit není procesorem nulován.

Bity L_i a G_i zapínají kontrolování zpracovávaných adres na shodu s obsahem registrů DR0 až DR3. Je-li L_i=0 a zároveň G_i=0, není obsah registru DR*i* použit.

RW$_i$ kvalifikují typ operace, která při shodě adresy s registrem DR*i* vyvolá ladicí výjimku. Čtyři možné kombinace těchto dvou bitů jsou popsány na obr. 7.2. Interpretace kombinace „10“ se liší podle hodnoty bitu DE[1] (Debug Extensions) v registru CR4 (viz též str. 64). Je-li DE=0 (mód kompatibilní s Intel386 a Intel486), je kombinace „10“ nepoužita. Je-li DE=1, kombinace „10“ znamená, že v registru DR*i* je uložena V/V adresa a sleduje se přístup k bráně nebo dvěma sousedním branám.

LN$_i$ kvalifikují délku operandu, který při shodě adresy operandu s registrem DR*i* vyvolá ladicí výjimku. Je-li velikost operandu 16bitové slovo, musí operand začínat na adrese dělitelné dvěma, jde-li o 32bitové dvojslovo, musí začínat na adrese dělitelné čtyřmi. Kombinace jsou opět popsány na obr. 7.2.

RW	Typ operace
00	Výběr instrukce
01	Zápis dat
10	Nepoužito při DE=0
10	Přístup k V/V bráně při DE=1
11	Čtení nebo zápis dat

LN	Délka operandu
00	Slabika nebo instrukce
01	16bitové slovo
10	Nepoužito
11	32bitové dvojslovo

Obr. 7.2 Obsah bitů RW a LN v registru DR7

LE (Local Exact) je procesorem Pentium a Intel486 ignorován. V Intel386 sděluje, kdy má procesor provádět testování shody adresy. Procesor Intel386 pracuje s proudovou architekturou, která vybírá instrukci nebo operand o několik strojových taktů dříve, než jej zpracovává. Je-li LE=0, testuje se shoda v okamžiku výběru z paměti. V tom případě se může stát, že se neodhalí ladicí bod umístěný bezprostředně (nebo blízko) za nastavením adresy do registru DR0 až DR3. Při LE=1 se kontrola provádí v okamžiku použití vybraného paměťového místa jen pro právě aktivní proces. Přepnutím procesu je bit LE nulován.

GE (Global Exact) je procesorem Pentium a Intel486 ignorován. V Intel386 má stejnou funkci jako LE s platností i po přepnutí procesu.

GD (Global Debug Access) nastavením na jedničku se zakazují veškeré další přístupy (zápis i čtení) ke všem ladicím registrům.

[1] Bit DE zavádí až procesor Pentium.

Má-li lineární adresa ladicího bodu ukazovat na instrukci, která je delší než slabika (příp. má i instrukční prefixy), musí ukazovat na první slabiku instrukce nebo na první z jejích prefixů (má-li nějaký). Není-li toto pravidlo dodrženo, ladicí bod nebude rozpoznán.

Registr **DR6** je **stavový registr** ladicího systému. Nutnost existence takového registru bylo patrná již při výkladu výjimky 1, jejíž aktivaci způsobuje pět různých událostí. Stavový registr indikuje, co bylo příčinou vyvolání této ladicí výjimky. Jeho obsah procesor nenuluje, to je záležitostí programového vybavení. Pokud by ladicí program bity stavového registru nenuloval, nebylo by možné výjimku identifikovat, protože by se příznaky kumulovaly. Jednotlivé bity (viz obr. 7.1) mají tento význam:

B_i (Breakpoint i Hit) je nastaven procesorem na jedničku tehdy, byla-li ladicí výjimka vyvolána shodou adresy s registrem DRi.

BD (Break for Debug Register Access) je nastaven na jedničku tehdy, byla-li ladicí výjimka vyvolána přístupem k ladicím registrům po nastavení GD=1.

BS (Break for Single-Step) je nastaven na jedničku, byla-li ladicí výjimka vyvolána krokovacím režimem procesoru (TF=1).

BT (Break for Task Switch) je nastaven na jedničku tehdy, byla-li ladicí výjimka vyvolána přepnutím na proces, který má ve svém TSS nastaven bit T=1.

Poznamenejme, že žádný bit DR6 neoznamuje detekování programového ladicího bodu (0CCh – INT 3), protože takové body nejsou obsluhovány výjimkou 1.

Rutina obsluhující výjimku 1 může zjistit, že v DR6 jsou všechny indikační bity nulové nebo je více než jeden nenulový bit. První případ nastane tehdy, bylo-li přerušení 1 vyvoláno vnějším technickým zařízením nebo instrukcí INT 1 (nikoli ladicími prostředky procesoru). Více než jeden bit může být nenulový tehdy, vyvolalo-li ladicí výjimku více podmínek splněných současně.

V takovém případě není procesor příliš důsledný v nastavování obsahu DR6 a může se stát, že budou nastaveny i ty B_i bity, jejichž sledování nebylo zapnuto. Všeobecně se doporučuje, aby obslužná rutina před analýzou situace vynulovala ty B_i bity, které mají odpovídající G_i a L_i nulové.

Jednou z výhod ladicího mechanismu je to, že ladicí body označené L_i jsou platné pouze v právě aktivním procesu, nikoli po přepnutí na jiný. Přepnutím se všechny bity L_i nulují a na rozdíl od ostatních registrů (všeobecné, segmentové, …) se ladicí registry s přepnutím procesu neukládají do TSS ani jinam. Ladicí program musí aktuální hodnoty udržovat sám.

Jedna z metod pro udržování hodnot L_i spočívá ve využití možnosti přerušení po přepnutí na proces označený T=1. Ladicí program nastaví indikátor T v TSS toho procesu, ve kterém má L_i udržovat. Jakmile je přepnuto provádění na takový proces, generuje se INT 1 (obslužná rutina detekuje BT=1) a pak se mohou nastavit

odpovídající hodnoty L_i. Obsah všech ladicích registrů může být pro daný proces uložen do rozšíření TSS (tj. nad limit 103). Připomeňme, že rutina musí pracovat v rámci stejného procesu, pro který L_i nastavuje. Kdyby pracovala v jiném procesu, potom by jeho přepnutím byly L_i vynulovány.

Pokud pracujeme ve víceúlohovém systému, můžeme místo L_i nastavit G_i, které nejsou přepnutím procesu dotčeny. Je vhodné je používat jenom tam, kde se jednotlivými procesy nepřekrývá lineární adresový prostor.

V procesorech Intel386 a Intel486 byly ladicí registry **DR4** a **DR5** mapovány na registry DR6 a DR7, ačkoli nebyly zdokumentovány. Je-li v registru CR4 nastaven bit DE=0, jsou stejně mapovány i v procesoru Pentium. Pokud je však nastaveno DE=1, generuje se při pokusu o přístup k registrům DR4 a DR5 výjimka nedefinovaného operačního znaku.

7.3 Příznak RF

Je-li výjimka 1 způsobena shodou obsahu jednoho z registrů DR*i* s adresou právě dekódované instrukce, je do zásobníku uložena adresa takto označené instrukce. Po obsloužení výjimky 1 ukončí rutina svoji činnost instrukcí IRET, která předá řízení na instrukci, jejíž adresa je uložena v zásobníku. Jenže tato instrukce má adresu shodnou s obsahem jednoho z registrů DR*i* a celý proces by se mohl donekonečna opakovat. Aby se takové situaci zabránilo, byl zaveden příznak **RF** (Resume Flag), který je uložen v příznakovém registru (viz obr. 5.11 na str. 59).

Příznak RF je nastaven na jedničku jenom při přerušení typu Fault (přerušení Fault do zásobníku uloží adresu instrukce, která přerušení způsobila). Do této kategorie patří i výše zmíněná ladicí výjimka. RF není nastaven přerušeními vyvolanými technickými prostředky, instrukcí INT a přerušeními klasifikovanými Trap a Abort.

Je-li RF nastaven, jsou ignorovány všechny výjimky ladicích prostředků procesoru. Příznak je umístěn v příznakovém registru proto, aby byl výjimkou uložen do zásobníku a posléze obnoven instrukcí IRET, a tím byla potlačena ladicí výjimka spojená s instrukcí, která se má po IRET provést.

Příznak RF se nuluje automaticky po každém úspěšném dokončení instrukce. Z toho plyne, že RF není nastaven déle než po dobu trvání jedné instrukce. Příznak RF potlačuje pouze výjimky ladicího systému a nejsou jím dotčena vnější přerušení ani výjimky generované procesorem (obecná chyba ochrany apod.).

7.4 Ladicí body

Nastavením lineární adresy do jednoho z registrů DR0 až DR3 a příslušných bitů do registru DR7 zapínáme sledování aktivity procesoru, která se dotýká této lineární adresy. Sledujeme-li výběr instrukce z této adresy (RW=0 a současně musí být LN=0), musí lineární adresa ukazovat na první slabiku této instrukce nebo, má-li instrukce prefixy, na první z jejích prefixů. Délka instrukce nerozhoduje.

Sledujeme-li čtení nebo zápis dat (RW=1 nebo 3), musí být správně nastavena velikost sledovaného objektu. Sledování slabiky (LN=0) je jednoduché. Sledujeme-li slovo (LN=1) nebo dvojslovo (LN=3), musí tento objekt ležet na adrese dělitelné 2 (pro slovo) nebo 4 (pro dvojslovo). Je-li LN=1 nebo 3, ignorují se nejnižší 1 nebo 2 bity lineární adresy v DRi. Hodnotu LN procesor použije pro maskování nejnižších dvou bitů. Potřebujeme-li sledovat slovo nebo dvojslovo, které neleží na adrese dělitelné 2 nebo 4, musíme nastavit dva ladicí body kratších délek, první ukazující na nižší část a druhý na vyšší část objektu.

Při sledování přístupů k V/V branám lze sledovat pouze 8 nebo 16bitové přístupy, přičemž adresa musí být zarovnána na hranici slova. V/V adresy se pro účely porovnávání neznaménkově rozšiřují z 16 na 32 bitů.

Zarovnání adresy objektů delších než slabika a ignorování nižších 1 nebo 2 bitů lineární adresy v DRi dovolí procesoru zachytit i takové odkazy, které zpřístupňují jen část objektu. Např. leží-li dvojslovo na adresách 1000 až 1003 a sledujeme je nastavením DR0=1000, LN_0=3, RW_0=3, potom procesor zachytí i např. čtení slova ležícího na adresách 1002 a 1003.

Bity LE a GE mají následující význam pouze v procesoru Intel386: Nastavením bitů LE nebo GE v registru DR7 se zapíná testování obsahů DR0 až DR3 na shodu s právě zpracovávanou lineární adresou až v okamžiku provádění instrukce. Jsou-li LE a GE nulové, testuje se shoda o několik procesorových taktů dříve – v okamžiku výběru instrukce a dat z paměti. Testování v okamžiku provádění instrukce (LE nebo GE=1) má tu výhodu, že je ladicí bod rozeznán vždy. Testuje-li se při výběru z paměti (LE a GE=0), nebudou zachyceny ty ladicí body, které byly nastaveny až po něm. Tato situace nastává kvůli proudovému zpracování, ve kterém se instrukce vybírá paralelně s prováděním předcházející instrukce nebo předcházejících instrukcí.

Jedinou nevýhodou režimu LE nebo GE=1 je výrazné **zpomalení** provádění instrukcí, a tím snížení výkonu procesoru, a to i tehdy, jsou-li všechny L_i a G_i nulové.

Procesory Intel486 a Pentium bity LE a GE ignorují s tím, že vypínají proudové zpracování a tím zpomalují provádění již při nastavení některého z bitů Li/Gi. Zajistí tak bezpečné rozpoznání všech ladicích bodů.

7.5 Ladicí body pro datové přístupy

Aktivace ladicího bodu určeného pro sledování instrukce (označeného RW=0 a LN=0) vyvolá výjimku, kterou klasifikujeme jako Fault. Sledovaná instrukce není provedena a do zásobníku se uloží adresa této sledované instrukce.

Naopak zpřístupnění ladicího bodu určeného pro sledování dat (RW=1 nebo 3) generuje výjimku typu Trap. Výjimka se vyvolá až po dokončení instrukce a do zásobníku se uloží adresa následující instrukce. Je-li sledovaná operace zápis, je předchozí obsah paměťového místa ztracen. To však nevadí, protože pokud nás předchozí hodnota zajímala, mohli jsme si ji přečíst v okamžiku nastavování ladicího bodu.

Ladicí nástroje nejsou účinné v takovém multiprocesorovém systému, kde více procesorů sdílí jeden fyzický paměťový prostor. Tam konkrétní procesor může sledovat jenom vlastní přístupy k paměti a nemůže ladicími nástroji hlídat, zda obsah paměťového místa změnil jiný procesor.

7.6 Zákaz přístupu k ladicím registrům

Nastaví-li programátor bit GD v registru DR7 na jedničku, jsou zakázány veškeré přístupy (zápis i čtení) ke všem registrům DR0 až DR3, DR6 a DR7.

Pokus o přístup ke kterémukoli z ladicích registrů vyvolá ladicí výjimku (Fault), která vynuluje bit GD, aby bylo možné zjistit příčinu výjimky čtením DR6. Toto je také jediný způsob jak bit GD vynulovat.

Funkce je implementována proto, aby ladicí systém měl jistotu, že on jediný manipuluje s ladicími registry.

8. Interní vyrovnávací paměť

Jedním z prostředků, kterými se dociluje vysokého výkonu procesoru, jsou dvě 8KB interní vyrovnávací paměti (IVP). Jedna je určena pro vyrovnávání čtení a zápisu dat; druhá je pro vyrovnávání výběru instrukcí. Úvodem uveďme několik obecných faktů k činnosti vyrovnávací paměti. Vyrovnávací paměť je umístěna mezi procesorem a fyzickou operační pamětí. Její kapacita bývá řádově nižší než kapacita operační paměti. Přístupová doba vyrovnávací paměti je proti paměti operační kratší.

Architektura vyrovnávacích pamětí vychází z předpokladu, že instrukce a data nejsou z operační paměti vybírány úplně nahodile, ale aspoň trochu sekvenčně. Je pravděpodobnější, že po výběru instrukce se bude vybírat instrukce bezprostředně následující než kterákoli jiná instrukce. Vyrovnávací paměť je rozdělena do položek. Kapacita položky je vyšší než kapacita adresovatelné jednotky. Mezi procesorem a vyrovnávací pamětí se přenáší požadovaná adresovatelná jednotka. Mezi vyrovnávací pamětí a operační pamětí se přenáší vždy celá položka.

Vyrovnávání je dostupné ve všech pracovních režimech procesoru: reálném, chráněném a virtuálním 8086. Velikost položky vyrovnávací paměti procesoru Pentium je 32 slabik. Tyto se plní obsahem fyzické paměti vždy z adres dělitelných 32.

Do IVP může být umístěna libovolná část fyzického adresovacího prostoru. Ty části (stránky) paměti, které se do IVP nesmějí ukládat, lze označit nastavením bitu PCD ve stránkové tabulce, nebo vnější systém při výběru takové adresy musí nastavit signál $\overline{\text{KEN}}$=1.

Požadavek na čtení dat nebo instrukce obsahuje fyzickou adresu čteného místa v paměti. Tato adresa se nejprve porovnává s klíči (tj. adresami přiřazenými položkám) IVP. Pokud se shodují (tj. adresa byla v IVP nalezena) a klíč je označen jako platný, předá se odpovídající obsah z IVP a tím operace skončila. Pokud se neshodují (tj. adresa v IVP nebyla nalezena), čte se obsah zadané adresy z paměti a přečteným obsahem se rovněž plní jedna z položek IVP.

8.1 Datová IVP

Datová interní vyrovnávací paměť procesoru Pentium je řízena protokolem MESI, který zabezpečuje konzistenci dat ve spolupracující interní a externí VP a i ve více-

procesorových systémech. Každá položka může nabývat stavu podle obr. 8.1.

Stav položky	**M Modified** *změněna*	**E Exclusive** *výlučná*	**S Shared** *sdílená*	**I Invalid** *neplatná*
Je položka platná?	ano	ano	ano	ne
Kopie paměti je	neaktuální	platná	platná	–
Existují kopie ve VP jiných procesorů?	ne	ne	možná	možná
Zápis do této položky	nebyl proveden i na sběrnici	nebyl proveden i na sběrnici	byl proveden i na sběrnici a aktualizoval VP	byl proveden přímo na sběrnici

Obr. 8.1 Stavy položek VP v protokolu MESI

Jednotlivé stavy lze popsat následovně:

M - Modified značí, že položka byla změněna (tj. liší se od obsahu hlavní paměti). Tento stav smí nastat pouze u jedné VP v systému. Čtení/zápis položky označené M nemusí vyvolat tutéž operaci přes sběrnici do hlavní paměti.

E - Exclusive smí být rovněž nastaveno pouze u jedné VP v systému a navíc položka nesmí být změněna (M), tj. musí být shodná s obsahem hlavní paměti. Zápis do této položky změní její stav z E na M. Operace čtení/zápisu nemusí vyvolat činnost sběrnice.

S - Shared označuje potencionálně více procesory sdílenou položku (tj. stejná položka je ve více VP). Čtení této položky nevyvolá aktivitu sběrnice. Zápis naopak musí být proveden přes sběrnici až do paměti (Write-Through) a tento zápisový cyklus vyprázdní stejné položky v ostatních VP. Po dokončení zápisu se stav změní na E.

I - Invalid označuje neplatnou položku.

Datová vyrovnávací paměť pracuje při zápisu vedeném do paměti ve dvou různých režimech:

Write-Through je režim, ve kterém se zápis provádí souběžně do položky VP a operační paměti. Tato metoda je užitečná např. při implementaci grafické video-paměti, kde se musí průběžně aktualizovat její obsah proto, aby i obraz byl aktuální.

Write-Back je režim, ve kterém se zápis provádí pouze do vyrovnávací paměti. V tomto režimu se snižuje zatížení sběrnice eliminací řady ne nezbytných zápisů do paměti. Obsah paměti se aktualizuje až později při vyprázdňování VP operací Write-Back.

Mezi oběma režimy se přepíná buď programově bitem PWT stránkové tabulky, nebo signálem WB/$\overline{\text{WT}}$.

8.2 Instrukční IVP

V instrukční vyrovnávací paměti je z protokolu MESI implementována pouze část „SI“. Je to proto, že do instrukční IVP nelze zapisovat. Instrukční IVP monitoruje změny v datové IVP. Pokud změny zasahují kód uložený v instrukční IVP, zneplatní se zasažená položka v instrukční IVP.

8.3 Řízení IVP

IVP je řízena bity CD a NW v CR0. Pomocí nich lze zvolit až čtyři režimy práce IVP:

CD=1, NW=1. Zápis do IVP je vypnut. Při čtení nenalezené položky se hodnota předá z paměti a obsah IVP se nemění. Při čtení nalezené položky se hodnota předá z IVP. Funkci prohledávání nelze zrušit. Nebezpečí tohoto režimu spočívá v možné nekonzistenci dat v IVP a v paměti, která vzniká proto, že se data v IVP neaktualizují. Aby tento stav nenastal, je nutno IVP ihned po nastavení CD=1 a NW=1 vyprázdnit.

Faktu, že nelze zrušit prohledávání, můžeme využít pro uložení některých neměnných, opakovaně čtených hodnot z paměti. Naplnění IVP těmito hodnotami provedeme před nastavením CD=1, NW=1 nebo po nastavení pomocí registrů TR3 až TR5.

Tento režim je nastaven po inicializaci procesoru signálem RESET.

CD=1, NW=0. V tomto režimu nejsou vypnuty zápisy do IVP, ale vlastní funkce IVP je stále potlačena. Při čtení nalezené položky se hodnota předá z IVP. Při čtení nenalezené položky se hodnota předá z paměti, ale do IVP se nic neukládá. Při zápisu nalezené položky se hodnota zapisuje jak do paměti, tak i do IVP. Při zápisu nenalezené položky se provede zápis pouze do paměti.

V tomto režimu nemohou nastat nekonzistence, protože položky v IVP jsou průběžně aktualizovány. Režim se používá tehdy, musí-li programové vybavení dočasně vypnout funkci IVP.

CD=0, NW=1. Toto je nepovolená kombinace. V okamžiku zadání CD=0, NW=1 do CR0 se generuje výjimka 13 (chybové slovo 0).

CD=0, NW=0. Jde o normální režim IVP. Při čtení nalezené položky se hodnota předá z IVP, při čtení nenalezené položky se hodnota přečte z paměti, uloží do IVP a předá z IVP. Při zápisu nalezené položky se hodnota zapíše do IVP i do paměti. Při zápisu nenalezené položky se hodnota zapisuje pouze do paměti (položka se neukládá do IVP).

Kombinace hodnot bitů CD a NW shrnuje tabulka na obr. 8.2.

		Čtení položky	
CD	NW	nalezené v IVP	nenalezené v IVP
1	1	z IVP	z paměti
1	0	z IVP	z paměti
0	1	Nepovolená kombinace: vyvolá INT 13	
0	0	z IVP	z paměti do IVP, z IVP
		Zápis položky	
CD	NW	nalezené v IVP	nenalezené v IVP
1	1	do paměti	do paměti
1	0	do IVP, do paměti	do paměti
0	1	Nepovolená kombinace: vyvolá INT 13	
0	0	do IVP, do paměti	do paměti

Obr. 8.2 Kombinace bitů CD, NW a jejich funkce

8.4 Plnění IVP

Do IVP může být umístěna libovolná část fyzického adresovacího prostoru. Ty části (stránky) paměti, které se do IVP nesmějí ukládat, lze označit nastavením bitu PCD ve stránkové tabulce, nebo vnější systém při výběru takové adresy musí nastavit $\overline{\text{KEN}}$=1.

Požadavek na čtení dat nebo instrukce obsahuje fyzickou adresu čteného místa v paměti. Tato adresa se nejprve porovnává s klíči (tj. adresami přiřazenými položkám) IVP. Pokud se shodují (tj. adresa byla v IVP nalezena) a klíč je označen jako platný (V=1), předá se odpovídající obsah z IVP, a tím operace skončila. Pokud se neshodují (tj. adresa v IVP nebyla nalezena), čte se obsah zadané adresy z paměti a přečteným obsahem se rovněž plní jedna z položek IVP. Položky IVP se plní vždy po blocích délky 32 slabik.

IVP se plní pouze při čtení z paměti – nikoli při zápisu položky do paměti, která nemá svůj obraz v IVP. Má-li svůj obraz v IVP, aktualizuje se IVP a rovněž obsah místa v paměti.

8.5 Vyprázdnění IVP

Vyprázdnění IVP je zneplatnění celého obsahu IVP. Vyprázdněním jsou vynulovány bity označující platnost všech klíčů IVP. Vnější technické vybavení má pro vyprázdnění IVP k dispozici signál $\overline{\text{FLUSH}}$.

Programově lze vyprázdnění vyvolat použitím privilegovaných instrukcí INVD (Invalidate Cache) a WBINVD (Write-Back Invalidate Cache). Instrukce se liší v tom, že WBINVD před vyprázdněním přepíše obsah IVP do paměti (tj. provede operaci Write-Back).

8.6 Organizace IVP

V procesoru Pentium je 8KB IVP pro instrukce a 8KB IVP pro data. Každá je organizována jako 2cestná asociativní paměť. Paměť má 128 řádků a každá položka má 32 slabik (srovnej s organizací IVP Intel486 na obr. 8.3 na str. 126). Pro realizaci algoritmu LRU je určen vždy jeden bit pro jeden řádek. Datová IVP pracuje protokolem MESI (viz str. 122) a instrukční IVP protokolem SI.

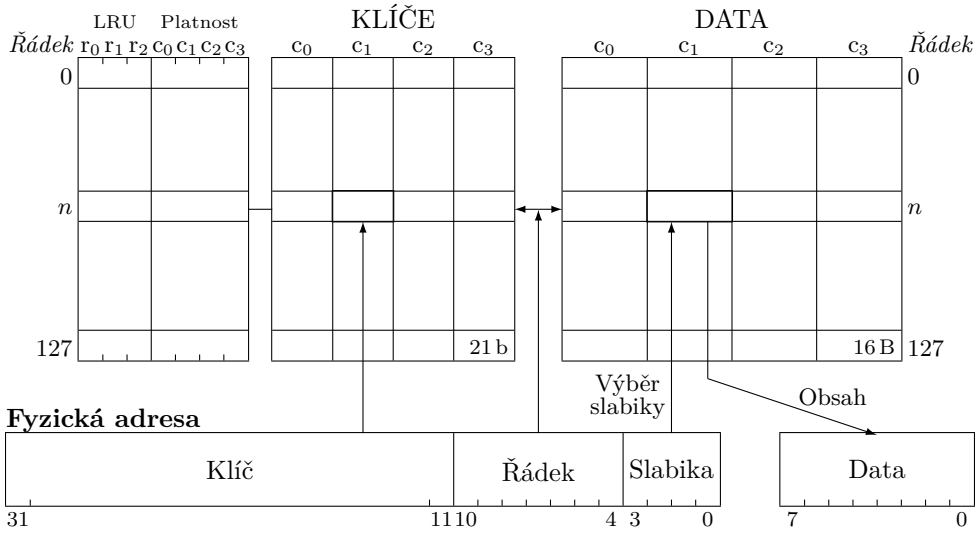
Obě IVP jsou 3portové, tj. umějí zpracovávat až 3 požadavky současně. Dva požadavky mohou přijít současně od dvou zřetězených instrukčních front „u“ a „v“ (viz str. 151) a jeden je vyhrazen pro sledování aktivity jiných procesorů na jedné sběrnici.

Poznámky k předchozím procesorům

IVP byla prvně zavedena v procesoru Intel486. Činnost IVP procesorů Intel486 a Pentium se dost liší, proto si popíšeme činnost IVP Intel486 podrobněji.

IVP procesoru Intel486 je společná pro data i pro instrukce a má kapacitu 8 KB. IVP je 4cestná asociativní paměť obsahující 4×128 klíčů. 21bitový klíč zahrnuje nejvyšších 21 bitů fyzické adresy. Ke každému klíči je přiřazeno 16 B jako obsah položky. Kapacita této datové oblasti je 8 KB. První slabika 16 B položky musí ležet na adrese dělitelné 16. Struktura a adresace IVP je patrná z obr. 8.3. Ke každému klíči je přiřazen jeden bit označující platnost obsahu (V - Valid). Každému řádku jsou navíc určeny tři bity (LRU - viz dále), pomocí kterých lze na řádku přibližně určit nejdéle nepoužitý klíč.

Každé fyzické adrese je bity 4 až 10 jednoznačně přiřazen řádek IVP (viz obr. 8.3). Má-li se IVP naplnit obsahem této fyzické adresy, testuje se, je-li některý z klíčů



Obr. 8.3 Struktura IVP Intel486

na určeném řádku neplatný (tj. má-li nastaveno V=0). Je-li, použije se pro zápis. Není-li, použije se algoritmus LRU pro určení nejdéle nepoužité položky, ta se vyřadí a nahradí právě zapisovanou.

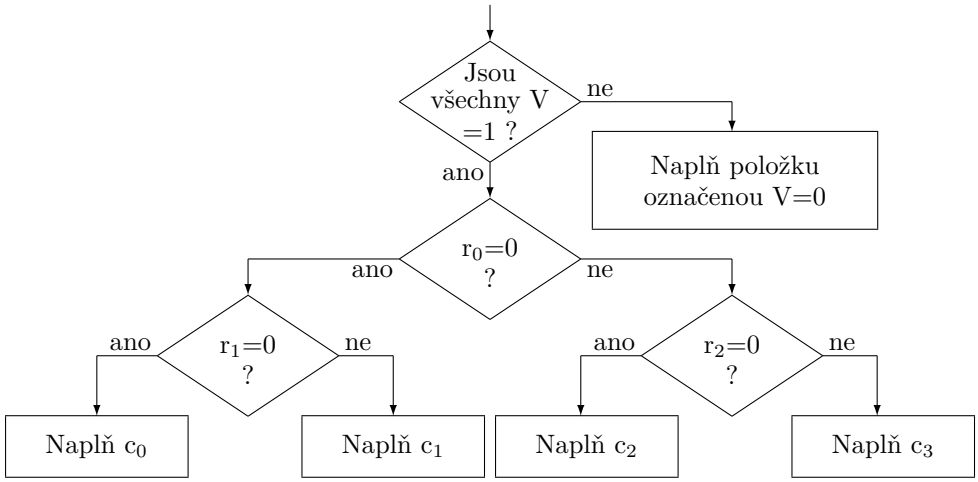
Pro realizaci LRU (Least Recently Used) jsou zde určeny 3 bity vybírající nejdéle nepoužitou ze čtyř položek. Protože rozhodovací bity jsou jenom 3, funguje algoritmus pouze částečně (pro úplnou funkci by bylo zapotřebí 6 bitů). Před vysvětlením tohoto pseudo-LRU algoritmu si označme jednotlivé cesty c_0, c_1, c_2 a c_3 a rozhodovací bity LRU r_0, r_1 a r_2. Každému použitému řádku nastavuje IVP rozhodovací bity podle tabulky na obr. 8.4.

Při použití položky z cesty	se nastaví rozhodovací bity		
	r_0	r_1	r_2
c_0	1	1	*beze změny*
c_1	1	0	*beze změny*
c_2	0	*beze změny*	1
c_3	0	*beze změny*	0

Obr. 8.4 Nastavování rozhodovacích bitů pseudo-LRU Intel486

Při plnění položkou, která nemá svůj obraz v IVP, se nejprve podle bitů 4 až 10

fyzické adresy vybere řádek IVP. Potom se postupuje podle algoritmu na obr. 8.5.



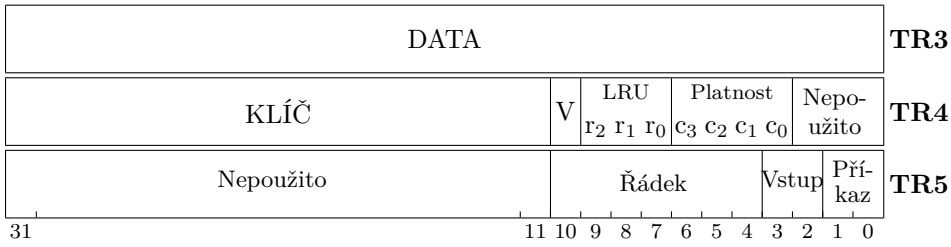
Obr. 8.5 Výběr nejdéle nepoužité položky algoritmem pseudo-LRU Intel486

Tento algoritmus správně funguje např. pro posloupnost použití: c_2, c_3, c_0, c_1. Částečně funguje např. pro posloupnost: c_2, c_0, c_1, c_3. Návrháři procesoru uznali, že vzhledem k velmi jednoduché implementaci algoritmu pseudo-LRU je taková účinnost dostačující.

Pro **testování IVP**, podobně jako pro testování TLB, je v procesoru Intel486 k dispozici sada testovacích registrů. Jejich pomocí si může programátor „osahat“ funkci IVP. Testovací registry pro IVP jsou tři: TR3 je datový registr, TR4 je stavový registr a TR5 je řídicí registr. Tvar těchto registrů je na obr. 8.6. Při testování IVP musí být její normální funkce vypnuta, tj. musí být nastaveno CD=1 a NW=1.

Registr **TR3** (Cache Data Test Register) slouží pro plnění nebo čtení datové části IVP. Datová část položky IVP, která je v tomto okamžiku reprezentována pomocným 16B datovým registrem, má velikost 16 slabik a registr TR3 pouze 4 slabiky. Z tohoto důvodu se plnění nebo čtení datové části provádí ve čtyřech krocích. Bity 2 a 3 registru TR5 sdělují, která čtvrtina datové části je právě plněna nebo čtena z nebo do pomocného 16B datového registru.

Registr **TR4** (Cache Status Test Register) obsluhuje paměť klíčů a bity „LRU“ a „Platnost“. Při plnění IVP musí být do TR4 uložen klíč a bit V. Bity „LRU“ a „Platnost“ nejsou při plnění významné. Hodnoty těchto bitů v TR4 nastavuje procesor při čtení IVP.



Obr. 8.6 Testovací registry TR3, TR4 a TR5 Intel486

Registr **TR5** (Cache Control Test Register) řídí testovací operaci. Je-li v poli „Příkaz“ zadána operace plnění nebo čtení pomocného datového registru, musí pole „Vstup“ obsahovat pořadí zadávaného dvojslova. Pole „Řádek“ je v tomto okamžiku bezvýznamné. Je-li v poli „Příkaz“ zadána operace plnění nebo čtení IVP, musí „Vstup“ obsahovat číslo cesty a „Řádek“ číslo řádku (viz obr. 8.3), kam se má obsah pomocného datového registru zapsat nebo odkud se má přečíst.

„Příkaz“		Operace
bit 1	bit 0	
0	0	Plnění nebo čtení pom. dat. registru
0	1	Plnění IVP obsahem pom. dat. registru
1	0	Čtení obsahu IVP do pom. dat. registru
1	1	Vyprázdnění celé IVP

Obr. 8.7 Řídicí bity testovacího registru TR5 Intel486

Testovací zápis do IVP se provádí ve dvou krocích. Nejprve musí programátor naplnit 16B pomocný datový registr a naplnit registr TR4 klíčem a bitem V. Poté programátor zapíše údaje do IVP.

Pomocný datový registr se plní tak, že nejprve do TR5 nastavíme indikaci „Vstup“ a „Příkaz“ (ten je pro tuto operaci nula) a potom zapíšeme příslušné dvojslovo do TR3. Tuto operaci opakujeme čtyřikrát. Po naplnění datového registru zapíšeme 21bitový klíč a bit V do TR4. Nakonec opíšeme pomocný datový registr a TR4 do IVP zadáním čísla řádku, cesty a příkazu 1 do TR5. Bity LRU jsou nastaveny vnitřní logikou IVP a jejich hodnotu nelze programově ovlivnit. Příklad testovacího plnění IVP je na obr. 8.8.

Připomeňme, že testovací zápisy se smějí do IVP provádět pouze tehdy, je-li IVP vypnuta. Protože vypnutím IVP se nezruší prohledávání, musí se při testovacím plnění dávat pozor na hodnoty klíčů, aby se testovací hodnoty neinterpretovaly jako

obsahy adres, které testovací program normálně používá. Této „nevýhody“ lze naopak využít tehdy, chceme-li IVP použít jako paměť na opakovaně čtené konstanty. Zápisem takových konstant do IVP urychlíme jejich výběr oproti klasickému výběru z paměti.

Testovací čtení IVP se rovněž provádí ve dvou krocích. Nejprve je naplněn 16B pomocný datový registr a stavový registr TR4 a potom programátor převezme obsah pomocného datového registru prostřednictvím registru TR3.

Nejprve programátor naplní registr TR5 příkazem 2, číslem řádku a cesty. Potom převezme obsah pomocného datového registru ve čtyřech krocích opakovaným zápisem do TR5 a čtením TR3. Nakonec převezme obsah stavového registru TR4, kde jsou mj. nastaveny bity „LRU“ a „Platnost“.

Uvnitř posloupnosti instrukcí, která čte obsah pomocného datového registru a stavového registru TR4, se nesmí vyskytnout operace odkazující se na operand v paměti. Takový paměťový odkaz by aktivoval IVP a tím by byl změněn obsah pomocného datového registru a TR4. Příklad testovacího čtení IVP je na obr. 8.8.

Plnění IVP

```
mov  ESI,0
mov  TR5,ESI
mov  TR3,EAX   ; dvojslovo 0
mov  ESI,4
mov  TR5,ESI
mov  TR3,EBX   ; dvojslovo 1
mov  ESI,8
mov  TR5,ESI
mov  TR3,ECX   ; dvojslovo 2
mov  ESI,0Ch
mov  TR5,ESI
mov  TR3,EDX   ; dvojslovo 3
mov  TR4,EDI   ; klíč a V
mov  ESI,1
mov  TR5,ESI   ; zápis
```

Čtení IVP

```
mov  ESI,2
mov  TR5,ESI   ; čtení
mov  ESI,0
mov  TR5,ESI
mov  EAX,TR3   ; dvojslovo 0
mov  ESI,4
mov  TR5,ESI
mov  EBX,TR3   ; dvojslovo 1
mov  ESI,8
mov  TR5,ESI
mov  ECX,TR3   ; dvojslovo 2
mov  ESI,0Ch
mov  TR5,ESI
mov  EDX,TR3   ; dvojslovo 3
mov  EDI,TR4   ; V, LRU, ...
```

Obr. 8.8 Příklad testovacího plnění a čtení IVP Intel486

Vyprázdnění IVP proběhne po nastavení příkazu 3 do TR5. V takové operaci nemají další pole TR5 ani ostatní testovací registry význam. Vyprázdněním jsou vynulovány všechny bity V a LRU v IVP. Obsahy klíčů a dat nejsou změněny.

Vyprázdnění IVP příkazem 3 zapsaným do TR5 se liší od vyprázdnění IVP instrukcemi INVD a WBINVD tím, že operace vyprázdnění není signalizována externí

VP. Je-li v počítači externí VP instalována, není příkazem zapsaným do TR5 vyprázdněna.

9. Režim správy systému

Režim správy systému (System Management Mode – SMM) dovoluje návrháři systému provádět vybrané důležité činnosti (řízení napájení počítače, správa bezpečnostních prostředků apod.) nezávisle nejenom na aplikaci, ale dokonce i na vlastním operačním systému.

SMM patří mezi základní režimy činnosti procesoru, kterými jsou chráněný, reálný a V86 režim. Na rozdíl od nich, je však určen pro firmware (tj. pro programové vybavení vestavěné do počítače) – nikoli tedy pro aplikace a systémové programy. Do SMM lze přejít z libovolného z režimů speciálním přerušením. Žádost o toto přerušení se přivádí aktivní hodnotou vstupního signálu $\overline{\text{SMI}}$. Z režimu SMM se zpět do původního režimu přejde instrukcí RSM. Signál $\overline{\text{SMI}}$ může počítač generovat např. při zavření víka s obrazovkou.

Režim SMM je transparentní (neviditelný) pro aplikace i operační systém z těchto důvodů:

- Jedinou možností, jak SMM zapnout, je externí nemaskovatelné přerušení přivedené speciálním signálem.
- Procesor zahájí provádění instrukcí určených pro SMM ze separátního adresového prostoru a separátní paměti (tzv. SMRAM – System Management RAM).
- Při přepínání do SMM procesor ukládá obsah všech registrů do zvláštní části SMRAM.
- Všechna přerušení, která normálně operační systém či aplikace obsluhuje, jsou během SMM zakázána.
- Stav před přepnutím do režimu SMM se vrátí provedením instrukce RSM.

SMM je podobný reálném režimu. Nejsou v něm úrovně oprávnění, privilegované instrukce nebo mapování adres. V SMM lze provádět V/V operace a adresovat celou 4GB kapacitu fyzické operační paměti.

9.1 Přerušení SMI

Signál $\overline{\text{SMI}}$ se vzorkuje vždy po ukončení provádění instrukce. Při rozpoznání aktivní úrovně na tomto signálu procesor vyčká na dokončení všech zápisů, uloží obsah všech registrů do SMRAM a předá řízení obslužné rutině SMM v SMRAM.

Přerušení $\overline{\text{SMI}}$ má vyšší prioritu než ladicí přerušení a než libovolné externí přerušení. To znamená, že při současném příchodu více žádostí o přerušení, se vybere právě $\overline{\text{SMI}}$.

Je-li jednou procesor v režimu SMM, ignorují se další žádosti o SMM a NMI. Stejně tak se nuluje povolení přerušení Machine Check Enable v registru CR4. Žádosti o SMM a NMI se zapamatují a uplatní se po návratu instrukcí RSM.

Po zapnutí režimu SMM se nuluje bit IF v registru EFLAGS, čímž se zakazují všechna externí přerušení. To proto, že SMM pracuje v separátním adresovém prostoru, ve kterém nejsou dostupné vektory původní IDT. Mají-li se přerušení povolit, musí se definovat nová tabulka přerušovacích vektorů v SMRAM. Tabulka má stejný formát jako v reálném režimu (viz str. 45). Rovněž musíte nastavit IDTR, aby ukazoval na tuto tabulku (tj. na začátek separátního adresového prostoru), viz též poznámka na str. 45. Tabulku přerušovacích vektorů a IDTR musíme nastavit také tehdy, má-li se během SMM generovat některá z výjimek. Tyto zakázat nelze.

Při vstupu do SMM se také nuluje bit TF v příznakovém registru (ruší se krokovací režim) a nuluje se registr DR7 (ruší se všechna ladicí přerušení). Mají-li se uvnitř SMM používat ladicí nástroje procesoru Pentium, musí se rovněž definovat tabulka přerušovacích vektorů a nastavit obsah IDTR.

9.2 Počáteční stav SMM

Po rozpoznání aktivní úrovně na signálu $\overline{\text{SMI}}$ procesor uchová obsahy všech registrů a nastaví je na implicitní hodnoty uvedené na obr. 9.1.

Vyprázdnění vyrovnávacích pamětí (vnitřní i vnější, datové i instrukční) musí zajistit technické vybavení počítače signály k tomu určenými.

9.3 Spuštění SMM

Procesor předá řízení rutině SMM na adrese 3000:8000 (báze segmentu 30000h, offset 8000h). Hodnotu báze lze změnit.

Při startu SMM se nastavují podmínky podobné reálnému režimu procesoru. Jsou vynulovány bity PE a PG (je vypnut chráněný režim a vypnuto stránkování). Báze segmentů (vyjma CS) jsou vynulovány a limit je nastaven na 4 GB. Báze se získá

Registr	*Implicitní obsah*
všeobecné registry	nedefinovány
EFLAGS	00000002h
EIP	00008000h
selektor CS	3000h
selektor DS, ES, FS, GS, SS	0000h
limit CS, DS, ES, FS, GS, SS	FFFFFFFFh (4 GB)
atribut CS, DS, ES, FS, GS, SS	16bitový, rostoucí nahoru
CR0	PE,EM,TS,PG:=0; ostatní nezměněny
CR4	00000000h
DR6	nedefinován
DR7	00000400h
GDTR,LDTR,IDTR,TR	nedefinovány
Model Specific Registers	nezměněny

Obr. 9.1 Implicitní obsah registrů po přepnutí do SMM

tak, že 16bitový selektor se posune o 4 bity vlevo. Limit a atributy se naplněním segmentového registru v jeho neviditelné části nemění.

Implicitní velikost operandu a adres je 16 bitů. Explicitním uvedením prefixu změny velikosti operandu či adresy se velikost mění na 32 bitů. Atributem změny velikosti operandu lze docílit skoků, volání podprogramu a návratů na libovolnou adresu v rámci 4GB adresového prostoru. Existují však tato omezení:

- Všechny instrukce pro větvení programu, které nemají prefix změny velikosti operandu, zkracují EIP na dolních 16 bitů.
- Žádná instrukce, přerušení nebo výjimka nemůže předat řízení do segmentu majícího bázi větší jak 1 MB. Toto omezení vyplývá z principu vyčíslování adres v reálném režimu na 20 bitech.
- Přerušení nebo výjimka nemohou předat řízení na offset větší jak 64 KB (16bitový offset).
- Výjimky a přerušení ukládají do zásobníku jako návratovou adresu pouze její dolní 16bitovou část. Je-li offset přerušené rutiny větší než 64 KB, není možné se bez dodatečných programových zásahů vrátit na přerušené místo.

9.4 Formát uložení registrů v SMRAM

Při přepnutí do SMM se do SMRAM ukládá obsah všech registrů procesoru. Registry se ukládají na offsety uvedené na obr. 9.2 do segmentu začínajícího bázovou adresou vyčíslenou z obsahu CS (ta je z implicitně nastavené hodnoty vyčíslena na 30000h). Do oblastí, které jsou označeny jako vyhrazené, nesmí ovladač režimu SMM nic zapisovat. Zápis do těchto oblastí může způsobit chybné chování procesoru. To proto, že vyhrazené oblasti obsahují neviditelné části registrů. Ve vyšších typech procesorů mohou vyhrazené oblasti obsahovat jiné údaje.

V některých případech může ovladač SMM potřebovat modifikovat obrazy uložených registrů. Ne obrazy všech registrů uvedených na obr. 9.2 lze libovolně měnit. Možnost nakládání s obrazy registrů je uvedena v tabulce na obr. 9.3. Stav „Uložen?“ označuje, zdali se registr ukládá a obnovuje při přepínání SMM. Stav „Čitelný?“ sděluje, zda je registr uložen do nevyhrazené (ano), či vyhrazené (ne) části. Stav „Zapisovatelný?“ označuje, zda lze obraz registru měnit. Poslední řádek tabulky popisuje registry, které přepnutím SMM se neukládají. Pokud je ovladač mění, musí je uložit sám.

9.5 Identifikátor verze SMM (offset FEFC)

Identifikátor verze SMM popisuje verzi SMM a dostupná rozšíření následovně:

Bity	*Obsah*
0-15	verze SMM
16	procesor podporuje opakování V/V instrukce
17	procesor podporuje přemísťování SMRAM
18-31	vyhrazeny

9.6 Opakování V/V instrukce (offset FF00)

Příznak opakování V/V instrukce (I/O Trap Restart) může nastavit ovladač SMM na hodnotu 0FFh tehdy, má-li procesor po provedení instrukce návratu z SMM (RSM) znovu provést přerušenou V/V instrukci. Pokud příznak obsahuje hodnotu 00h, procesor instrukci neopakuje. Příznak je vždy při zapnutí SMM nastaven procesorem na 00h a ovladač má právo jej nastavit na 0FFh jenom tehdy, byla-li přerušena právě V/V instrukce. Nastaví-li jej při přerušení jiné instrukce, bude další činnost procesoru nedefinovaná.

Offset	*Registr*
FFFC	CR0
FFF8	CR3
FFF4	EFLAGS
FFF0	EIP
FFEC	EDI
FFE8	ESI
FFE4	EBP
FFE0	ESP
FFDC	EBX
FFD8	EDX
FFD4	ECX
FFD0	EAX
FFCC	DR6
FFC8	DR7
FFC4	TR*
FFC0	LDTR*
FFBC	GS*
FFB8	FS*
FFB4	DS*
FFB0	SS*
FFAC	CS*
FFA8	ES*
FFA7-FF04	Vyhrazeno
FF02	Opakování instrukce HLT
FF00	Opakování V/V instrukce
FEFC	Identifikátor verze SMM
FEF8	Báze záznamu o registrech
FEF7-FE00	Vyhrazeno

* Horní dvě slabiky jsou vyhrazeny.

Obr. 9.2 Formát uložení registrů po přepnutí do SMM

Uložený registr	*Uložen?*	*Čitelný?*	*Zapisovatelný?*
EDI, ESI, EBP, ESP, EBX, EDX, ECS, EAX, EFLAGS, EIP	ano	ano	ano
CR0, CR3, DR6, DR7, TR, LDTR, GS, FS, DS, SS, CS, ES	ano	ano	ne
CR1, CR2, CR4, neviditelné části GDT, LDT, IDT, CS, DS, ES, FS, GS	ano	ne	ne
DR0-7, všechny FP registry	ne	ne	ne

Obr. 9.3 Možnost nakládání s obrazy uložených registrů SMM

9.7 Opakování instrukce HLT (offset FF02)

Je-li před příchodem aktivní úrovně signálu $\overline{\text{SMI}}$ procesor zastaven instrukcí HLT, nastaví se indikátor opakování instrukce HLT (Halt Auto Restart) na 1. V ostatních případech je indikátor vynulován. Pokud je indikátor nastaven na 1, může ovladač jeho hodnotu změnit podle potřeby. Možné kombinace jsou tyto:

Indikátor na vstupu	*Indikátor na výstupu*	*Akce procesoru na výstupu*
0	0	Pokračování další instrukcí přerušeného procesu
0	1	Nedefinovaná činnost
1	0	Pokračování instrukcí za HLT
1	1	Zopakování instrukce HLT

9.8 Báze záznamu o registrech (offset FEF8h)

V procesoru Pentium je neviditelný registr, ve kterém je uložena báze záznamu o registrech (State Dump Base) a adresa první instrukce ovladače SMM. Po přepnutí do SMM se obsah tohoto registru ukládá na offset FEF8h. Ovladač SMM může bázi záznamu o uložených registrech změnit (na offsetu FEF8h). Při návratu z SMM instrukcí RSM se báze z paměti přepíše zpět do registru. Další přepnutí do SMM už bude první instrukci a bázi záznamu o registrech mít uloženu na nové adrese.

Implicitně je neviditelný registr nastaven na adresu 30000h. Takový je i obsah offsetu FEF8h při prvním přepnutí do SMM. Nově nastavená hodnota musí být dělitelná 32 K. Jestli procesor přemísťování SMRAM povoluje, je oznamováno hodnotou bitu 17 v identifikátoru verze SMM.

9.9 Návrat z SMM

Pro návrat z režimu SMM se musí použít instrukce RSM. Instrukce se nesmí použít jinde než v SMM. Takový pokus vyvolá výjimku chybného operačního znaku. Během provádění instrukce se obnovuje stav všech registrů procesoru ze záznamu o registrech a kontroluje se následující:

- hodnota uložená jako báze záznamu o registrech musí být dělitelná 32 K,
- vyhrazené bity registru CR4 nesmějí být nastaveny na 1,
- v CR0 nesmí být chybná kombinace bitů: (PG=1 a PE=0) nebo (NW=1 a CD=0).

Při rozpoznání chyby se procesor zastaví. Z toho stavu jej vyvede nemaskovatelné přerušení (NMI) nebo signál RESET.

Poznámky k předchozím procesorům

Režim SMM je zaveden od procesoru Pentium.

10. Režim virtuální 8086

Režim *virtuální 8086* (dále též **V86**) umožňuje v rámci *chráněného režimu* spouštět programy určené pro procesory 8086 a 8088 a pro *reálné režimy* ostatních procesorů Intel, aniž by bylo nutné modifikovat jejich kód. Režim V86 se zapíná pro konkrétní úlohu a nevylučuje se tím možnost víceúlohového zpracování. Z pohledu uživatele umožňuje režim *virtuální 8086* provozovat jeden nebo více virtuálních procesorů 8086 uvnitř jednoho. Virtuální 8086 proces má vlastní TSS, je mu přiřazen první 1 MB lineárního adresového prostoru, přístup procesu k V/V branám je kontrolován mapou přístupných V/V bran atd.

10.1 Struktura procesu V86

Proces V86 uvnitř operačního systému je tvořen dvěma podprocesy: vlastním V86 procesem a řídicím procesem, běžícím v chráněném režimu. Řídicí proces se aktivuje výjimkami a přerušeními, které vznikly činností V86 procesu, příp. směřují zvnějšku do V86 procesu.

10.2 Zapnutí a vypnutí režimu V86

Režim V86 je zapnut tehdy, je-li nastaven příznak **VM** v příznakovém registru. Procesor pro manipulaci s příznakem VM neposkytuje žádné instrukce. Programátor jej může nastavit ze zásobníku nebo naplněním z TSS virtuálního 8086 procesu:

- Registr příznaků (včetně bitu VM) naplní ze zásobníku instrukce IRET. Má-li být VM nastaven na jedničku, musí být IRET provedeno z procesu běžícího na úrovni CPL=0, jinak se VM nezmění. V zásobníku musí být uloženy obsahy registrů CS:EIP podle konvencí 8086 (segment:offset) a musí jít o 32bitovou instrukci návratu z přerušení, protože příznak VM je v horním slově registru.

  Poznamenejme, že instrukce POPF nemění hodnotu příznaku VM.

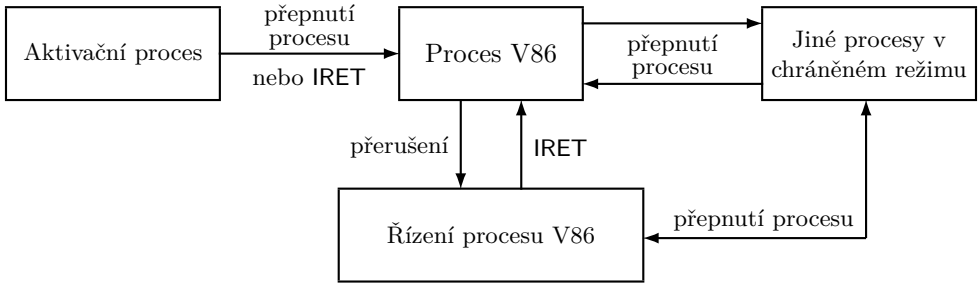
- Registr příznaků se plní z TSS přepnutím procesu. V TSS budoucího V86 procesu musí být nastaven obsah CS:EIP podle konvencí 8086. Aby se naplnil

i příznak VM, který je v horním slově, musí mít nový proces TSS 32bitového typu, nikoli typu Intel286 (ten obsahuje pouze 16bitový registr příznaků).

Procesor vypíná režim V86 v okamžiku přerušení:

- Procesor nuluje příznak VM po uložení registru EFLAGS do zásobníku při přerušení, které aktivovalo obslužnou rutinu s CPL=0. Obslužná rutina potom probíhá normálně v chráněném režimu. Pokud by měl být přerušením aktivován instrukční segment s C=1 nebo rutina s CPL jiné úrovně než 0, generuje se chybové přerušení a vrací se chybový kód obsahující selektor segmentu, do něhož mělo být předáno řízení.
- Přerušení vyvolalo přepnutí procesu z režimu V86 do jiného procesu. Příznak VM je vynulován tehdy, má-li nový proces 32bitový TSS s VM=0 nebo TSS typu Intel286.

Možné způsoby předávání řízení v operačním systému povolujícím V86 úlohy jsou na obr. 10.1.



Obr. 10.1 Způsoby předávání a vracení řízení z režimu V86

10.3 Ochrany v režimu V86

V procesoru 8086 ochrany nejsou implementovány. Libovolně lze přecházet z jednoho segmentu do druhého, přistupovat ke všem částem paměti, ke všem V/V branám atd. Poněvadž v chráněném režimu může být v paměti umístěno více procesů, musí být od sebe izolovány. Tedy je nutno také izolovat proces virtuální 8086 od procesů ostatních. Dovnitř procesu, podle zvyklostí 8086, prostředky ochrany nezasahují, hlídají pouze ty akce, které by mohly ovlivnit ostatní procesy.

Fakt, že proces pracuje v režimu virtuální 8086, sděluje, že má přidělenu **úroveň oprávnění CPL=3**. Z toho plyne, že v procesu virtuální 8086 nelze používat

ty privilegované instrukce, které lze provádět pouze na úrovni oprávnění CPL=0 (většinu z nich stejně procesor 8086 nezná).

Provádění V/V instrukcí (IN, OUT, INS, OUTS) je řízeno pouze mapou přístupných V/V bran v TSS procesu. Před provedením V/V instrukce procesor zkontroluje, zda je v mapě hodnota bitu odpovídající V/V bráně nulová. Je-li jedničková, potom se V/V instrukce neprovede a generuje se výjimka 13. V režimu V86 se u těchto instrukcí nebere ohled na IOPL. Hodnota IOPL v registru příznaků řídí provádění instrukcí:

CLI, STI
PUSHF, POPF
LOCK
INT n
IRET

Základní pravidlo pro použití výše uvedených instrukcí v režimu V86 je následující: je-li IOPL<3, je při pokusu o provedení kterékoli z výše uvedených instrukcí generována výjimka „Obecná chyba ochrany“, je-li úroveň IOPL=3, lze tyto instrukce v V86 procesu používat bez omezení (ve V86 je IOPL běžně =3).

Výše uvedené instrukce lze „hlídat“ z toho důvodu, že při používání programů, které byly z procesoru 8086 zvyklé vlastnit celý stroj, by se např. zakázáním přerušení mohly odstavit ostatní procesy v paměti procesoru, protože plánovač operačního systému, který procesům přiděluje řízení po časových kvantech, nemůže pracovat. Po zachycení těchto instrukcí výjimkou 13 lze zjistit, zda např. instrukce POPF modifikuje příznak IF. Pokud ne, instrukci může obslužná rutina nechat provést; pokud ano, rozhodne co dál.

Zachycování instrukce INT n má význam také proto, že touto instrukcí jsou volány služby operačního systému, které se mohou v režimu V86 lišit. Poznamenejme, že instrukce INTO a INT 3 (ladicí bod) nejsou tímto přerušením zachyceny proto, že ve vyšších typech procesorů mají stále stejnou funkci.

10.4 Výjimky a přerušení v režimu V86

V okamžiku přerušení procesu V86 není předáno řízení podle zvyklostí 8086 na tabulku přerušovacích vektorů, uloženou od adresy 0000:0000, ale přepne se do chráněného režimu a provede se obsluha podle příslušného popisovače v IDT. Je-li popisovačem některá z bran pro přerušení, musí ukazovat na proces s CPL=0. (Popisovačem může být též brána zpřístupňující TSS, o tom se však zmíníme později.) Zpět do režimu V86 se řízení vrací až instrukcí IRET.

V okamžiku přepnutí do chráněného režimu (vyvolaného přerušením) se podle momentálního obsahu TR (ukazuje na TSS procesu V86) přečte z TSS selektor a offset zásobníku určeného pro úroveň oprávnění 0.

Do tohoto zásobníku se ukládají obsahy datových segmentových registrů (GS, FS, DS a ES) a potom se tyto registry plní nulami (neplatným selektorem). Je to proto, že přepnutím procesu došlo ke změně mechanismu adresování a po přepnutí do chráněného režimu by tyto segmentové registry obsahovaly nedefinované hodnoty. Rutina, obsluhující přerušení V86 procesu, musí na začátku podle svých potřeb nastavit platné obsahy těchto segmentových registrů. Poněvadž se přepnutím procesů změnila úroveň oprávnění, uloží se do zásobníku také ukazatele SS a ESP předchozí úrovně. Potom se uloží EFLAGS, CS a EIP, platné v okamžiku přerušení ve tvaru podle konvencí 8086. Pokud se tímto přerušením předává chybové slovo, je také uloženo do zásobníku.

Přesná posloupnost ukládání registrů do zásobníku úrovně 0 (jeho obsah po přerušení V86 procesu) je na obr. 10.2.

Adresa	31	0	
n+32	??	GS	
n+28	??	FS	
n+24	??	DS	
n+20	??	ES	
n+16	??	SS	
n+12	ESP		
n+8	EFLAGS		
n+4	??	CS	
n	EIP		← SS:ESP (žádné chybové slovo)
n-2	Chybové slovo (volitelně)		← SS:ESP (s chybovým slovem)

Obr. 10.2 Obsah zásobníku úrovně 0 po přerušení procesu V86

Rutina, obsluhující přerušení, musí vědět, zda byla aktivována přerušením V86 procesu. První možnost, jak si tento fakt ověřit, je test všech datových segmentových registrů na nulový obsah (je pravděpodobné, i když velmi málo, že by i v jiném případě mohly být registry nulové). Druhá, jistější metoda je testování hodnoty příznaku VM uloženého do zásobníku úrovně 0 v rámci registru EFLAGS. Je-li VM v zásobníku jedničkový, byl přerušen proces V86, v opačném případě byl přerušen jiný proces v chráněném režimu.

Tento bit kontroluje také procesor v okamžiku výskytu instrukce IRET. Je-li při obnovování obsahu registru EFLAGS ze zásobníku úrovně 0 tento bit nastaven, musí procesor z toho zásobníku vyzvednout rovněž obsahy datových segmentových registrů.

Přerušení procesu V86 je obsluhováno branou pro maskující nebo nemaskující přerušení s těmito podmínkami: brána musí být typu Intel386 (nikoli Intel286), DPL brány musí být 3, DPL segmentu s obslužnou rutinou musí být 0 (protože pouze IRET na úrovni 0 smí změnit hodnotu příznaku VM).

V IDT může být také popisovač brány zpřístupňující TSS. Potom při přerušení dojde k přepnutí procesu. Proces obsluhující přerušení může být klasický proces chráněného režimu 32bitového procesoru, proces chráněného režimu Intel286 nebo jiný V86 proces. Při tomto přerušení se stav přerušeného procesu neukládá do zásobníku, protože byl přepnutím uložen do původního TSS. Pouze pokud přerušení generovalo chybové slovo, bylo toto uloženo na vrchol zásobníku nového procesu. Pokud je novým procesem opět V86 proces, je slovo uloženo na adresu SS:SP.

Při použití brány zpřístupňující TSS musí tato mít DPL=3. Vlastní proces může potom mít už libovolnou úroveň oprávnění (případný obslužný V86 proces má CPL=3).

10.5 Použití V86 procesu pro obsluhu přerušení

Je-li přerušením aktivován popisovač IDT, který obsahuje bránu zpřístupňující TSS, dojde k přepnutí procesu. Přepnutím procesu, které bylo vyvoláno přerušením, je nastaven příznak NT=1 a do nového TSS je uložen zpětný ukazatel na TSS původního procesu proto, aby instrukce IRET mohla přepnout zpět na původní proces. Na místě obslužného procesu může být i jiný V86 proces.

Problém nastane v okamžiku návratu z obslužného V86 procesu do původního, protože při vykonávání instrukce IRET procesor testuje bit NT pouze v chráněném režimu (nikoli v V86). Z tohoto důvodu je nereálné používat V86 procesy jako obslužné procesy přerušení.

10.6 Použití brány pro přerušení

Reálné je používat obslužné rutiny z toho V86 procesu, který byl přerušen (např. je-li V86 proces MS-DOS, je žádoucí používat na jeho přerušení také jeho obsluhu). Každé přerušení však musí nejprve projít jedním z popisovačů IDT, a tím přejít do chráněného režimu. Odtud můžeme procesem v chráněném režimu, který V86 proces řídí, vrátit obsluhu zpět do přerušeného procesu v těchto krocích:

1. Přerušení procesu V86 je obslouženo branou pro přerušení (mající v popisovači DPL=3), předávající řízení obslužné rutině uložené v segmentu s CPL=0. Stav V86 procesu je uložen do zásobníku (viz obr. 10.2) na úrovni 0 podle obsahu TSS (jde stále o TSS procesu V86, protože k přepnutí procesu nedošlo).
2. Ze zásobníku úrovně 0 zkopírujeme IP, CS a FLAGS (pouze jejich 16bitové části) do zásobníku V86 procesu (SS:SP – úroveň 3) a odpovídajícím způsobem upravíme obsah SP. Tím jsme simulovali klasické 8086 přerušení.
3. Do zásobníku úrovně 0 uložíme CS:EIP ve tvaru 8086 jako ukazatel na místo do procesu V86, kde začíná obsluha přerušení. Dále uložíme 32 bitů registru EFLAGS s nastaveným bitem VM a nulovými bity IOPL.
4. Provedeme instrukci IRET. Tím se ukončí chráněný režim a řízení se předá do V86 procesu (NT nastaveno nebylo, protože po celou dobu nedošlo k přepnutí procesu). Instrukcí IRET byla v tomto případě aktivována obslužná rutina, nebyl jí proveden návrat z obslužné rutiny.
5. Probíhá obsluha přerušení uvnitř vlastního V86 procesu až do okamžiku výskytu instrukce IRET. Pokus o provedení instrukce IRET v procesu V86 s IOPL=0 způsobí výjimku „Obecná chyba ochrany“ 13. Za předpokladu, že obsluha tohoto přerušení používá stejný zásobník úrovně 0, jsou zde uloženy informace z bodu 1.
6. Obsluha „Obecné chyby ochrany“ předá řízení rutině, obsluhující přerušení V86 procesu (stejné jako v bodě 1).
7. Ze zásobníku V86 procesu (úroveň 3) odstraníme hodnoty uložené v kroku 2 a upravíme SP. Tím simulujeme provedení IRET v procesu V86.
8. Provedeme instrukci IRET (nyní na úrovni oprávnění 0), která ukončí chráněný režim, protože v zásobníku úrovně 0 je uložen EFLAGS s nastaveným VM=1 a CS:EIP (ve tvaru 8086) ukazující do V86 procesu. Tím je dokončena obsluha přerušení.
9. Pokračuje se v přerušeném V86 procesu.

Kroky 2 a 7 mohou být vynechány, pokud víme, že obslužná rutina ve V86 procesu tyto hodnoty v zásobníku nepotřebuje.

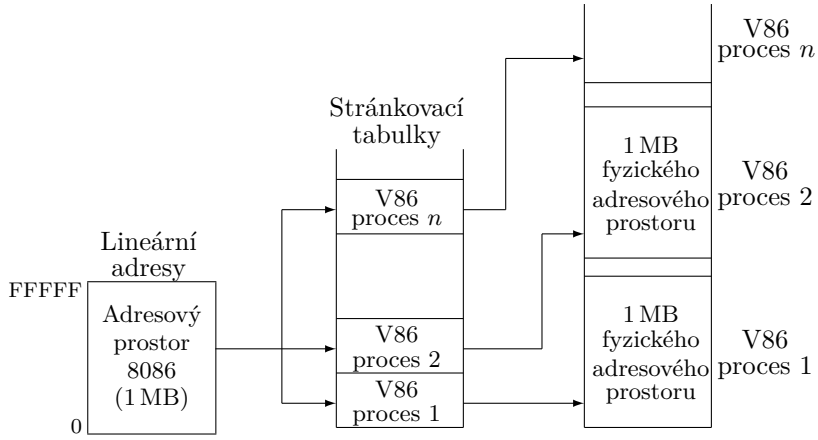
Pokud v kroku 3 nastavíme IOPL=3, není provedením instrukce IRET v kroku 5 vyvolána výjimka „Obecné chyby ochrany“, ale je uvnitř V86 procesu proveden návrat z přerušení. Potom se kroky 6 až 8 neprovedou. Tato varianta je jednodušší na implementaci, ale nastavením IOPL=3 povolujeme obsluze přerušení V86 procesu manipulovat s příznakem IF, což může být v rámci víceúlohového zpracování nebezpečné.

10.7 Stránkování v režimu V86

Pracuje-li proces v režimu V86, je vypnut mechanismus segmentování chráněného režimu, poněvadž segmentová část adresy je jinak interpretována. Přistupuje-li V86 proces k paměti, je k dispozici lineární adresa, vytvořená součtem složek (*segment*×16)+*offset* tak, jak to bylo uvedeno při popisu 8086 na obr. 4.1 na str. 43. Takto vytvořenou lineární adresu lze stránkováním, při zapnuté stránkovací jednotce (v registru CR0 je bit PG=1), transformovat na fyzickou adresu.

Není-li stránkovací jednotka zapnuta, jsou lineární a fyzické adresy totožné. V tom případě bude V86 proces adresovat první 1 MB fyzické operační paměti. Není-li stránkovací jednotka zapnuta a není-li stránkování obsluhováno, smí být spuštěn maximálně jeden V86 proces.

Je-li žádoucí provádět více V86 procesů zároveň, musí být stránkování zapnuto a prováděna výměna stránek tak, aby nedocházelo ke kolizím (např. aby dva procesy nezapisovaly do stejné stránky). Stránkovací jednotka potom stránky v paměti mapuje tak, jak je uvedeno na obr. 10.3. Faktu, že více procesů může přistupovat k jedné stránce, lze naopak využít ke sdílení části paměti (programy v pamětech ROM nebo reentrantní části procesů) těmito procesy.

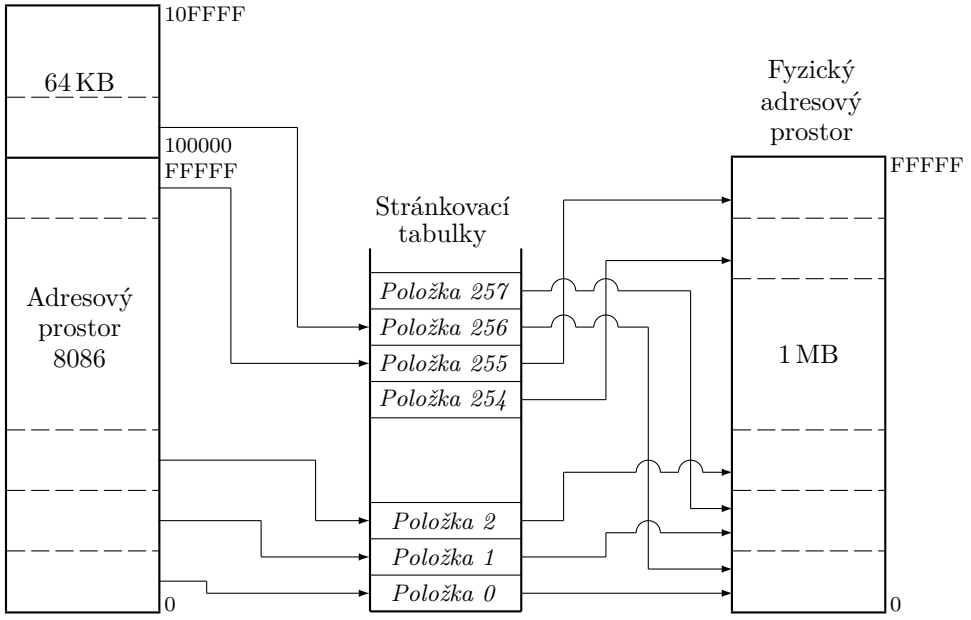


Obr. 10.3 Stránkování paměti při zpracovávání více V86 procesů

Na tomto místě je potřeba si uvědomit jeden důležitý rozdíl mezi procesem v 8086 a V86 procesem. Kapacita paměti 8086 je maximálně 1 MB, i když součtem (*segment*×16)+*offset* můžeme získat adresu větší než 1 MB (konkrétně 10FFEF). Omezení na 1 MB je dáno 20bitovou adresovou sběrnicí. Pokud by v 8086 byly k dispozici hodnoty segmentu a offsetu, které by po sečtení daly hodnotu větší než 1 MB,

výsledná adresa by ukazovala na začátek paměti, protože nejvyšší řád je ignorován (konkrétně maximální hodnota by byla 0FFEF, tj. o něco méně než 64 KB).

Poněvadž mají procesory Intel386 a vyšší 32bitovou adresovou logiku, neprovádí se v režimu V86 omezování lineární adresy na 1 MB. Tím lze ve V86 režimu adresovat paměť kapacity téměř 1 088 KB (tj. 1 MB+64 KB). Triku s přeplněním adresové logiky procesoru 8086 bylo využito v některých programech pro přechod od nejvyšších adres k nejnižším. Pokud by V86 režim tento trik neuměl, nebylo by možné některé programy zde spouštět. Zařídit to lze tak, že pomocí mechanismu stránkování namapujeme přebývajících 64 KB na začátek paměti (viz obr. 10.4).



Obr. 10.4 Mapování stránek 256 až 271 do stránek 0 až 15 v režimu V86

Poněvadž ve V86 režimu není v činnosti segmentování a s tím spojené ochrany, lze použít pouze stránkové ochrany. Ochran se využije např. pro označení stránek „pouze ke čtení“, takové stránky mohou simulovat paměť ROM. Poněvadž V86 proces pracuje vždy jenom na úrovni CPL=3, jsou těmto procesům dostupné jenom ty stránky, které jsou označeny U=1.

10.8 Rozdíly V86 oproti 8086

Z rysů procesorů od Intel386 vyplývají některé vlastnosti, které nelze v režimu V86 ošetřit tak, aby chování bylo naprosto totožné s 8086. Navíc v 8086 bylo několik chyb, které ve vyšších typech procesorů jsou již odstraněny. Některé rozdíly vyplývají jenom z toho faktu, že jde o 32bitové procesory. Rozdíly nejsou zásadního charakteru, a proto zpravidla nezpůsobují problémy.

Je opravena výjimka 0. V Intel386 a vyšších se v přerušení, generovaném přeplněním při dělení, ukládá do zásobníku adresa ukazující na instrukci způsobující přerušení (DIV a IDIV). V 8086 se ukládá adresa ukazující za instrukci, která přerušení způsobila.

Je rozšířen rozsah IDIV. Zvětšením velikosti slova od Intel386 je také zvětšena maximální hodnota podílu, a proto vyšší typy procesorů nemusejí vždy generovat výjimku 0 tak, jak ji generoval 8086.

Adresa 1 MB není maximální. V 8086 omezovala maximální hodnotu adresy 20bitová adresová sběrnice. V režimu V86 může být maximální hodnota adresy 10FFEF (viz str. 146).

Větší prostor pro uložení offsetu. Proces v 8086 nemůže generovat offset větší než FFFF, protože má pouze 16bitové registry. V86 proces může použít instrukční prefix pro změnu velikosti operandu a potom produkovat offset velikosti až 4 GB. Tento stav procesory od Intel386 v režimu V86 hlídají a pokud je offset větší než 64 KB, generuje přerušení. Je-li takový offset spojen se segmentovým registrem DS, ES, FS nebo GS, je vyvolána výjimka 13, ve spojení se SS je vyvolána výjimka 12.

Je změněna instrukce PUSH SP. Instrukce od procesoru Intel386 ukládá do zásobníku obsah SP před provedením instrukce. Tím se liší od 8086, který ukládá hodnotu SP sníženou o 2 (tj. novou hodnotu SP).

Prefix LOCK se nesmí používat kdykoli. Procesory od Intel386 generují výjimku 6 (Chybný operační znak) tehdy, je-li prefix použit s jinou instrukcí než s jednou z následujících:

BT, BTS, BTR, BTC,
XCHG, XADD, CMPXCHG, CMPXCHG8B,
ADD, ADC, SUB, SBB,
AND, OR, XOR, NOT, NEG,
INC, DEC.

Je rozšířen registr příznaků. Od Intel386 mají korektní hodnoty i ty bity příznakového registru, o které je příznakový registr rozšířen.

Je opraveno chybové přerušení koprocesoru. Když matematický koprocesor 8087 přerušením oznamoval chybu, ukládal procesor do zásobníku adresu chy-

bující instrukce. Od Intel386 je do zásobníku ukládána adresa prvního z prefixů chybující instrukce.

Je navíc generována výjimka 16. Po detekování signálu $\overline{\text{ERROR}}$ v době čekání na dokončení operace koprocesoru je generována výjimka 16.

Je omezen počet posuvů a rotací. Od Intel386 je maskován počet rotací a posuvů jednoho objektu tak, že je ponecháno pouze nejnižších 5 bitů (tj. maximálně 32 rotací nebo posuvů). Omezení je zavedeno proto, že instrukce rotující 255× trvala poměrně dlouho a byla nepřerušitelná.

Druhé NMI se uplatní až po dokončení obsluhy prvního. Procesory od Intel386 po dobu obsluhy NMI blokují všechna přerušení včetně dalších NMI až do provedení instrukce IRET.

Je opraveno přerušení řetězcových operací. Je-li v procesorech od Intel386 přerušena opakovaně prováděná (REP) řetězcová instrukce, je do zásobníku uložena návratová adresa ukazující na první prefix přerušené instrukce. V 8086 nebyly do návratové adresy zahrnuty prefixy.

Je stanoven limit délky instrukce. Jedna instrukce nesmí v Intel386 být delší než 15 slabik. Je-li při dekódování instrukce rozpoznáno překročení tohoto limitu (instrukce má nadbytečné prefixy), generuje se výjimka „Obecná chyba ochrany“.

Nedefinované operační znaky 8086 mohou být instrukce Intel386 nebo vyšších. Poněvadž se rozšířil počet instrukcí a většinu z nich lze v režimu V86 používat, mohou mít operační kódy, které nebyly v 8086 definovány, přiřazenu smysluplnou instrukci.

Vyšší typy procesorů jsou rychlejší. Procesor je rychlejší nejen tím, že pracuje na vyšší frekvenci, ale také tím, že některé instrukce trvají méně taktů. Rychlost pracujícího programu může být oproti 8086 vyšší 5 až 25× (podle typu počítače). To je všeobecně přijímáno jako hlavní výhoda vyšších typů. Na škodu může být u těch programů přenesených z 8086, které odpočítávaly čas počtem provedených instrukcí.

Poznámky k předchozím procesorům

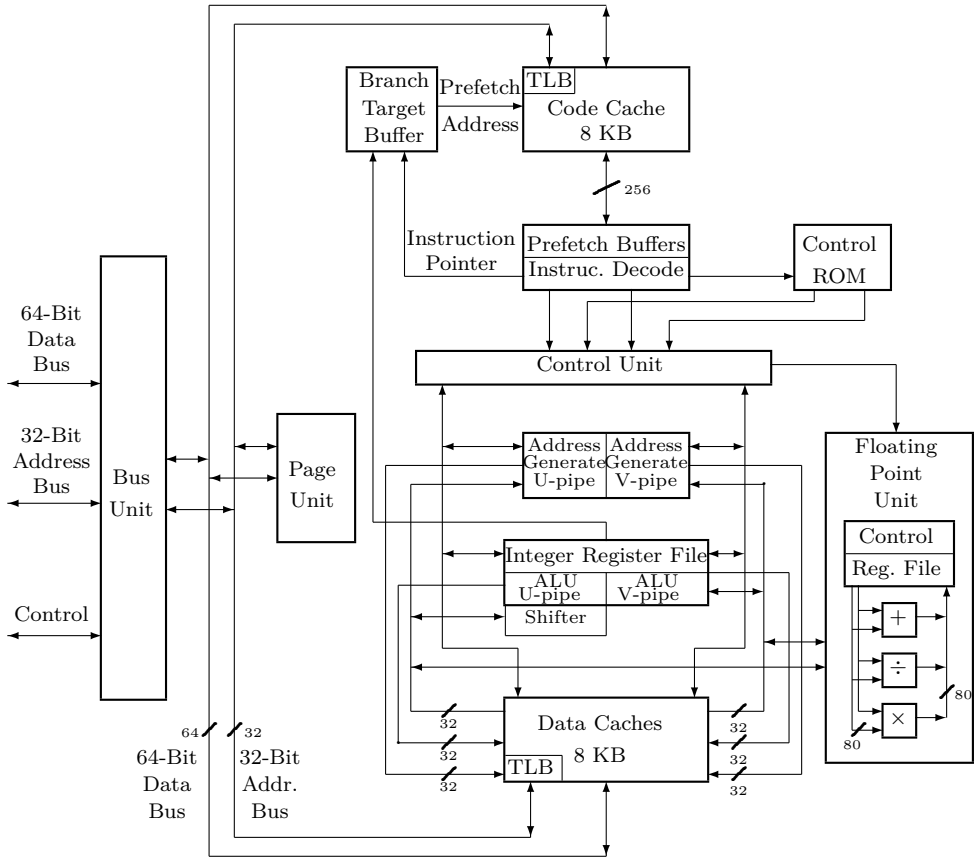
Režim V86 je zaveden od procesoru Intel386.

11. Další rysy architektury procesoru Pentium

V procesoru Pentium jsou integrovány všechny vlastnosti procesoru Intel486. Navíc poskytuje tato významná rozšíření:

- superskalární architekturu,
- dynamické předvídání skoků,
- zřetězenou FPU,
- zkrácení doby provádění instrukcí,
- oddělené 8KB datové a instrukční vnitřní vyrovnávací paměti,
- protokol MESI pro řízení datové vyrovnávací paměti,
- 64bitovou datovou sběrnici,
- zřetězování cyklů sběrnice,
- adresové parity,
- vnitřní kontrolu parity,
- kontrolu správné funkce znásobením čipů s procesorem,
- sledování provádění,
- monitorování výkonnosti,
- ladění prostřednictvím IEEE 1149.1 Boundary Scan,
- režim správy systému a
- rozšíření v režimu V86.

Instrukční repertoár Pentia je plně kompatibilní s Intel486. Všechny programy určené pro procesory Intel386 a Intel486 lze na Pentiu spouštět bez omezení. Plně kompatibilní je také i správa paměti (MMU).



Obr. 11.1 Blokový diagram procesoru Pentium

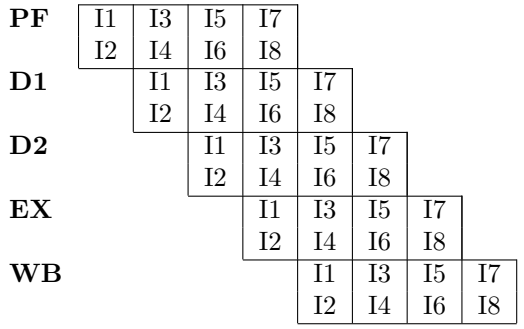
11.1 Zřetězené provádění instrukcí

Procesor Pentium má několik nových rysů vedoucích ke zvýšení výkonu procesoru. Operace lze samostatně a nezávisle provádět ve dvou instrukčních frontách pro zřetězené zpracování a v FPU. Každá fronta se zřetězeným zpracováním dokončí v každém hodinovém cyklu jednu běžnou instrukci. Dvě nezávislé fronty dokončí v jednom hodinovém cyklu až dvě instrukce a FPU jednu (ve výjimečných případech i dvě) FPU instrukce.

Většina instrukcí (např. instrukce pro celočíselnou aritmetiku) se provádí v 5 fázích. Tyto fáze jsou:

PF	Prefetch	Výběr instrukce
D1	Instruction Decode	Dekódování instrukce
D2	Address Generate	Generování adresy
EX	Execute	Provedení instrukce
WB	Write Back	Dokončení instrukce

Každá z těchto fází se provádí v samostatné jednotce. Jednotky jsou na sobě nezávislé, a proto lze provádět souběžně např. fázi PF instrukce 2 spolu s fází D1 instrukce 1. Procesor Pentium má navíc každou jednotku zdvojenou. Díky této architektuře lze v každém hodinovém cyklu dokončit až 2 instrukce. Řetězové zpracování procesoru Pentium je na obr. 11.2.



Obr. 11.2 Zřetězené zpracování v procesoru Pentium

Tyto dvě zřetězené fronty se nazývají „u“ a „v“. Proces souběžného zpracovávání instrukcí se nazývá „párování“. Ve zřetězené frontě „u“ lze provádět libovolnou instrukci, zatímco ve frontě „v“ lze provádět pouze jednoduché instrukce, popsané v pravidlech pro párování instrukcí. Jednoduše řečeno – párovat lze instrukce podobné RISCovým. Instrukce vložená do fronty „v“ je vždy instrukcí následující za instrukcí ve frontě „u“.

Ve fázi výběru instrukce (PF) se instrukce vybírá z vnitřní vyrovnávací paměti procesoru (Cache) nebo z operační paměti. Protože Pentium má oddělené vyrovnávací paměti pro instrukce a pro data, nedochází ke konfliktům výběru instrukce se zápisem dat. Pokud požadovaná instrukce není ve vyrovnávací paměti, vybírá se z paměti operační. Při výběru instrukce se používají dvě 32slabikové pomocné paměti. V jedné se provádí výběr sekvenčně (tj. instrukce následující) a v druhé výběr podle paměti adres skoků (viz dále). Výhodou tohoto mechanismu je, že procesor ještě před vyhodnocením podmínky pro provedení skoku má již vybranou následující instrukci. Ze dvou vybraných použije tu, na jejíž zpracování vyhodnocení podmínky vede.

Ve fázi dekódování instrukce (D1) se dva paralelně pracující dekodéry rozhodují, zdali lze instrukce provést v páru, nebo sekvenčně pouze ve frontě „u“. Pro vyhodnocení každého z prefixů instrukce je zapotřebí jednoho hodinového cyklu. Instrukce se dekóduje teprve po dekódování všech jejích prefixů.

Ve fázi D2 se vyčíslí adresa operandu v paměti. Procesor Pentium vyčíslí adresu skládající se z přímé hodnoty a přírůstku nebo z báze a indexu během jednoho cyklu (Intel486 toto řešil ve dvou cyklech).

Do fáze provedení instrukce (EX) je zahrnuto jak vlastní zpracování operandu umístěného v registru, tak i výběr dat z datové vnitřní vyrovnávací paměti a případný zápis dat zpět. Na výběr dat z vyrovnávací paměti je potřebný další hodinový cyklus. Závěrečná fáze WB slouží k dokončení instrukce a ke změně stavu procesoru vyvolaného provedením instrukce.

11.2 Předvídání skoků

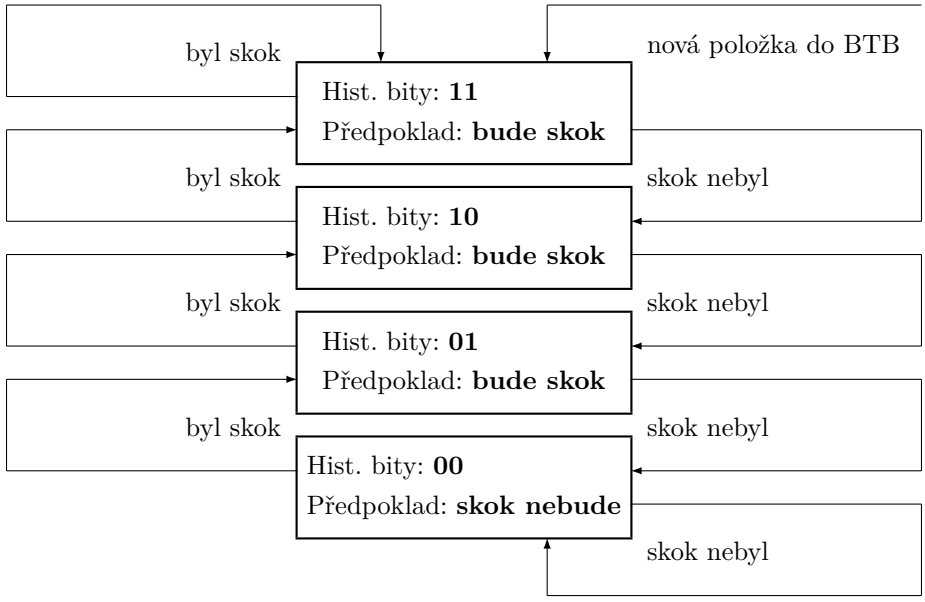
Procesor Pentium má v sobě implementován mechanismus předvídání výsledku podmíněných skokových instrukcí. Algoritmus je založen na následujícím faktu: vždy po provedení těla cyklu se podmíněnou skokovou instrukcí rozhoduje o jeho opětném provedení. S významně větší pravděpodobností nastává situace, že při zopakování téže podmíněné skokové instrukce bude její výsledek stejný (tj. tělo cyklu se bude opakovat vícekrát a jenom jednou se cyklus opustí).

Návrháři do procesoru zahrnuli **paměť adres skoků** (**Branch Target Buffer - BTB**). Do této paměti procesor ukládá cílové adresy ze skokových instrukcí (získaných fází D1), které provedly skok, spolu s 2bitovým záznamem historie skoků.

Při výběru instrukce se testuje obsah BTB na shodu s adresou vybírané instrukce. Pokud se adresa v BTB najde, zkoumá se obsah bitů historie. Předpokládá-li se skok, pokračuje se výběrem následujících instrukcí počínaje instrukcí, na kterou ukazuje operand skokové instrukce. Pokud se ve fázi provedení skokové instrukce zjistí, že

předpověď byla špatná, musí se fronta předvybraných instrukcí vyprázdnit a výběr se zahajuje znovu ze správné adresy.

Historické bity představují automat, jehož graf přechodů je na obr. 11.3. Vždy po provedení skoku se bity aktualizují. Pouze v kombinaci 00 se předpokládá, že skok nenastane.



Obr. 11.3 Graf přechodů historických bitů v BTB

BTB je 4cestná asociativní paměť se 64 řádky. Každá z 256 položek uchovává cílovou adresu a 2 historické bity. Při ukládání se nová položka do BTB uloží na náhodně vybrané místo.

11.3 Pravidla párování instrukcí

Procesor Pentium je zařízen na párování instrukcí a jejich souběžné provádění. Předem je vhodné sdělit: i když existuje mechanismus na paralelní provádění instrukcí, programátor na instrukce stále nahlíží tak, jako by se zpracovávaly pouze sekvenčně. Instrukce mohou být spojeny do páru za splnění následujících podmínek.

- Obě instrukce v páru musí být „jednoduché“ podle dále uvedené definice.

- Mezi instrukcemi v páru nesmí být vztah „čtení až po zápisu“ nebo „zápis až po čtení“.
- Žádná z instrukcí nesmí mít výpočet adresy složen ze dvou částí: z přímé hodnoty a zároveň z přírůstku.
- Instrukce s prefixy (vyjma 0F před podmíněným skokem) lze provádět pouze ve frontě „u“.

Jednoduché instrukce jsou ty, které nevyžadují mikrokód (jsou realizovány zapojením logických obvodů) a provedou se během jednoho hodinového cyklu. Výjimkou jsou instrukce aritmeticko-logické jednotky (ALU) *mem,reg* a *reg,mem*, které se provádějí ve dvou nebo třech taktech a jsou považovány za jednoduché. Za jednoduché se považují tyto instrukce určené pro celočíselné zpracování:

1. mov *reg, reg/mem/imm*
2. mov *mem, reg/imm*
3. alu *reg, reg/mem/imm*
4. alu *mem, reg/imm*
5. inc *reg/mem*
6. dec *reg/mem*
7. push *reg/mem*
8. pop *reg*
9. lea *reg,mem*
10. jmp/call/j*cond* near
11. nop

Podmíněné a nepodmíněné skoky smějí být párovány pouze jako druhé instrukce v páru. Nesmějí se párovat s následující instrukcí sekvenční v pořadí. Instrukce shift/rot o 1 a instrukce shift o *imm* smí být v páru pouze jako první.

Do páru se nesmějí zahrnout instrukce provázané přes libovolný registr či příznak v příznakovém registru se vztahem vyjádřeným implicitně, či explicitně. Např. instrukce ALU nastavující příznak ve frontě „u“ nesmí být párována s instrukcí ADC nebo SBB zpracovávající příznak ve frontě „v“. Výjimkou jsou dva případy: častá posloupnost instrukcí porovnání a skok se párovat smí, stejně tak posloupnost push-pop. Tyto dva případy jsou obslouženy speciálními, k tomu určenými obvody tak, aby je bylo možné provést paralelně.

Jsou-li v páru instrukce ALU s operandem v paměti, doplňují se 2 až 3 hodinové cykly pro zpracování jednotlivých instrukcí.

11.4 Vyrovnávání zápisu a serializační instrukce

V procesoru Pentium jsou spolu se dvěma zřetězenými frontami „u“ a „v“ implementovány také dvě zápisové paměti (Write Buffers) pro vyrovnávání zápisu do paměti ze souběžně zpracovávaných instrukcí. Každá z pamětí je 64bitová. Obě zápisové paměti se mohou plnit současně. Obsah zápisové paměti se potom posílá na sběrnici, aby se zápis dokončil až do paměti. Instrukce zapisující do paměti končí svoji činnost po zápisu požadovaného obsahu do zápisové paměti. Bez ohledu na to, zda již proběhl opis zápisové paměti po sběrnici do paměti, se zahajuje provádění další instrukce. Požaduje-li zahájená další instrukce ke svému provedení cyklus sběrnice, musí se počkat na dokončení zápisových cyklů z předchozí instrukce. Nepožaduje-li cyklus sběrnice, probíhá nezávisle na zápisových cyklech instrukce předchozí.

Výjimku tvoří tzv. **serializační instrukce**. Tyto dříve, než zahájí svoji činnost, vyčkají na dokončení modifikací příznaků, registrů, zápisů do paměti vyvolaných předcházejícími instrukcemi. Mezi serializační instrukce procesoru Pentium patří: MOV do speciálního registru vyjma CR0, INVD, INVLPG, IRET, IRETD, LGDT, LLDT, LIDT, LTR, WBINVD, CPUID, RSM a WRMSR.

Pro serializaci na libovolné úrovni oprávnění lze použít instrukci CPUID. Při čekání na dokončení všech zápisů vyvolaných předchozími instrukcemi se také čeká na aktivní úroveň signálu $\overline{\text{EWBE}}$, kterým vnější technické prostředky mohou signalizovat nedokončenost zápisů. Při serializaci se neprovádí přepis změněných položek interních vyrovnávací pamětí (Cache) do paměti. Je-li to nutné, musí programátor vyrovnávací paměti vyprázdnit instrukcí WBINVD.

11.5 Sledování toku instrukcí

Procesor Pentium má v sobě zabudovány prostředky na sledování toku instrukcí. Pomocí těchto prostředků lze sledovat využití „u“ a „v“ fronty zřetězeného zpracování a provádění skokových instrukcí.

11.5.1 Signály IU, IV a IBT

Využití zřetězených front a provádění skokové instrukce lze sledovat na speciálních signálech IU, IV a IBT. Signál IU se do aktivní úrovně nastavuje při dokončení instrukce v „u“ frontě. Signál IV totéž indikuje pro frontu „v“. Signál IBT indikuje dokončení skokové instrukce. Kombinace těchto signálů shrnuje tabulka na obr. 11.4.

IU	IV	IBT	Význam
0	0	0	Není dokončena žádná instrukce.
0	0	1	Kombinace nenastane.
0	1	0	Kombinace nenastane.
0	1	1	Kombinace nenastane.
1	0	0	Ve frontě „u“ se dokončila jiná instrukce než skoková.
1	0	1	Ve frontě „u“ se dokončila skoková instrukce.
1	1	0	V obou frontách byly dokončeny instrukce. Žádná nebyla skoková.
1	1	1	V obou frontách byly dokončeny instrukce. Instrukce ve frontě „v“ byla skoková.

Obr. 11.4 Interpretace signálů IU, IV a IBT

11.5.2 Registr TR12

Nastavením bitu 1 (bit TR) v registru TR12 na 1 se zapíná zapisování speciálního cyklu na sběrnici. Tento cyklus se nazývá „Branch Trace Message Special Cycle“ a oznamuje okolí procesoru cílovou adresu skokové instrukce. Do registru TR12 se zapisuje instrukcí WRMSR (viz str. 286).

Signály IU, IV a IBT se nastavují bez ohledu na obsah registru TR12. Cyklus se na sběrnici objeví s 0 nebo více taktovým zpožděním. Obsahy cyklů se před zápisem na sběrnici ukládají do pomocné paměti a asi po dvou následujících instrukcích skoku se teprve na sběrnici zapíší. Na sběrnici se pošle tato informace: bity A31-A3 obsahují bity 31-3 cílové lineární adresy skoku, BT2-BT0 obsahují bity 2-0 adresy skoku a BT3 obsahuje 1 při implicitní velikosti operandu 32 bitů a 0 při velikosti operandu 16 bitů.

Při inicializaci procesoru se bit TR v registru TR12 nuluje. Ostatní bity registru TR12 jsou popsány ve speciálních dodatcích k dokumentaci, vydávaných výrobcem. Jejich význam se může podle modelů procesoru měnit.

12. Instrukční repertoár

Následující obsažná kapitola předkládá čtenáři kompletní popis instrukčního repertoáru. Instrukce nejsou seřazeny abecedně, jak bývá v referenčních příručkách zvykem[1], ale jsou řazeny do skupin podle jejich funkce. Abecední odkazy na instrukce nechť čtenář hledá v rejstříku.

Dříve, než začneme popisovat jednotlivé instrukce, musíme se seznámit s pojmy, které se při popisu používají.

12.1 Atributy velikosti operandu a adresy

Při provádění instrukce může procesor používat 16 nebo 32bitové adresování operační paměti. Aby bylo procesoru jasné, jaké adresování má použít, je každá instrukce přistupující k operandu v paměti spojena s atributem, který typ adresování určí. Použití 16bitového adresování znamená použití 16bitového přírůstku a 16bitového vyčíslování offsetu (relativní adresy v segmentu). Na druhé straně 32bitové adresování implikuje použití 32bitového přírůstku a 32bitového vyčíslování offsetu. Podobným způsobem procesor pracuje i s operandy. Atribut velikosti operandu určuje, zdali se pracuje se slovy (16bitové) nebo dvojslovy (32bitové).

Výsledná hodnota atributu je kombinací dvou veličin: jeho implicitního nastavení, které se v chráněném režimu přebírá ze specifikačního bitu v popisovačích segmentů, určujícího atribut pro segment a z instrukčního prefixu změny atributu.

12.1.1 Implicitní atribut segmentu

Programům pracujícím v chráněném režimu je atribut přiřazen v popisovači segmentu následovně: v popisovači instrukčního segmentu je bit D, který určuje implicitní atribut velikosti adresy i operandu. Tento atribut se vztahuje na všechny instrukce prováděné v tomto segmentu. Nulová hodnota bitu D nastavuje implicitní hodnotu atributu velikosti adresy a operandu na 16 bitů a jedničková na 32 bitů.

[1] Autor nechce případného zájemce o studium instrukcí „otrávit“ hned na začátku popisem instrukce AAA, kterou s největší pravděpodobností nikdy nepoužije.

Programy pracující v reálném režimu a v režimu V86 mají tento atribut nastaven na 16 bitů.

12.1.2 Prefixy měnící hodnotu atributu

Vnitřní kódování instrukcí dovoluje používat dva jednoslabikové instrukční prefixy: prefix změny velikosti adresy (67h) a prefix změny velikosti operandu (66h). Oba tyto prefixy „přebijí“ implicitní nastavení pro jednu instrukci, která bezprostředně následuje. Možné kombinace implicitního nastavení, instrukčního prefixu a výsledné hodnoty atributu jsou uvedeny v tabulce na obr. 12.1.

D=	0	0	0	0	1	1	1	1
Prefix změny velik. operandu	ne	ne	ano	ano	ne	ne	ano	ano
Prefix změny velik. adresy	ne	ano	ne	ano	ne	ano	ne	ano
Velikost operandu v bitech	16	16	32	32	32	32	16	16
Velikost adresy v bitech	16	32	16	32	32	16	32	16

Obr. 12.1 Použití instrukčních prefixů 66h a 67h v závislosti na parametru D

12.1.3 Atribut velikosti zásobníkové adresy

Svůj atribut také mají instrukce, které implicitně používají zásobník (např. POP EAX). Instrukce, které mají atribut velikosti zásobníkové adresy nastaven na 16 bitů, používají 16bitový registr SP. Instrukce, mající atribut velikosti zásobníkové adresy nastaven na 32 bitů, používají 32bitový registr ESP.

Implicitní nastavení tohoto atributu je v bitu B popisovače datového segmentu se zásobníkem. Nulová hodnota bitu B sděluje, že se používá 16bitové adresování, a jedničková určuje 32bitové adresování.

12.2 Formát instrukcí

Všechny instrukce procesoru jsou kódovány podle základního scématu uvedeného na obr. 12.2. Instrukce jsou složeny z volitelných instrukčních prefixů (v libovolném pořadí), z jedné nebo dvou slabik primárního operačního znaku, případných specifikátorů adresy složených ze slabik ModR/M a SIB (Scale Index Base), případného přírůstku a případných přímých operandů (dat). Přímý operand – je-li uveden – je poslední položkou instrukce.

Prefixy jsou rozděleny do skupin. Každá skupina použije jednu nebo žádnou slabiku na uložení svého prefixu. Skupiny jsou tyto:

Název instrukčního pole	*Velikost ve slabikách*
Instrukční prefix	0 nebo 1
Atribut změny velikosti adresy	0 nebo 1
Atribut změny velikosti operandu	0 nebo 1
Změna implic. segment. registru	0 nebo 1
Operační znak	1 nebo 2
ModR/M	0 nebo 1
SIB	0 nebo 1
Přírůstek	0, 1, 2 nebo 4
Přímý operand	0, 1, 2 nebo 4

Obr. 12.2 Základní formát instrukcí

1. Instrukční prefixy: REP, REPE, REPZ, REPNE, REPNZ, LOCK
2. Prefixy měnící implicitní segment: CS, SS, DS, ES, FS, GS
3. Prefixy měnící velikost operandu
4. Prefixy měnící velikost adresy

Z každé skupiny se smí uvést nejvýše jeden prefix. Použije-li se před jednou instrukcí více prefixů z jedné skupiny, nelze předem určit činnost procesoru. Prefixy mohou být zadány v libovolném pořadí. Operační znaky prefixů jsou následující:

F3h	REP (lze použít pouze s řetězcovou instrukcí)
F3h	REPE/REPZ (lze použít pouze s řetězcovou instrukcí)
F2h	REPNE/REPNZ (lze použít pouze s řetězcovou instrukcí)
F0h	LOCK
2Eh	změna segmentu na CS
36h	změna segmentu na SS
3Eh	změna segmentu na DS
26h	změna segmentu na ES
64h	změna segmentu na FS
65h	změna segmentu na GS
66h	změna velikost operandu
67h	změna velikost adresy

12.2.1 Slabiky ModR/M a SIB

Slabiky ModR/M a SIB (Scale Index Base) následují za operačním znakem a nesou tyto informace:

- způsob indexace nebo číslo v instrukci použitého registru,

- číslo použitého registru nebo další informace upřesňující činnost instrukce,
- informace o bázi a indexu.

Slabika ModR/M obsahuje tři pole informací:

- Pole **mod** ležící ve dvou bitech nejvyššího řádu v kombinaci s polem r/m vytváří prostor pro 32 kombinací: 8 pro určení registru a 24 pro indexové módy.
- Pole **reg** ležící v dalších třech bitech určuje buď číslo registru, nebo poskytuje prostor na rozšíření operačního znaku o další 3 bity. Význam pole reg je vymezen první slabikou operačního znaku.
- Pole **r/m** ležící ve třech bitech nejnižšího řádu určuje registr jako operand nebo spolu s polem mod definuje použitý typ indexace.

Přítomnost slabiky SIB v instrukci určuje obsah slabiky ModR/M. SIB obsahuje tato pole:

- Pole **ss** obsahuje měřítko.
- V poli **index** je číslo registru s indexem.
- Pole **base** obsahuje číslo registru s bází.

Slabika ModR/M

7 6	5 4 3	2 1 0
mod	**reg/opcode**	**r/m**

Slabika SIB (Scale Index Base)

7 6	5 4 3	2 1 0
ss	**index**	**base**

Obr. 12.3 Formát slabik ModR/M a SIB

Formát slabik ModR/M a SIB shrnuje obr. 12.3. Hodnoty a jim odpovídající význam bitů slabik ModR/M a SIB jsou uvedeny v tabulkách na obrázcích A.1, A.2, A.3. Na obr. A.1 je obsah slabiky ModR/M pro 16bitový adresový režim, na obr. A.2 je totéž pro 32bitový adresový režim a na obr. A.3 je formát SIB určený pro 32bitovou adresaci. Obrázky začínají na str. 361.

12.3 Formát popisu instrukcí

V následujících odstavcích si uveďme informace, které musí čtenář vědět dříve, než začne studovat popisy jednotlivých instrukcí.

12.3.1 Popis operandů instrukce

V popisu operandů se vyskytují tyto symboly:

- **rel8:** relativní 8bitová adresa v rozsahu 128 slabik před koncem instrukce až 127 slabik za koncem instrukce.
- **rel16, rel32:** relativní adresa v rámci segmentu, ve kterém je použita instrukce. **rel16** se použije u instrukcí s 16bitovým atributem velikosti operandu, **rel32** se použije u instrukcí s 32bitovým atributem velikosti operandu.
- **ptr16:16, ptr16:32:** vzdálený ukazatel použitý zpravidla v instrukčním segmentu různém od adresovaného. Zápis **16:16** znamená, že hodnota ukazatele je složena ze dvou částí. Hodnota nalevo od dvojtečky představuje 16bitový selektor nebo hodnotu, která se má uložit do registru CS. Hodnota napravo od dvojtečky představuje offset uvnitř cílového segmentu. **ptr16:16** je použito při 16bitovém atributu velikosti operandu, **ptr16:32** při 32bitovém atributu.
- **r8:** jeden ze slabikových registrů: AL, CL, DL, BL, AH, CH, DH nebo BH.
- **r16:** jeden ze slovních registrů: AX, CX, DX, BX, SP, BP, SI nebo DI.
- **r32:** jeden ze dvojslovních registrů: EAX, ECX, EDX, EBX, ESP, EBP, ESI nebo EDI.
- **eAX:** zápisem sdělujeme, že na tomto místě se použije buď registr AX (v 16bitovém kontextu), nebo registr EAX (v 32bitovém kontextu).
- **imm8:** přímá slabiková hodnota uložená v instrukci jako operand. **imm8** je číslo se znaménkem v intervalu −128 až +127 včetně. V instrukcích, ve kterých se hodnota **imm8** kombinuje se slovním nebo dvojslovním operandem, se přímá 8bitová hodnota znaménkově rozšiřuje na 16 nebo 32 bitů (znaménkový bit dolní slabiky se kopíruje do všech vyšších bitů).
- **imm16:** přímá slovní hodnota použitá v instrukci s atributem velikosti operandu 16 bitů. Hodnota je v intervalu −32 768 až +32 767.
- **imm32:** přímá dvojslovní hodnota použitá v instrukci s atributem velikosti operandu 32 bitů. Hodnota je v intervalu −2 147 483 648 až +2 147 483 647.
- **r/m8:** jeden ze slabikových registrů (viz **r8**) nebo slabika v paměti. Adresa paměti se získává běžným výpočtem efektivní adresy.
- **r/m16:** jeden ze slovních registrů (viz **r16**) nebo slovo v paměti použité v instrukci s 16bitovým atributem velikosti operandu. Adresa paměti se získává běžným výpočtem efektivní adresy.
- **r/m32:** jeden ze dvojslovních registrů (viz **r32**) nebo dvojslovo v paměti použité v instrukci s 32bitovým atributem velikosti operandu. Adresa paměti se získává běžným výpočtem efektivní adresy.

- **r/m64:** čtyřslovní registr nebo operand v paměti použitý v instrukci s 64bitovým atributem velikosti operandu. Adresa paměti se získává běžným výpočtem efektivní adresy.
- **m:** 16 nebo 32bitový paměťový operand.
- **m8:** slabika v paměti adresovaná DS:eSI nebo ES:eDI (použito pouze v řetězcových instrukcích).
- **m16:** slovo v paměti adresované DS:eSI nebo ES:eDI (použito pouze v řetězcových instrukcích).
- **m32:** dvojslovo v paměti adresované DS:eSI nebo ES:eDI (použito pouze v řetězcových instrukcích).
- **m16:16, m16:32:** paměťový operand obsahující vzdálený ukazatel složený ze dvou částí. Hodnota nalevo od dvojtečky představuje 16bitový selektor nebo hodnotu, která se má uložit do registru CS. Hodnota napravo od dvojtečky představuje offset uvnitř cílového segmentu.
- **m16&32, m16&16, m32&32:** paměťový operand obsahující dvojici údajů, jejichž velikost v bitech je popsána čísly nalevo a napravo od znaku &. Jsou povoleny všechny adresovací módy. Operandy **m16&16** a **m32&32** používá instrukce BOUND pro uložení hranic intervalu testovaného čísla. Operand **m16&32** používají instrukce LIDT a LGDT pro uložení limitu a báze tabulek.
- **moffs8, moffs16, moffs32:** jednoduchá paměťová proměnná typu slabika, slovo nebo dvojslovo, používaná některými variantami instrukce MOV. Aktuální adresu představuje offset od báze segmentu. Tato instrukce nemá slabiku ModR/M. Mezi **moffs8**, **moffs16** a **moffs32** se vybírá podle atributu velikosti adresy.
- **Sreg:** segmentový registr. Segmentové registry mají v kódování instrukcí tato čísla: ES=0, CS=1, SS=2, DS=3, FS=4, GS=5.
- **m32real, m64real, m80real:** paměťové operandy pro FPU v pohyblivé řádové čárce: single, double nebo extended real.
- **m16int, m32int, m64int:** paměťové operandy pro FPU celočíselné: word, short a long integer.
- **mNbyte:** paměťový operand pro FPU délky N slabik.
- **ST** nebo **ST(0):** prvek na vrcholu zásobníku FPU.
- **ST(i):** i-tý prvek od vrcholu zásobníku FPU (i=0..7).

12.3.2 Popis operací vykonávaných instrukcí

Sekce „Operace“ popisuje v algoritmické podobě přesnou činnost instrukce. Postup je popsán v nepříliš detailně specifikovaném vyšším programovacím jazyce. Ze specifikace uveďme:

- Poznámky jsou uzavřeny do závorek „`(*`“ a „`*)`“.
- Složené příkazy jsou uzavřeny mezi příkazové závorky: příkaz **if** má tvar IF, THEN, ELSE, FI; příkaz **do** má příkazové závorky DO, OD; příkaz **case** je ve tvaru CASE ... OF, ESAC.
- Provádění je ukončeno vyčerpáním příkazů nebo příkazem END.
- Jméno registru implikuje obsah registru. Jméno registru uzavřené do hranatých závorek sděluje, že zpracovávaný obsah není v registru, ale je na adrese uvedené v registru. Např. ES:[DI] označuje obsah paměti na segmentové adrese ES a relativní offsetové adresy uložené v DI. Zápis [SI] značí obsah paměti na relativní adrese uložené v registru SI a v segmentu určeném implicitním segmentovým registrem (což je pro SI registr DS).
- Hranaté závorky se také používají spolu s paměťovými operandy a označují, že obsahem tohoto operandu je offset relativní k začátku segmentu.
- `A := B;` znamená, že obsah `B` je uložen do `A`.
- Symboly `=`, `<>`, `>`, `<`, `>=`, `<=` jsou relační operátory použité k porovnání dvou hodnot a znamenají rovno, nerovno, větší, menší, větší-nebo-rovno, menší-nebo-rovno. Operandy spolu s operátory tvoří zápis podmínky.

V jazyku jsou použity následující identifikátory:

- `velikost operandu` reprezentuje hodnotu atributu velikosti operandu instrukce, který je buď 16 nebo 32 bitů. `velikost adresy` reprezentuje hodnotu atributu velikosti adresy instrukce, která je rovněž buď 16 nebo 32 bitů.
  Např.:

  ```
  IF instrukce = CMPSW
  THEN velikost operandu := 16;
  ELSE
    IF instrukce = CMPSD
    THEN velikost operandu := 32;
    FI;
  FI;
  ```

  V příkladu se hodnota atributu velikosti operandu nastavuje podle zadané instrukce. Je-li instrukce CMPSW, nastaví se atribut velikosti operandu na 16 bitů. Je-li zadána instrukce CMPSD, nastaví se atribut velikosti operandu na 32 bitů.

- **eSP** reprezentuje buď registr SP, nebo registr ESP v závislosti na hodnotě bitu B v popisovači zásobníkového segmentu.

V jazyku jsou použity následující funkce:

- **zkraťˇ_na_16_bitů(hodnota)** redukuje velikost hodnoty na 16 bitů uřezáním bitů vyšších řádů.
- **neznaménkové_rozšíření(hodnota)** vrací hodnotu rozšířenou na požadovaný počet bitů doplněním binárních nul do přidávaných bitů. Např. slabikovou hodnotu −10 ve tvaru 0F6h neznaménkově rozšíří na potřebných 32 bitů do tvaru 000000F6h. Má-li funkce rozšiřovat na stejný počet bitů, jako má vstupní hodnota, vrací se nezměněná vstupní hodnota.
- **znaménkové_rozšíření(hodnota)** vrací hodnotu rozšířenou na požadovaný počet bitů zkopírováním znaménkového bitu vstupní hodnoty do všech přidávaných bitů. Např. slabikovou hodnotu −10 ve tvaru 0F6h znaménkově rozšíří na potřebných 32 bitů do tvaru 0FFFFFFF6h. Má-li funkce rozšiřovat na stejný počet bitů, jako má vstupní hodnota, vrací se nezměněná vstupní hodnota.
- **16bitová_adresa(operand), 32bitová_adresa(operand)** vrací 16 nebo 32bitovou efektivní adresu operandu v rámci daného segmentu.
- **Push(hodnota)** ukládá hodnotu do zásobníku. Počet do zásobníku ukládaných bitů je určen atributem velikosti operandu. Činnost funkce Push je popsána algoritmem u instrukce PUSH.
- **Pop()** vybírá hodnotu ze zásobníku. Počet ze zásobníku vybíraných bitů je určen velikostí operandu, kterému se vybraná hodnota přiřazuje, např. `EAX := Pop()` vybírá 32bitovou položku. Činnost funkce Pop je popsána algoritmem u instrukce POP.
- **Bit[BitBase,BitOffset]** vrací hodnotu bitu z bitového řetězce. Bitový řetězec může být v registru nebo v paměti. Bity jsou očíslovány podle svých binárních řádů: bit nejnižšího řádu má číslo 0. Hodnota **BitBase** představuje registr nebo adresu v paměti. Hodnota **BitOffset** je číslo bitu.

  Je-li řetězec umístěn v registru, může BitOffset nabývat hodnot 0 až 31. Zápis Bit[EAX,21] znamená 22. bit v registru EAX.

  Je-li BitBase adresa paměti, může se hodnota BitOffsetu pohybovat v intervalu −2 Gbity až 2 Gbity. Adresa slabiky je potom vyčíslena vztahem (BitBase + (BitOffset DIV 8)) a číslo bitu ve slabice vztahem (BitOffset MOD 8).
- **Povolení_V/V (V/V_adresa, šířka_registru)** testuje hodnotu bitu nebo bitů v mapě přístupných V/V bran v tabulce TSS. Tuto funkci definujme následovně:

```
IF TSS je 16bitové THEN RETURN Nepovoleno; FI;
Ptr := [TSS + 102]; (* připrav ukazatel do bitové mapy *)
BitStringAddr := SHR(V/V_adresa, 3) + Ptr;
MaskShift := V/V_adresa AND 7;
CASE šířka_registru OF
  slabika: nBitMask := 1;
  slovo: nBitMask := 3;
  dvojslovo: nBitMask := 0Fh;
ESAC;
mask := SHL(nBitMask,MaskShift);
CheckString := [BitStringAddr] AND mask;
IF CheckString = 0
THEN RETURN Povoleno;
ELSE RETURN Nepovoleno;
FI;
```

- **PŘEPNUTÍ PROCESŮ** je podrobně popsáno v kapitole 5.6.

12.3.3 Nastavované příznaky

Nastavované příznaky jsou uvedeny v tabulce u jména instrukce. Jednotlivá políčka mají následující význam:

O	OF (Overflow Flag)
D	DF (Direction Flag)
I	IF (Interrupt Flag)
T	TF (Trap Flag)
S	SF (Sign Flag)
Z	ZF (Zero Flag)
A	AF (Auxiliary CF)
P	PF (Parity Flag)
C	CF (Carry Flag)

?	hodnota příznaku nedefinována
*	nastavena podle výsledku
0	hodnota příznaku vždy nula
1	hodnota příznaku vždy jedna
	hodnota příznaku nezměněna

U některých instrukcí je ještě příznakům věnována zvláštní odrážka „Nastavované příznaky“. U instrukcí FPU jsou navíc uvedeny speciální příznaky FPU.

12.3.4 Výjimky

Symbol výjimky je uveden znakem # následovaným dvěma znaky identifikace výjimky a volitelným chybovým kódem v závorce. Např. #GP(0) znamená výjimku obecného porušení ochrany s chybovým kódem 0. V tabulce na obr. 12.4 jsou symbolům výjimek přiřazeny významy a čísla přerušení.

Zkratka	*Přerušení*	*Význam výjimky*
#UD	6	Chybný operační znak
#NM	7	Zařízení nepřístupné
#DF	8	Dvojnásobný výpadek
#TS	10	Chybný TSS
#NP	11	Segment nebo brána nepřístupna
#SS	12	Výpadek stránky zásobníku
#GP	13	Obecné porušení ochrany
#PF	14	Výpadek stránky
#MF	16	FP chyba
#AC	17	Kontrola zarovnání

Obr. 12.4 Symboly výjimek, významy a čísla přerušení

Jednotlivé výjimky chráněného režimu a s nimi spojené stavy procesoru jsou podrobně popsány v kapitole 5.7. Aplikační programátoři ještě navíc musejí znát reakci jejich operačního systému na tu kterou výjimku.

12.3.5 Poznámky k předchozím procesorům

Na závěr popisu instrukce mohou být uvedeny poznámky týkající se existence či odlišností instrukce v předchozích typech procesorů (tj. Intel486, Intel386, Intel286 a 8086).

Není-li uvedeno nic, jde o instrukci, která je v repertoáru od procesoru 8086. V textu není uveden původ instrukcí vztahujících se k chráněnému režimu, ty byly zavedeny v procesoru Intel286. Stejně tak nejsou uvedeny poznámky ke 32bitovým operandům a 32bitovém zpracování, to spolu s V86 režimem platí od procesoru Intel386.

12.4 Instrukce pro přesun dat

Přehled instrukcí zahájíme zřejmě nejpoužívanější instrukcí MOV.

12.4.1 Instrukce přesunů dat pro obecné použití

MOV

Move Data

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce MOV přenáší obsah zdrojového operandu (tj. druhého operandu) do cílového operandu. Přitom obsah zdrojového operandu zůstává beze změny.

Instrukce	Popis
MOV *r/m8,r8*	Přesun obsahu *r8* do *r8* nebo *m8*
MOV *r/m16,r16*	Přesun obsahu *r16* do *r16* nebo *m16*
MOV *r/m32,r32*	Přesun obsahu *r32* do *r32* nebo *m32*
MOV *r8,r/m8*	Přesun obsahu *r8* nebo *m8* do *r8*
MOV *r16,r/m16*	Přesun obsahu *r16* nebo *m16* do *r16*
MOV *r32,r/m32*	Přesun obsahu *r32* nebo *m32* do *r32*
MOV *r/m16,Sreg*	Přesun obsahu segmentového registru do slova *r/m*
MOV *Sreg,r/m16*	Přesun slova *r/m* do segmentového registru
MOV AL,*moffs8*	Přesun slabiky z adresy *seg:offs* do AL (bez ModR/M)
MOV AX,*moffs16*	Přesun slova z adresy *seg:offs* do AX (bez ModR/M)
MOV EAX,*moffs32*	Přesun dvojslova z adresy *seg:offs* do EAX (bez ModR/M)
MOV *moffs8*,AL	Přesun slabiky z AL na adresu *seg:offs* (bez ModR/M)
MOV *moffs16*,AX	Přesun slova z AX na adresu *seg:offs* (bez ModR/M)
MOV *moffs32*,EAX	Přesun dvojslova z EAX na adresu *seg:offs* (bez ModR/M)
MOV *r/m8,imm8*	Plnění *r/m8* slabikovou konstantou *imm8*
MOV *r/m16,imm16*	Plnění *r/m16* slovní konstantou *imm16*
MOV *r/m32,imm32*	Plnění *r/m32* dvojslovní konstantou *imm32*

Operace

```
první_operand := druhý_operand;
```

Komentář

Je-li na místě cílového operandu segmentový registr (DS, ES, SS apod.), potom se v chráněném režimu plní i jeho neviditelná část obsahem odpovídajícího popisovače. Popisovač se vybírá podle selektoru, kterým je plněn segmentový registr. Při plnění segmentového registru se kontrolují přístupová práva procesu k segmentu. Pokud proces k segmentu jakýmkoli způsobem nemá přístup, generuje se výjimka. Zvláštní postavení má *neplatný selektor* (hodnoty 0000-0003), kterými lze segmentový registr naplnit vždy. V tomto případě se výjimka generuje až při použití segmentového registru – nikoli při jeho plnění, jak je tomu ve všech jiných případech.

Při plnění segmentového registru SS se zakazuje přerušení až do dokončení instrukce následující po této. Předpokládá se, že následující instrukcí je instrukce plnění registru eSP. Procesor tak zajišťuje nedělitelnost této kritické operace. Plnění segmentového registru v chráněném režimu popíšeme následovně:

```
IF se plní SS
THEN
  selektor je platný, jinak #GP(0);
  index selektoru je uvnitř limitu tabulky popisovačů, jinak #GP(selektor);
  RPL ze selektoru se rovná CPL, jinak #GP(selektor);
  slabika AR indikuje zapisovatelný datový segment, jinak #GP(selektor);
  DPL ve slabice AR se rovná CPL, jinak #GP(selektor);
  segment ve slabice AR je označen jako přítomný, jinak #SS(selektor);
  naplň selektor SS;
  naplň neviditelnou část SS;
FI;
IF se plní DS, ES, FS nebo GS platným selektorem
THEN
  index selektoru je uvnitř limitu tabulky popisovačů, jinak #GP(selektor);
  slabika AR indikuje datový nebo čitelný instrukční segment,
                                                    jinak #GP(selektor);
  IF slabika AR indikuje datový nebo nepřizpůsobitelný instrukční segment
  THEN RPL a CPL musí být menší nebo rovné DPL ve slabice AR;
  ELSE #GP(selektor);
  FI;
  segment ve slabice AR je označen jako přítomný, jinak #NP(selektor);
  naplň selektor segmentového registru;
  naplň neviditelnou část segmentového registru;
FI;
IF se plní DS, ES, FS nebo GS neplatným selektorem
THEN
  naplň selektor segmentového registru;
  vynuluj bit platnosti neviditelné části segmentového registru;
FI;
```

Výjimky ve chráněném režimu

Při plnění segmentového registru se může generovat #GP, #SS a #NP. Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu, #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

XCHG *Exchange Memory/Register with Register*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce XCHG vzájemně zamění obsahy prvního a druhého operandu. Výměna je možná mezi dvěma registry nebo mezi registrem a pamětí.

Instrukce	Popis
XCHG *r/m8,r8*	záměna slabik
XCHG *r8,r/m8*	záměna slabik
XCHG *r/m16,r16*	záměna slov
XCHG *r16,r/m16*	záměna slov
XCHG *r/m32,r32*	záměna dvojslov
XCHG *r32,r/m32*	záměna dvojslov

Operace

```
Pomocná := první_operand;
první_operand := druhý_operand;
druhý_operand := Pomocná;
```

Komentář

Je-li jeden operand v paměti, je po dobu trvání instrukce blokována sběrnice signálem $\overline{\text{LOCK}}$ bez ohledu na uvedení nebo neuvedení instrukčního prefixu LOCK a bez ohledu na hodnotu IOPL. Instrukce XCHG lze použít na BSWAP pro 16bitová data.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu, #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

12.4.2 Instrukce pro manipulaci se zásobníkem

PUSH — *Push Operand onto the Stack*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce PUSH uloží hodnotu operandu do zásobníku.

Instrukce	Popis
PUSH *r/m16*	Uložení slova do zásobníku
PUSH *r/m32*	Uložení dvojslova do zásobníku
PUSH *imm8*	Uložení slabiky do zásobníku
PUSH *imm16*	Uložení slova do zásobníku
PUSH *imm32*	Uložení dvojslova do zásobníku
PUSH *Sreg*	Uložení seg. registru do zásobníku

Operace

```
IF segment uchovávající zásobník je 16bitový
THEN
  IF velikost operandu = 16
  THEN
    SP := SP - 2;
    (SS:SP) := operand;  (* 16bitové přiřazení *)
  ELSE (* velikost operandu = 32 *)
    SP := SP - 4;
    (SS:SP) := operand;  (* 32bitové přiřazení *)
  FI;
ELSE (* segment uchovávající zásobník je 32bitový *)
  IF velikost operandu = 16
  THEN
    ESP := ESP - 2;
    (SS:ESP) := operand;  (* 16bitové přiřazení *)
  ELSE
    ESP := ESP - 4;
    (SS:ESP) := operand;  (* 32bitové přiřazení *)
  FI;
FI;
```

Komentář

Instrukce provede operaci ve dvou krocích: nejprve sníží obsah registru ukazatele vrcholu zásobníku (SP v 16bitovém segmentu, ESP v 32bitovém segmentu) o 2 (pro 16bitový operand) nebo o 4 (pro 32bitový operand) a potom na nový vrchol zásobníku (SS:SP, SS:ESP) uloží obsah operandu.

Instrukce PUSH SP a PUSH ESP ukládají do zásobníku originální hodnotu SP nebo ESP (tj. hodnotu před provedením instrukce, nesníženou o 2 nebo 4).

Instrukce PUSH *m16/32*, která používá registr ukazatele vrcholu zásobníku (SP, ESP) jako bázový, si hodnotu báze vyčíslí před provedením instrukce a tu použije.

Instrukce PUSH používající operand v paměti trvá déle než posloupnost dvou instrukcí, která přesune operand do registru a potom uloží registr do zásobníku.

Výjimky ve chráněném režimu

Pokud nová hodnota SP nebo ESP ukazuje mimo rozsah segmentu se zásobníkem, generuje se #SS(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Žádné. Pokud je SP nebo ESP rovno 1, procesor se zastaví.

Výjimky v režimu V86

#PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Procesor 8086 v instrukci PUSH SP ukládal do zásobníku novou hodnotu SP (sníženou o 2). Provede-li se v 8086 posloupnost instrukcí PUSH SP a POP SP, je po provedení posloupnosti hodnota SP o 2 nižší než před vykonáním těchto instrukcí. Varianty s přímým operandem jsou k dispozici od procesoru Intel286.

POP — *Pop Operand from the Stack*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce POP vybere hodnotu ze zásobníku a uloží ji do operandu instrukce.

Instrukce	Popis
POP *r/m16*	Výběr slova ze zásobníku
POP *r/m32*	Výběr dvojslova ze zásobníku
POP *Sreg*	Výběr slova ze zásobníku a jeho uložení do seg. registru (vyjma CS)

Operace

```
IF segment uchovávající zásobník je 16bitový
THEN
  IF velikost operandu = 16
  THEN
    operand := (SS:SP);  (* 16bitové přiřazení *)
    SP := SP + 2;
  ELSE (* velikost operandu = 32 *)
    operand := (SS:SP);  (* 32bitové přiřazení *)
    SP := SP + 4;
  FI;
ELSE (* segment uchovávající zásobník je 32bitový *)
  IF velikost operandu = 16
  THEN
```

```
    operand := (SS:ESP);  (* 16bitové přiřazení *)
    ESP := ESP + 2;
  ELSE
    operand := (SS:ESP);  (* 32bitové přiřazení *)
    ESP := ESP + 4;
  FI;
FI;
```

Komentář

Instrukce POP provede dva základní kroky: nejprve z vrcholu zásobníku (SS:SP v 16bitovém segmentu nebo SS:ESP v 32bitovém segmentu) zkopíruje slovo nebo dvojslovo (podle velikosti operandu) do operandu a potom zvýší ukazatel vrcholu zásobníku (SP nebo ESP) o 2 nebo 4 (opět podle velikosti operandu).

Instrukce POP CS **není** součástí instrukčního repertoáru procesoru. Na tuto činnost je určena instrukce RET.

Je-li operandem segmentový registr (DS, ES, FS, GS nebo SS), musí být v chráněném režimu plněn pouze selektorem. Potom plnění segmentového registru automaticky vyvolává i plnění jeho neviditelné části vč. kontroly přístupových práv.

Neplatným selektorem se mohou plnit registry DS, ES, FS a GS. V tomto případě se výjimka #GP(0) generuje až při použití obsahu takto naplněného segmentového registru.

Instrukce POP SS zakazuje všechna přerušení (vč. NMI) až do skončení následující instrukce. Předpokládá se, že bude následovat instrukce MOV SP,BP. Tím procesor umožní naplnit registry SS:SP, aniž by v případě přerušení posloupnosti byl k dispozici chybný ukazatel do zásobníku. Vhodnější však je plnit registry SS, SP (ESP) instrukcí LSS.

Instrukce POP *m16/32*, která používá registr ukazatele vrcholu zásobníku (SP, ESP) jako bázový, si hodnotu báze vyčíslí až po naplnění registru SP nebo ESP.

Instrukce POP SP nebo POP ESP nejprve zvýší obsah SP nebo ESP a teprve potom obsah vrcholu zásobníku vloží do SP nebo ESP.

V chráněném režimu proběhnou při plnění segmentového registru kontroly podle tohoto algoritmu:

```
IF je plněn SS
THEN
  IF selektor je neplatný THEN #GP(0);
  index selektoru ukazuje dovnitř limitu tabulky popisovačů,
        jinak #GP(selektor);
  RPL selektoru = CPL, jinak #GP(selektor);
  slabika AR ukazuje zapisovatelný datový segment, jinak #GP(selektor);
  DPL ve slabice AR = CPL, jinak #GP(selektor);
```

```
  segment je označen jako přítomný, jinak #SS(selektor);
  naplň SS selektorem;
  naplň neviditelnou část SS popisovačem;
FI;

IF je plněn DS, ES, FS nebo GS
THEN
  IF selektor je neplatný
  THEN
    naplň segmentový registr selektorem;
    vynuluj bit platnosti v neviditelné části segm. registru;
  ELSE (* selektor je platný *)
    slabika AR ukazuje datový nebo čitelný instrukční segment,
         jinak #GP(selektor);
    IF segment je datový nebo instrukční nepřizpůsobitelný
    THEN
      RPL a současně CPL <= DPL ve slabice AR, jinak #GP(selektor);
    FI;
    segment je označen jako přítomný, jinak #NP(selektor);
    naplň segmentový registr selektorem;
    naplň neviditelnou část segmentového registru popisovačem;
  FI;
FI;
```

Výjimky ve chráněném režimu

Při plnění segmentového registru se může generovat #GP, #SS a #NP. Ukazuje-li vrchol zásobníku mimo meze zásobníkového segmentu, generuje se #SS(0). Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

PUSHA
PUSHAD

Push all General Registers
Push all General Extended Registers

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce PUSHA ukládá na vrchol zásobníku registry AX, CX, DX, BX, SP, BP, SI a DI. Viz též instrukce PUSH. Instrukce PUSHAD ukládá registry EAX, ECX, EDX, EBX, ESP, EBP, ESI a EDI.

Instrukce	Popis
PUSHA	Ulož AX, CX, DX, BX, SP, BP, SI a DI do zásobníku
PUSHAD	Ulož EAX, ECX, EDX, EBX, ESP, EBP, ESI a EDI do zásobníku

Operace

```
IF nastavená velikost operandu = 16 (* instrukce PUSHA *)
THEN
  Pomocná := SP;
  Push(AX);
  Push(CX);
  Push(DX);
  Push(BX);
  Push(Pomocná);
  Push(BP);
  Push(SI);
  Push(DI);
ELSE  (* velikost operandu = 32, instrukce PUSHAD *)
  Pomocná := ESP;
  Push(EAX);
  Push(ECX);
  Push(EDX);
  Push(EBX);
  Push(Pomocná);
  Push(EBP);
  Push(ESI);
  Push(EDI);
FI;
```

Komentář

Instrukce PUSHA sníží ukazatel vrcholu zásobníku celkově o 16 a instrukce PUSHAD o 32.

Výjimky ve chráněném režimu

Pokud hodnota SP nebo ESP ukazuje mimo rozsah segmentu se zásobníkem, generuje se #SS(0). #PF(výpadek) se generuje při výpadku stránky.

Výjimky v reálném režimu

Pokud je před provedením PUSHA SP rovno 1, 3 nebo 5, procesor se zastaví. Pokud je před provedením PUSHAD ESP rovno 1, 3, ...,15, procesor generuje výjimku 13.

Výjimky v režimu V86

Stejné jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky.

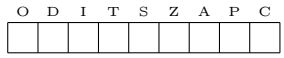
Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel286.

POPA
POPAD

Pop all General Registers

Pop all General Extended Registers



Popis:

Instrukce POPA vybírá ze zásobníku obsah registrů DI, SI, BP, BX, DX, CX a AX. Viz též instrukce POP. Instrukce POPAD vybírá ze zásobníku obsah registrů EDI, ESI, EBP, EBX, EDX, ECX a EAX.

Instrukce	Popis
POPA	Vyber DI, SI, BP, BX, DX, CX a AX ze zásobníku
POPAD	Vyber EDI, ESI, EBP, EBX, EDX, ECX a EAX ze zásobníku

Operace

```
IF nastavená velikost operandu = 16 (* instrukce POPA *)
THEN
  DI := Pop();
  SI := Pop();
  BP := Pop();
  SP := SP + 2; (* přeskoč jedno slovo v zásobníku *)
  BX := Pop();
  DX := Pop();
  CX := Pop();
  AX := Pop();
```

```
ELSE  (* velikost operandu = 32, instrukce PUSHAD *)
  EDI := Pop();
  ESI := Pop();
  EBP := Pop();
  SP := SP + 4; (* přeskoč jedno dvojslovo v zásobníku *)
  EBX := Pop();
  EDX := Pop();
  ECX := Pop();
  EAX := Pop();
FI;
```

Komentář

Instrukce POPA vybírá ze zásobníku informaci uloženou dříve instrukcí PUSHA a instrukce POPAD vybírá informaci uloženou PUSHAD. Obě instrukce ignorují uloženou hodnotu registru ukazatele zásobníku. Instrukce PUSHA zvýší ukazatel vrcholu zásobníku celkově o 16 a instrukce PUSHAD o 32.

Výjimky ve chráněném režimu

Pokud hodnota SP nebo ESP ukazuje mimo rozsah segmentu se zásobníkem, generuje se #SS(0). #PF(výpadek) se generuje při výpadku stránky.

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

#PF(výpadek) se generuje při výpadku stránky.

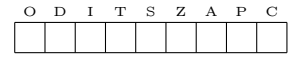
Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel286.

12.4.3 Instrukce pro konverzi typů

CBW *Convert Byte to Word*

Popis:



Instrukce CBW aritmeticky převádí obsah slabiky do 16bitového slova se zachováním znaménka. CBW konkrétně obsah registru AL rozšíří do AX tak, že zkopíruje znaménkový bit AL do všech bitů registru AH, tj. provede znaménkové rozšíření.

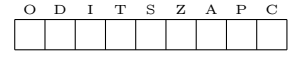
Instrukce	Popis
CBW	AX := znaménkové_rozšíření(AL)

Komentář

Viz též CWDE. Instrukce nemění příznaky a negeneruje výjimky.

CWD *Convert Word to Doubleword*

Popis:



Instrukce CWD aritmeticky převádí obsah 16bitového slova do 32bitového dvojslova se zachováním znaménka. CWD konkrétně obsah registru AX rozšíří do DX&AX tak, že zkopíruje znaménkový bit AX do všech bitů registru DX (tj. vyšší bity výsledku jsou v DX a nižší bity v AX).

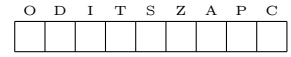
Instrukce	Popis
CWD	DX&AX := znaménkové_rozšíření(AX)

Komentář

Viz též CDQ.

CWDE

Convert Word to Doubleword



Popis:

Instrukce CWDE aritmeticky převádí obsah slova do dvojslova se zachováním znaménka. CWDE konkrétně obsah registru AX rozšíří do EAX tak, že zkopíruje znaménkový bit AX do všech bitů horní poloviny EAX.

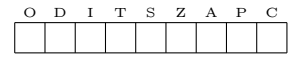
Instrukce	Popis
CWDE	EAX := znaménkové_rozšíření(AX)

Komentář

Instrukce má stejný operační znak jako CBW. O šířce zpracovávaného operandu rozhoduje typ instrukčního segmentu (16 nebo 32bitový) nebo instrukční prefix měnící velikost operandu.

CDQ

Convert Doubleword to Quadword



Popis:

Instrukce CDQ aritmeticky převádí obsah dvojslova do čtyřslova se zachováním znaménka. CDQ konkrétně obsah registru EAX rozšíří do registrového páru EDX&EAX tak, že zkopíruje znaménkový bit EAX do všech bitů EDX.

Instrukce	Popis
CDQ	EDX&EAX := znaménkové_rozšíření(EAX)

Komentář

Instrukce má stejný operační znak jako CWD. O šířce zpracovávaného operandu rozhoduje typ instrukčního segmentu (16 nebo 32bitový) nebo instrukční prefix měnící velikost operandu.

MOVSX
MOVSZ

Move with Sign-Extend
Move with Zero-Extend

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce MOVSX (MOVSZ) rozšíří číslo uložené ve druhém operandu znaménkově (v instrukci MOVSZ neznaménkově) na velikost prvního operandu a tuto hodnotu do prvního operandu uloží.

Instrukce	Popis
MOVSX *r16,r/m8*	přesuň slabiku do slova se znaménkovým rozšířením
MOVSX *r32,r/m8*	přesuň slabiku do dvojslova se znaménkovým rozšířením
MOVSX *r32,r/m16*	přesuň slovo do dvojslova se znaménkovým rozšířením
MOVZX *r16,r/m8*	přesuň slabiku do slova s neznaménkovým rozšířením
MOVZX *r32,r/m8*	přesuň slabiku do dvojslova s neznaménkovým rozšířením
MOVZX *r32,r/m16*	přesuň slovo do dvojslova s neznaménkovým rozšířením

Komentář

Instrukce MOVSX rozšiřuje obsah menšího operandu znaménkově, tzn. že instrukce zkopíruje nejvyšší (znaménkový) bit menšího operandu do všech bitů, o které operand rozšiřuje. Tím je zachováno znaménko původní hodnoty.

Instrukce MOVZX rozšiřuje obsah menšího operandu neznaménkově, tzn. že instrukce do všech bitů, o které operand rozšiřuje, vloží binární nuly.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce jsou zavedeny od procesoru Intel386.

12.5 Instrukce binární celočíselné aritmetiky

ADD

Integer Addition

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce ADD celočíselně přičítá obsah zdrojového operandu (druhý operand) k obsahu cílového operandu (první operand) a výsledek této operace umístí do cílového operandu. Obsah zdrojového operandu se přitom nezmění.

Instrukce	Popis
ADD *r/m8,r8*	Přičte obsah *r8* k *r8* nebo *m8*
ADD *r/m16,r16*	Přičte obsah *r16* k *r16* nebo *m16*
ADD *r/m32,r32*	Přičte obsah *r32* k *r32* nebo *m32*
ADD *r8,r/m8*	Přičte obsah *r8* nebo *m8* k *r8*
ADD *r16,r/m16*	Přičte obsah *r16* nebo *m16* k *r16*
ADD *r32,r/m32*	Přičte obsah *r32* nebo *m32* k *r32*
ADD *r/m8,imm8*	Přičte slabikovou přímou hodnotu k *r8* nebo *m8*
ADD *r/m16,imm16*	Přičte slovní přímou hodnotu k *r16* nebo *m16*
ADD *r/m32,imm32*	Přičte dvojslovní přímou hodnotu k *r32* nebo *m32*
ADD *r/m16,imm8*	Přičte znaménkově rozšířenou přímou hodnotu ke slovu *r/m*
ADD *r/m32,imm8*	Přičte znaménkově rozšířenou přímou hodnotu ke dvojslovu *r/m*

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl + znaménkové_rozšíření(Zdroj);
ELSE Cíl := Cíl + Zdroj;
FI;
```

Komentář

Pokud se k cílovému operandu šířky 16 nebo 32 bitů přičítá přímá hodnota šířky 8 bitů (*imm8*), rozšíří se tato přímá hodnota s respektováním znaménka na 16 nebo 32 bitů tak, že horní bity se naplní hodnotou znaménkového bitu dolní slabiky (pro kladné číslo budou všechny horní bity nulové, pro záporné číslo budou všechny jedničkové). Tím se podle pravidel dvojkového doplňkového kódu zajistí zachování znaménka při transformaci slabiky na slovo nebo dvojslovo.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

ADC — *Integer Addition with Carry*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce ADC přičte k cílovému operandu (první operand) obsah zdrojového operandu (druhý operand) a hodnotu příznaku přenosu (CF). Obsah zdrojového operandu zůstane nezměněn a příznak přenosu (CF) se nastaví podle konečného součtu. Operace je určena k postupnému sčítání operandů, jejichž šířka je větší než možná zpracovávaná.

Instrukce	Popis
ADC *r/m8,r8*	K *r/m8* přičte *r8* a CF
Další varianty viz ADD	

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl + znaménkové_rozšíření(Zdroj) + CF;
ELSE Cíl := Cíl + Zdroj + CF;
FI;
```

Komentář

Přičtení CF znamená přičtení jedničky nebo nuly k operandu. Toto přičtení probíhá podle pravidel dvojkového doplňkového kódu. Další informace jsou shodné s ADD.

SUB

Integer Subtraction

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Instrukce SUB odčítá od obsahu cílového operandu (první operand) obsah operandu zdrojového (druhý operand) a výsledek této operace ukládá do cílového operandu. Obsah zdrojového operandu zůstane nezměněn.

Instrukce	Popis
SUB *r/m8,r8*	Od *r/m8* odečte *r8*
Další varianty viz ADD	

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl - znaménkové_rozšíření(Zdroj);
ELSE Cíl := Cíl - Zdroj;
FI;
```

Komentář

Další informace viz ADD.

SBB

Integer Subtraction with Borrow

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Instrukce SBB odčítá od obsahu cílového operandu (první operand) obsah operandu zdrojového (druhý operand) a obsah příznaku přenosu (CF). Výsledek této operace ukládá do cílového operandu. Obsah zdrojového operandu zůstane nezměněn a příznak přenosu se nastaví podle výsledku operace. Instrukce se používá pro odčítání operandů, jejichž šířka je větší než možná zpracovávaná.

Instrukce	Popis
SBB *r/m8,r8*	Od *r/m8* odečte (*r8* + CF)
Další varianty viz ADD	

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl - (znaménkové_rozšíření(Zdroj) + CF);
ELSE Cíl := Cíl - (Zdroj + CF);
FI;
```

Komentář

Další informace viz ADD.

CMP

Compare Two Operands

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce CMP nastavuje příznaky, podle kterých lze porovnat obsah dvou operandů. Příznaky nastaví tak, že odečte druhý operand od prvního operandu a podle výsledku nastaví bity příznakového registru. Instrukce rozdíl neukládá (oba operandy zůstávají po provedení operace nezměněny). Nastavených příznaků lze využít některou instrukcí ze skupiny J*cond*.

Instrukce	Popis
CMP *r/m8,r8*	Od *r/m8* odečte *r8* a podle výsledku nastaví příznaky
Další varianty viz ADD	

Operace

```
IF druhý operand je slabika a první operand je slovo nebo dvojslovo
THEN nastav příznaky podle výsledku
             (první operand - znaménkové_rozšíření(druhý operand));
ELSE nastav příznaky podle výsledku (první operand - druhý operand);
FI;
```

Komentář

Ve skupině instrukcí podmíněných skoků (J*cond*) jsou instrukce pro znaménkové a neznaménkové porovnávání. Díky promyšlené konstrukci příznakového registru nastavuje instrukce CMP příznaky pro oba typy porovnávání. Znaménkové a neznaménkové porovnávání se liší až použitím instrukce J*cond*.

Výjimky jsou stejné jako u instrukce ADD.

INC

Increment by 1

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Instrukce INC zvyšuje hodnotu operandu o jedničku. Tato instrukce neovlivňuje příznak CF.

Instrukce	Popis
INC *r/m8*	Zvýší slabiku *r/m8* o 1
INC *r/m16*	Zvýší slovo *r/m16* o 1
INC *r/m32*	Zvýší dvojslovo *r/m32* o 1

Operace

```
Cíl := Cíl + 1;
```

Komentář

Zvýšení o jedničku je přičtením jedničky podle pravidel dvojkového doplňkového kódu. Pokud chceme, aby byl nastaven příznak CF, musíme použít ADD s druhým operandem 1.

Výjimky viz ADD.

DEC

Decrement by 1

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Popis:

Instrukce DEC snižuje hodnotu operandu o 1. Nemění nastavení příznaku CF.

Instrukce	Popis
DEC *r/m8*	Sníží slabiku *r/m8* o 1
DEC *r/m16*	Sníží slovo *r/m16* o 1
DEC *r/m32*	Sníží dvojslovo *r/m32* o 1

Operace

```
Cíl := Cíl - 1;
```

Komentář

Snížení o jedničku je odečtením jedničky podle pravidel dvojkového doplňkového kódu. Pokud chceme, aby byl nastaven příznak CF, musíme použít SUB s druhým operandem 1.

Výjimky viz ADD.

IMUL

Signed Multiplication

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	*

Popis:

Instrukce IMUL provádí celočíselné znaménkové násobení (tzn. respektuje znaménka činitelů). Instrukce má celou řadu variant operandů:

Instrukce	Popis
IMUL *r/m8*	AL := AL $*$ *r/m8*
IMUL *r/m16*	DX&AX := AX $*$ *r/m16*
IMUL *r/m32*	EDX&EAX := EAX $*$ *r/m32*
IMUL *r8,r/m8*	*r8* := *r8* $*$ *r/m8*
IMUL *r16,r/m16*	*r16* := *r16* $*$ *r/m16*
IMUL *r32,r/m32*	*r32* := *r32* $*$ *r/m32*
IMUL *r16,r/m16,imm8*	*r16* := *r/m16* $*$ znaménkové_rozšíření(*imm8*)
IMUL *r32,r/m32,imm8*	*r32* := *r/m32* $*$ znaménkové_rozšíření(*imm8*)
IMUL *r16,imm8*	*r16* := *r16* $*$ znaménkové_rozšíření(*imm8*)
IMUL *r32,imm8*	*r32* := *r32* $*$ znaménkové_rozšíření(*imm8*)
IMUL *r16,r/m16,imm16*	*r16* := *r/m16* $*$ *imm16*

IMUL *r32,r/m32,imm32*	*r32* := *r/m32* ∗ *imm32*
IMUL *r16,imm16*	*r16* := *r16* ∗ *imm16*
IMUL *r32,imm32*	*r32* := *r32* ∗ *imm32*

Operace

```
výsledek := násobenec * násobitel;
```

Komentář

Po vynásobení operandů jsou příznaky SF, ZF, AF a PF nedefinovány a příznaky OF a CF instrukce vynuluje za těchto podmínek (v opačném případě jsou nastaveny na jedničku):

Typ Instrukce	Podmínky Nulování CF a OF
r/m8	výsledné AX = znaménkové_rozšíření(výsledné AL)
r/m16	výsledné DX&AX = znaménkové_rozšíření(výsledné AX)
r/m32	výsledné EDX&EAX = znaménkové_rozšíření(výsledné EAX)
r16,r/m16	výsledek nepřekročil rozsah *r16*
r32,r/m32	výsledek nepřekročil rozsah *r32*
r16,r/m16,imm16	výsledek nepřekročil rozsah *r16*
r32,r/m32,imm32	výsledek nepřekročil rozsah *r32*

Pokud používáme instrukce typu *r/m8*, *r/m16*, *r/m32* (tj. instrukce ukládající výsledek do střádače), je k dispozici na uložení výsledku dvojnásobný prostor vzhledem k operandům. Tento prostor je dostatečně velký pro uložení všech možných výsledků, a proto je výsledek správný i při nastaveném příznaku OF.

Výjimky viz ADD.

Poznámky k předchozím procesorům

Varianty se třemi operandy jsou k dispozici od Intel286.

MUL *Unsigned Multiplication*

O	D	I	T	S	Z	A	P	C
∗				?	?	?	?	∗

Popis:

Instrukce MUL provádí celočíselné neznaménkové násobení (bit nejvyššího řádu se neuvažuje jako znaménkový).

Instrukce	Popis
MUL AL,*r/m8*	AL := AL ∗ *r/m8* neznaménkově
MUL AX,*r/m16*	DX&AX := AX ∗ *r/m16* neznaménkově
MUL EAX,*r/m32*	EDX&EAX := EAX ∗ *r/m32* neznaménkově

Operace

```
výsledek := neznaménkové násobení (násobenec,násobitel)
```

Komentář

Po vynásobení operandů jsou příznaky SF, ZF, AF a PF nedefinovány a příznaky OF a CF instrukce vynuluje tehdy, je-li horní polovina bitů výsledku nulová. V opačném případě jsou nastaveny na jedničku.

Výjimky viz ADD.

IDIV *Signed Divide*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Popis:

Instrukce IDIV provádí celočíselné znaménkové dělení obsahu registru AX, DX&AX nebo EDX&EAX (dělenec) operandem instrukce (dělitel). Je-li operand typu slabika, pak je dělencem obsah registru AX. Potom se výsledek (podíl) uloží do registru AL a zbytek po dělení do registru AH. Je-li operand 16bitový, dělí se obsah registrového páru DX&AX (DX obsahuje vyšších 16 bitů a AX nižších 16 bitů dělence). Podíl je potom uložen do registru AX a zbytek po dělení do registru DX. Je-li operand 32bitový, dělí se EDX&EAX. Podíl se ukládá do EAX a zbytek do EDX.

Instrukce	Popis
IDIV AL,*r/m8*	AL := AX ÷ *r/m8* a AH := AX *mod r/m8*
IDIV AX,*r/m16*	AX := DX&AX ÷ *r/m16* a DX := DX&AX *mod r/m16*
IDIV EAX,*r/m32*	EAX := EDX&EAX ÷ *r/m32* a EDX := EDX&EAX *mod r/m32*

Operace

```
pomocná := dělenec / dělitel;
IF pomocná se nevejde do registru podílu
THEN výjimka INT 0;
ELSE
  podíl := pomocná;
  zbytek := dělenec MOD dělitel;
FI;
```

Komentář

Zbytek má stejné znaménko jako dělenec. Absolutní hodnota zbytku je vždy menší než absolutní hodnota dělitele. Příznaky jsou po provedení instrukce nedefinovány.

Pokud je podíl větší než rozsah zobrazení registru pro jeho uložení nebo se dělí nulou, nastane v kterémkoli režimu výjimka 0. Ostatní výjimky jsou stejné s instrukcí ADD.

DIV *Unsigned Divide*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Popis:

Instrukce DIV provádí celočíselné neznaménkové dělení. Jinak má instrukce stejnou funkci a stejné operandy jako IDIV.

NEG *Two's Complement Negation*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce NEG nahradí obsah operandu jeho dvojkovým doplňkem (změní znaménko, tj. vynásobí operand hodnotou -1). Dvojkový doplněk je inverze všech bitů a přičtení jedničky nebo též odečtení od nuly.

Instrukce	Popis
NEG *r/m8*	*r/m8* := dvojkový doplněk(*r/m8*)
NEG *r/m16*	*r/m16* := dvojkový doplněk(*r/m16*)
NEG *r/m32*	*r/m32* := dvojkový doplněk(*r/m32*)

Operace

```
IF operand = 0 THEN CF := 0 ELSE CF := 1; FI;
operand := - operand;
```

Komentář

Příznaky OF, SF, ZF a PF se nastaví podle běžných pravidel. Příznak CF se nastaví na jedničku při nenulovém operandu.

Výjimky viz ADD.

CMPXCHG *Compare and Exchange*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce CMPXCHG porovnává obsah střádače (AL, AX nebo EAX) s obsahem prvního operandu. Pokud jsou shodné, nastaví příznak ZF na jedničku a první operand naplní obsahem druhého operandu. V opačném případě je vynulován příznak ZF a střádač (AL, AX nebo EAX) je naplněn prvním operandem.

Instrukce	Popis
CMPXCHG *r/m8,r8*	porovnej a vyměň
CMPXCHG *r/m16,r16*	porovnej a vyměň
CMPXCHG *r/m32,r32*	porovnej a vyměň

Operace

```
IF střádač = první operand
THEN
  ZF := 1;
  první operand := druhý operand;
ELSE
  ZF := 0;
  střádač := první operand;
FI;
```

Komentář

Instrukce se smí použít s prefixem LOCK. Protože procesor neprodukuje uzamknuté čtení bez uzamknutého zápisu, generuje se vždy po přečtení zápisový cyklus na sběrnici bez znalosti toho, jak dopadne porovnání. Pokud je shoda, zapíše se druhý operand; v opačném případě se do paměti zpět zapisuje první operand.

Nastavované příznaky

Instrukce nastavuje všechny obvyklé příznaky podle výsledku rozdílu: první_operand−střádač.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce je podporována od procesoru Intel486.

CMPXCHG8B *Compare and Exchange 8 Bytes*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukce CMPXCHG8B porovnává 64bitovou hodnotu v páru EDX&EAX s operandem. V registrovém páru EDX&EAX obsahuje EDX horních 32 bitů a EAX dolních 32 bitů testované hodnoty. Pokud je nalezena shoda, zkopíruje se 64 bitů z ECX&EBX do operandu. Operandem je 64bitový objekt v paměti.

Instrukce	Popis
CMPXCHG8B *r/m64*	porovnej a vyměň

Operace

```
IF EDX&EAX = první operand
THEN
  ZF := 1;
  první operand := ECX&EBX;
ELSE
  ZF := 0;
  EDX&EAX := první operand;
FI;
```

Komentář

Instrukce se smí použít s prefixem LOCK. Protože procesor neprodukuje uzamknuté čtení bez uzamknutého zápisu, generuje se vždy po přečtení zápisový cyklus na sběrnici bez znalosti toho, jak dopadne porovnání. Pokud je shoda, zapíše se ECX&EBX; v opačném případě se do paměti zpět zapisuje první operand.

Nastavované příznaky

Instrukce nastavuje pouze příznak ZF.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. Není-li operand paměťový, generuje se #UD.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. Není-li operand paměťový, generuje se #UD.

Poznámky k předchozím procesorům

Instrukce je podporována od procesoru Pentium.

12.6 Logické instrukce

AND

Logical AND

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Popis:

Instrukce AND provádí logický součin obsahu zdrojového (druhého) operandu s obsahem cílového (prvního) operandu a výsledek uloží do cílového operandu. Operace se provádí po bitech podle následující tabulky:

op_1	op_2	*AND*
0	0	0
0	1	0
1	0	0
1	1	1

Instrukce	Popis
AND *r/m8,r8*	*r/m8* := *r/m8* ∧ *r8*
AND *r/m16,r16*	*r/m16* := *r/m16* ∧ *r16*
AND *r/m32,r32*	*r/m32* := *r/m32* ∧ *r32*
AND *r8,r/m8*	*r8* := *r8* ∧ *r/m8*
AND *r16,r/m16*	*r16* := *r16* ∧ *r/m16*
AND *r32,r/m32*	*r32* := *r32* ∧ *r/m32*
AND *r/m8,imm8*	*r/m8* := *r/m8* ∧ *imm8*
AND *r/m16,imm16*	*r/m16* := *r/m16* ∧ *imm16*
AND *r/m32,imm32*	*r/m32* := *r/m32* ∧ *imm32*
AND *r/m16,imm8*	*r/m16* := *r/m16* ∧ znaménkové_rozšíření(*imm8*)
AND *r/m32,imm8*	*r/m32* := *r/m32* ∧ znaménkové_rozšíření(*imm8*)

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl AND znaménkové_rozšíření(Zdroj);
ELSE Cíl := Cíl AND Zdroj;
FI;
CF := 0;
OF := 0;
```

Komentář

Pokud se má provést logický součin operandu šířky 16 nebo 32 bitů s přímou hodnotou šířky 8 bitů (*imm8*), rozšíří se tato přímá hodnota s respektováním znaménka na 16 nebo 32 bitů tak, že horní bity se naplní hodnotou znaménkového bitu dolní slabiky (pro kladné číslo budou všechny horní bity nulové, pro záporné číslo budou všechny jedničkové). Tím se podle pravidel dvojkového doplňkového kódu zajistí zachování znaménka při transformaci slabiky na slovo nebo dvojslovo. Výjimky viz ADD.

OR

Logical Inclusive OR

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Popis:

Instrukce OR provádí logický součet obsahu zdrojového (druhého) operandu s obsahem cílového (prvního) operandu a výsledek uloží do cílového operandu. Operace se provádí po bitech podle následující tabulky:

op_1	op_2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Instrukce	Popis
OR *r/m8,r8*	$r/m8 := r/m8 \vee r8$

Další varianty viz AND

Operace

```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl OR znaménkové_rozšíření(Zdroj);
ELSE Cíl := Cíl OR Zdroj;
FI;
CF := 0;
OF := 0;
```

Komentář

Další informace viz AND.

XOR

Logical Exclusive OR

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Popis:

Instrukce XOR provádí logickou funkci nonekvivalence (součet modulo 2) obsahu zdrojového (druhého) operandu s obsahem cílového (prvního) operandu. Výsledek se uloží do cílového operandu. Operace se provádí po bitech podle následující tabulky:

op_1	op_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Instrukce	Popis
XOR *r/m8,r8*	$r/m8 := r/m8 \neq r8$

Další varianty viz AND

Operace

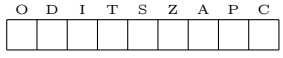
```
IF Zdroj je slabika a Cíl je slovo nebo dvojslovo
THEN Cíl := Cíl XOR znaménkové_rozšíření(Zdroj);
ELSE Cíl := Cíl XOR Zdroj;
FI;
CF := 0;
OF := 0;
```

Komentář

Další informace viz AND.

NOT — *One's Complement Negation*

Popis:



Instrukce NOT nahradí obsah operandu jeho jedničkovým doplňkem. Jedničkový doplněk je inverze všech bitů (všechny jedničky se nahradí nulami a obráceně).

Instrukce	Popis
NOT *r/m8*	$r/m8 := \overline{r/m8}$
NOT *r/m16*	$r/m16 := \overline{r/m16}$
NOT *r/m32*	$r/m32 := \overline{r/m32}$

Operace

```
operand := NOT operand;
```

Komentář

Instrukce NOT neovlivňuje registr příznaků. Výjimky viz ADD.

TEST *Logical Compare*

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Popis:

Instrukce TEST provádí logický součin (viz instrukce AND) obsahu druhého operandu s obsahem prvního operandu. Výsledek nikam neukládá, ale podle výsledku nastaví příznaky v příznakovém registru. Oba operandy jsou po provedení instrukce nezměněny.

Instrukce	Popis
TEST *r/m8,r8*	příznaky nastav podle *r/m8* ∧ *r8*
TEST *r/m16,r16*	příznaky nastav podle *r/m16* ∧ *r16*
TEST *r/m32,r32*	příznaky nastav podle *r/m32* ∧ *r32*
TEST *r/m8,imm8*	příznaky nastav podle *r/m8* ∧ *imm8*
TEST *r/m16,imm16*	příznaky nastav podle *r/m16* ∧ *imm16*
TEST *r/m32,imm32*	příznaky nastav podle *r/m32* ∧ *imm32*

Operace

```
Nastav SF, ZF a PF podle výsledku (první operand AND druhý operand);
CF := 0;
OF := 0;
```

Komentář

Instrukce na rozdíl od předchozích nedovoluje kombinace slova, či dvojslova se slabikovou přímou hodnotou.

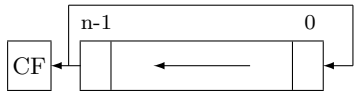
Výjimky viz ADD.

12.7 Rotace a posuvy

ROL *Rotate Left*

O	D	I	T	S	Z	A	P	C
*								*

Popis:



Instrukce ROL provádí rotaci vlevo bitů prvního operandu o jeden nebo více bitů zadaných druhým operandem.

Instrukce	Popis
ROL *r/m8*,1	Rotace slabiky *r/m8* o 1 bit vlevo
ROL *r/m8*,CL	Rotace slabiky *r/m8* o CL bitů vlevo
ROL *r/m8,imm8*	Rotace slabiky *r/m8* o *imm8* bitů vlevo
ROL *r/m16*,1	Rotace slova *r/m16* o 1 bit vlevo
ROL *r/m16*,CL	Rotace slova *r/m16* o CL bitů vlevo
ROL *r/m16,imm8*	Rotace slova *r/m16* o *imm8* bitů vlevo
ROL *r/m32*,1	Rotace dvojslova *r/m32* o 1 bit vlevo
ROL *r/m32*,CL	Rotace dvojslova *r/m32* o CL bitů vlevo
ROL *r/m32,imm8*	Rotace dvojslova *r/m32* o *imm8* bitů vlevo

Operace

```
pomocná := O_Kolik_Bitů_Rotovat;
WHILE (pomocná <> 0)
DO
  pomocnýCF := nejvyšší bit operandu;
  operand := operand*2 + pomocnýCF;
  pomocná := pomocná - 1;
OD;
IF O_Kolik_Bitů_Rotovat = 1
THEN
  IF nejvyšší bit operandu <> CF
  THEN OF := 1;
  ELSE OF := 0;
  FI;
ELSE OF := ?;
FI;
```

Komentář

Každý bit prvního operandu se posune vlevo, bit nejvyššího řádu se přenese na pozici nejnižšího řádu a současně se zkopíruje do příznakového bitu CF. Hodnota druhého operandu udává, o kolik bitů se první operand rotuje. Na místě druhého operandu smí být přímý operand s hodnotou 1, odkaz na registr CL, který obsahuje počet bitů, nebo přímá 8bitová hodnota. Při rotaci se žádný bit neztrácí.

Číslo v registru CL a přímá hodnota *imm8* určující počet bitů, o který se má rotovat a posouvat, se chápe jako číslo bez znaménka. Procesor 8086 nijak neomezoval počet rotací a posuvů umístěný v registru CL. Všechny vyšší typy procesorů berou z CL a z přímé hodnoty v úvahu pouze nejnižších 5 bitů (tj. lze rotovat a posouvat o nejvýše 31 bitů). Režim V86 rovněž bere v úvahu pouze dolních 5 bitů.

Hodnota příznaku OF je definována pouze po provedení rotace nebo posuvu o jeden bit (druhý operand je 1). V jiných případech je hodnota OF nedefinována.

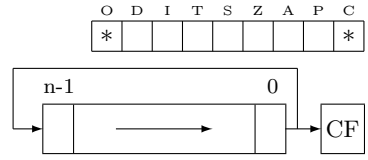
Při rotaci a posuvu vlevo je OF := CF $\neq$ bit$_{n-1}$ a při rotaci a posuvu vpravo je OF := bit$_{n-1}$ $\neq$ bit$_{n-2}$, tj. OF se nastaví, pokud se hodnota CF nerovná při rotaci a posuvu vlevo novému (při rotaci a posuvu vpravo starému) nejvyššímu bitu.

Výjimky viz ADD.

Poznámky k předchozím procesorům

Varianta s *imm8* u této a dalších instrukcí je dostupná od Intel286.

ROR *Rotate Right*



Popis:

Instrukce ROR provádí rotaci vpravo bitů prvního operandu o jeden nebo více bitů zadaných druhým operandem.

Instrukce	Popis
ROR *r/m8*,1	Rotace slabiky *r/m8* o 1 bit vpravo

Další varianty viz ROL

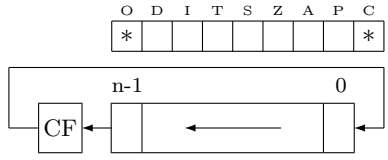
Operace

```
pomocná := O_Kolik_Bitů_Rotovat;
WHILE (pomocná <> 0)
DO
  pomocnýCF := nejnižší bit operandu;
  operand := operand/2 + (pomocnýCF * 2^šířka_operandu);
  pomocná := pomocná - 1;
OD;
IF O_Kolik_Bitů_Rotovat = 1
THEN
  IF nejvyšší bit operandu <> druhému nejvyššímu bitu operandu
  THEN OF := 1;
  ELSE OF := 0;
  FI;
ELSE OF := ?;
FI;
```

Komentář

Každý bit prvního operandu se posune vpravo, bit nejnižšího řádu se přenese na pozici nejvyššího řádu a současně se zkopíruje do příznakového bitu CF. Druhý operand udává počet rotací prvního operandu. Další podrobnosti jsou shodné s instrukcí ROL.

RCL — *Rotate Left through Carry*



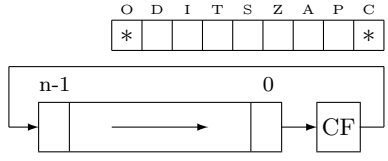
Popis:

Instrukce RCL provádí rotaci vlevo spojeného prvního operandu s příznakem CF o jeden nebo více bitů zadaných druhým operandem.

Komentář

Každý bit prvního operandu se posune vlevo, bit nejnižšího řádu se naplní hodnotou příznaku CF a bit z pozice nejvyššího řádu se přepíše do příznaku CF. Další podrobnosti jsou shodné s instrukcí ROL.

RCR — *Rotate Right through Carry*



Popis:

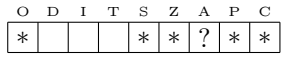
Instrukce RCR provádí rotaci vpravo spojeného prvního operandu s příznakem CF o jeden nebo více bitů zadaných druhým operandem.

Komentář

Každý bit prvního operandu se posune vpravo, bit nejvyššího řádu se naplní hodnotou příznaku CF a bit z pozice nejnižšího řádu se přepíše do příznaku CF. Další podrobnosti jsou shodné s instrukcí ROL.

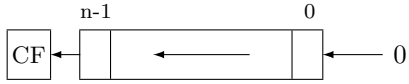
SAL
SHL

Shift Arithmetic Left
Shift Logical Left



Popis:

Obě instrukce SAL a SHL (synonymní označení jednoho operačního znaku) posouvají všechny bity prvního operandu vlevo o počet zadaný druhým parametrem.

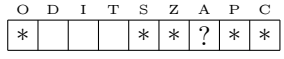


Komentář

Při posuvu se původní hodnota nejvyššího bitu prvního operandu přenáší do příznaku CF a nejnižší uvolněný bit se plní nulou. Jde o operaci znaménkového aritmetického násobení 2. Hodnota druhého operandu udává, o kolik bitů se první operand posouvá. Při posuvech se „ztrácí“ hodnota alespoň jednoho bitu. Další podrobnosti jsou shodné s instrukcí ROL.

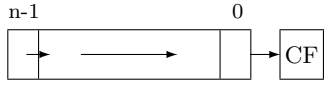
SAR

Shift Arithmetic Right



Popis:

Instrukce SAR posouvá všechny bity prvního operandu vpravo o počet zadaný druhým parametrem.



Komentář

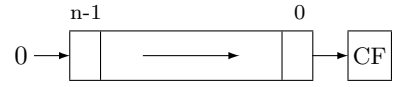
Při posunutí je nejnižší bit přepsán do příznaku CF a nejvyšší bit (znaménkový) zůstává nezměněn a kopíruje se do sousedního nižšího bitu. Výsledek instrukce SAR s posunutím pouze o jeden bit je shodný s provedením operace aritmetického znaménkového dělení prvního operandu 2. Při posuvech o jeden bit je OF vždy nulový. Instrukce SAR zachovává hodnotu znaménkového bitu, proto patří do skupiny aritmetických posuvů. Další podrobnosti viz ROL.

SHR

Shift Logical Right

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	?	*	*



Instrukce SHR posouvá všechny bity prvního operandu vpravo o počet zadaný druhým operandem.

Komentář

Při posunutí je nejnižší bit přepsán do příznaku CF a nejvyšší bit je naplněn 0. Při posuvech o jeden bit je do OF nastavena hodnota nejvyššího bitu původního posouvaného operandu. Instrukce SHR nezachovává hodnotu znaménkového bitu, a proto patří do skupiny logických posuvů. Další podrobnosti viz ROL.

SHLD
SHRD

Double Precision Shift Left

Double Precision Shift Right

Popis:

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	*

Instrukce SHLD (SHRD) provádí operaci logického posuvu doleva (doprava) hodnoty, která vznikne spojením prvního a druhého operandu, o počet bitů určený třetím operandem. Ve třetím operandu je významných pouze nejnižších 5 bitů (maximální hodnota je tedy 31).

Instrukce	Popis
SHLD *r/m16,r16,imm8*	SHL spojených *r/m16* a *r16*
SHLD *r/m32,r32,imm8*	SHL spojených *r/m32* a *r32*
SHLD *r/m16,r16,CL*	SHL spojených *r/m16* a *r16*
SHLD *r/m32,r32,CL*	SHL spojených *r/m32* a *r32*
SHRD *r/m16,r16,imm8*	SHR spojených *r/m16* a *r16*
SHRD *r/m32,r32,imm8*	SHR spojených *r/m32* a *r32*
SHRD *r/m16,r16,CL*	SHR spojených *r/m16* a *r16*
SHRD *r/m32,r32,CL*	SHR spojených *r/m32* a *r32*

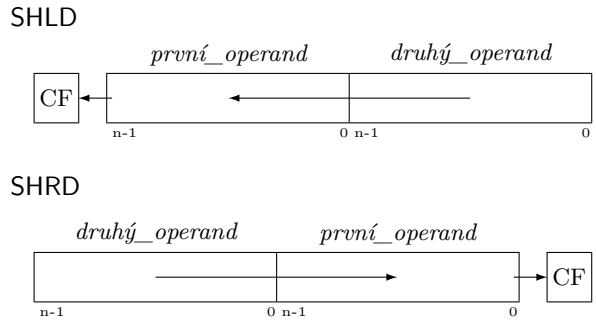
Operace

```
(* Čítač je neznaménkové celé číslo odpovídající poslednímu
   operandu instrukce, který je buď přímá hodnota imm8 nebo CL *)
Čítač := třetí_operand MOD 32;
IF Čítač = 0
THEN END; (* konec, není co dělat *)
ELSE
  IF Čítač >= velikost_operandu
  THEN (* chybný parametr *)
    první_operand := nedefinovaná hodnota;
    CF,OF,SF,ZF,AF,PF := nedefinovaná hodnota;
  ELSE (* proveď posuv *)
    CASE instrukce OF
SHLD:
    CF := Bit[první_operand, velikost_operandu - Čítač];
          (* poslední bit posunutý ven *)
    FOR i := velikost_operandu - 1 DOWNTO 0
    DO
      Bit[první_operand,i] := Bit[první_operand, i - Čítač];
    OD;
    FOR i := Čítač - 1 DOWNTO 0
    DO
      Bit[první_operand,i] := Bit[druhý_operand,i-Čítač+velikost_operandu];
    OD;
SHRD:
    CF := Bit[první_operand, Čítač - 1];
          (* poslední bit posunutý ven *)
    FOR i := 0 TO velikost_operandu - 1 - Čítač
    DO
      Bit[první_operand,i] := Bit[první_operand, i - Čítač];
    OD;
    FOR i := velikost_operandu - Čítač TO velikost_operandu - 1
    DO
      Bit[první_operand,i] := Bit[druhý_operand,i+Čítač-velikost_operandu];
    OD;

    ESAC;
    nastav SF,ZF,PF podle hodnoty prvního_operandu; (* tj. výsledku *)
    AF := nedefinovaná hodnota;
  FI;
FI;
```

Komentář

Algoritmus posuvu je patrný z obr. 12.5. Druhý operand zůstává po provedení instrukce beze změny. Výsledek je uložen v prvním operandu. Od instrukcí SHL (SHR) se tyto liší tím, že do posouvaného objektu nevstupují nuly, ale bity zadané druhý operandem.
Na místě třetího operandu smí být přímá hodnota nebo registr CL.



Obr. 12.5 Algoritmus instrukcí SHLD a SHRD

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce jsou zavedeny od procesoru Intel386.

12.8 Větvení programu

JMP

Unconditional Jump

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce nepodmíněného skoku JMP předá řízení instrukci, jejíž adresa je zadána operandem. Rozlišujeme tyto varianty instrukce JMP:

- Přímý skok. V operandu instrukce JMP je některým z následujících způsobů přímo specifikována cílová adresa.
  - Instrukce měnící obsah segmentového registru CS. Touto instrukcí lze předávat řízení na libovolnou adresu v adresovém prostoru. Instrukce JMP obsahuje přímou absolutní adresu (segmentovou i offsetovou část) cílového místa v paměti, kterou se naplní registry CS:EIP. Tuto operaci nazýváme **vzdálený skok** (Far Jump).
  - Instrukce neměnící obsah segmentového registru CS. Instrukce obsahuje pouze „vzdálenost“ místa, na které se má provést skok, od začátku následující instrukce. Tato relativní adresa se přičte ke stávajícímu obsahu registru EIP. Potom podle délky skoku (tj. rozdílu cílové adresy a obsahu EIP) rozlišujeme další dva typy:
    - Relativní adresa v instrukci JMP je 16 nebo 32bitová. Taková instrukce dovoluje předávat řízení uvnitř celého 64KB nebo 4GB segmentu tím, že k současnému obsahu EIP přičte relativní adresu, která je číslo v intervalu $-32\,768$ až $+32\,767$ pro 16bitovou adresu a -2^{31} až $+2^{31}-1$ pro 32bitovou adresu. Současný obsah EIP ukazuje na první slabiku následující instrukce. Tuto operaci nazýváme **blízký skok** (Near Jump).
    - Relativní adresa v instrukci JMP je 8bitová. Potom lze řízení předat pouze v intervalu -128 až 127 slabik od současného obsahu EIP. Tuto operaci nazýváme **krátký skok** (Short Jump).
- Nepřímý skok. V tomto případě je cílová adresa popsána buď odkazem na registr, který může obsahovat offsetovou část absolutní cílové adresy (SI, DI, SP, BP nebo BX), nebo přímou adresou místa v paměti, které obsahuje absolutní cílovou adresu (buď jenom offset, nebo segment:offset), nebo kombinací obou způsobů tak, jak to bylo popsáno v odstavci 2.5 od strany 26.

V chráněném režimu se navíc instrukce vzdáleného skoku používají pro předávání řízení mezi segmenty a procesy tak, že selektor ukazuje na popisovač příslušné tabulky popisovačů segmentů.

Instrukce	Popis
JMP *rel8*	Krátký skok
JMP *rel16*	Přímý blízký skok 16bitový
JMP *r/m16*	Nepřímý blízký skok 16bitový
JMP *ptr16:16*	Přímý vzdálený skok podle 4slabikové adresy
JMP *ptr16:16*	Skok na bránu pro předání řízení na stejné úrovni
JMP *ptr16:16*	Skok přes TSS
JMP *ptr16:16*	Skok přes bránu zpřístupňující TSS
JMP *m16:16*	Nepřímý vzdálený skok podle 4slabikové adresy ukazující na adresu v paměti
JMP *m16:16*	Nepřímý vzdálený skok podle 4slabikové adresy na bránu pro předání řízení na stejné úrovni
JMP *m16:16*	Nepřímý vzdálený skok podle 4slabikové adresy přes TSS
JMP *m16:16*	Nepřímý vzdálený skok podle 4slabikové adresy přes bránu zpřístupňující TSS
JMP *rel32*	Přímý blízký skok 32bitový
JMP *r/m32*	Nepřímý blízký skok 32bitový
JMP *ptr16:32*	Přímý vzdálený skok podle 6slabikové adresy
JMP *ptr16:32*	Skok na bránu pro předání řízení na stejné úrovni
JMP *ptr16:32*	Skok přes TSS
JMP *ptr16:32*	Skok přes bránu zpřístupňující TSS
JMP *m16:32*	Nepřímý vzdálený skok podle 6slabikové adresy ukazující na adresu v paměti
JMP *m16:32*	Nepřímý vzdálený skok podle 6slabikové adresy na bránu pro předání řízení na stejné úrovni
JMP *m16:32*	Nepřímý vzdálený skok podle 6slabikové adresy přes TSS
JMP *m16:32*	Nepřímý vzdálený skok podle 6slabikové adresy přes bránu zpřístupňující TSS

Operace

```
IF instrukce je relativní skok (* tj. operand rel8, rel16, rel32 *)
THEN
  EIP := EIP + rel8/16/32;
  IF nastavená velikost operandu = 16
  THEN EIP := EIP AND 0000FFFFh;
  FI;
FI;
```

```
IF instrukce je blízký nepřímý skok (* tj. operand r/m16, r/m32 *)
THEN
  IF nastavená velikost operandu = 16
  THEN
    EIP := [r/m16] AND 0000FFFFh;
  ELSE (* nastavená velikost operandu = 32 *)
    EIP := [r/m32];
  FI;
FI;

IF (PE=0 OR (PE=1 AND VM=1)) (* reálný nebo V86 režim *)
   AND instrukce je vzdálený skok
       (* tj. operand m16:16, m16:32, ptr16:16, ptr16:32 *)
THEN
  IF typ operandu = m16:16 nebo m16:32
  THEN (* nepřímý vzdálený skok v reálném nebo V86 režimu *)
    IF nastavená velikost operandu = 16
    THEN
      CS:IP := [m16:16];
      EIP := EIP AND 0000FFFFh; (* vynulování horní poloviny *)
    ELSE (* nastavená velikost operandu = 32 *)
      CS:EIP := [m16:32];
    FI;
  ELSE (* přímý vzdálený skok v reálném nebo V86 režimu *)
    IF nastavená velikost operandu = 16
    THEN
      CS:IP := ptr16:16;
      EIP := EIP AND 0000FFFFh; (* vynulování horní poloviny *)
    ELSE (* nastavená velikost operandu = 32 *)
      CS:EIP := ptr16:32;
    FI;
  FI;
FI;

IF (PE=1 AND VM=0)  (* Chráněný režim a ne V86 režim *)
   AND instrukce je vzdálený skok
       (* tj. operand m16:16, m16:32, ptr16:16, ptr16:32 *)
THEN
  IF typ operandu = m16:16 nebo m16:32
  THEN (* nepřímý vzdálený skok v chráněném a ne V86 režimu *)
    kontrola přístupu;
    není překročen limit, jinak #GP(0) nebo #SS(0);
```

```
  FI;
  cílový selektor není neplatný, jinak #GP(0);
  cílový index ze selektoru ukazuje pod limit tabulky, jinak #GP(selektor);
  podle AR slabiky cílového selektoru:
    GOTO PŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT;
    GOTO NEPŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT;
    GOTO BRÁNA-PRO-PŘEDÁNÍ-ŘÍZENÍ;
    GOTO BRÁNA-ZPŘÍSTUPŇUJÍCÍ-TSS;
    GOTO TSS;
    jinak #GP(selektor);  (* chybný obsah AR slabiky *)
FI;

PŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT:
  DPL popisovače je menší nebo rovno CPL, jinak #GP(selektor);
  segment musí být v AR slabice označen jako přítomný, jinak #NP(selektor);
  offset musí ukazovat dovnitř limitu segmentu, jinak #GP(0);
  IF nastavená velikost operandu = 32
  THEN naplň CS:EIP podle cílového popisovače;
  ELSE naplň CS:IP podle cílového popisovače;
  FI;
  naplň CS novým popisovačem;

NEPŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT:
  RPL cílového popisovače je menší nebo rovno CPL, jinak #GP(selektor);
  DPL popisovače je rovno CPL, jinak #GP(selektor);
  segment musí být v AR slabice označen jako přítomný, jinak #NP(selektor);
  offset musí ukazovat dovnitř limitu segmentu, jinak #GP(0);
  IF nastavená velikost operandu = 32
  THEN naplň CS:EIP podle cílového popisovače;
  ELSE naplň CS:IP podle cílového popisovače;
  FI;
  naplň CS novým popisovačem;
  nastav RPL v CS na CPL;

BRÁNA-PRO-PŘEDÁNÍ-ŘÍZENÍ:
  DPL popisovače je větší nebo rovno CPL, jinak #GP(selektor brány);
  DPL popisovače je větší nebo rovno RPL selektoru brány,
       jinak #GP(selektor brány);
  brána je označena jako přítomná, jinak #NP(selektor brány);
  kontrola selektoru uloženého v bráně:
    selektor není neplatný, jinak #GP(0);
    selektor ukazuje dovnitř limitu tabulky, jinak #GP(selektor CS);
    slabika AR označuje instrukční segment, jinak #GP(selektor CS);
```

```
    IF není přizpůsobitelný
    THEN DPL popisovače instrukčního segmentu musí = CPL
    ELSE #GP(selektor CS);
    FI;
    IF přizpůsobitelný
    THEN DPL popisovače instrukčního segmentu musí být <= CPL
    ELSE #GP(selektor CS);
    FI;
    instrukční segment musí být přítomný, jinak #NP(selektor CS);
    offset musí ukazovat dovnitř limitu segmentu, jinak #GP(0);
    IF nastavená velikost operandu = 32
    THEN naplň CS:EIP podle brány;
    ELSE naplň CS:IP podle brány;
    FI;
  naplň CS novým popisovačem instrukčního segmentu;
  nastav RPL v CS na CPL;

BRÁNA-ZPŘÍSTUPŇUJÍCÍ-TSS:
  DPL popisovače brány je >= CPL, jinak #GP(selektor brány);
  DPL popisovače brány je >= RPL popisovače brány, jinak #GP(selektor brány);
  příznak přítomnosti je v bráně nastaven, jinak #NP(selektor brány);
  kontrola selektoru uloženého v bráně:
    ukazuje do globálního adresového prostoru, jinak #GP(selektor TSS);
    index ukazuje dovnitř limitu GDT, jinak #GP(selektor TSS);
    slabika AR indikuje neaktivní TSS (nejnižší bity 00001),
                         jinak #GP(selektor TSS);
    TSS musí být přítomný, jinak #NP(selektor TSS);
  přepnutí procesu (bez vnoření) podle TSS;
  nové EIP ukazuje dovnitř limitu instrukčního segmentu, jinak #GP(0);

TSS:
  DPL popisovače TSS je >= CPL, jinak #GP(selektor TSS);
  DPL popisovače TSS je >= RPL selektoru TSS, jinak #GP(selektor TSS);
  slabika AR indikuje neaktivní TSS (nejnižší bity 00001),
                         jinak #GP(selektor TSS);
  TSS musí být přítomný, jinak #NP(selektor TSS);
  přepnutí procesu (bez vnoření) podle TSS;
  nové EIP ukazuje dovnitř limitu instrukčního segmentu, jinak #GP(0);
```

Komentář

Skokové instrukce s operandy *r/m16*, *r/m32*, *rel16* a *rel32* jsou blízkými skoky, protože nemění segmentovou část adresy. Operandy *rel16* a *rel32* jsou přírůstky, které se přičtou k EIP. EIP v okamžiku provádění instrukce ukazuje již na instrukci

následující. Je-li nastavena velikost operandu na 16 bitů nebo se zpracovává 16bitový segment, použije se *rel16*. Varianta *rel32* se použije při nastavené velikosti 32 bitů nebo ve 32bitových segmentech. Výsledek se vždy ukládá do 32bitového registru EIP. Používá-li se 16bitová varianta, vynuluje se horních 16 bitů.

Varianty instrukcí JMP *r/m16* a *r/m32* jsou nepřímými skoky. Operand specifikuje buď registr, nebo místo v paměti s absolutním offsetem cílové adresy. Mezi *r/m16* a *r/m32* se opět vybírá podle nastavené velikosti operandu.

Instrukce s operandy *ptr16:16* a *ptr16:32* mají v instrukci uloženy 4 nebo 6 slabik segmentu a offsetu cílového místa. Jde tedy o vzdálené skoky. Varianty *m16:16* a *m16:32* jsou vzdálenými nepřímými skoky. Adresa cílového místa je uložena na 4 nebo 6 slabikách v paměti na adrese, na kterou ukazuje operand. V reálném a V86 režimu obsahují cílové adresy vždy 16 bitů nové hodnoty registru CS a 16 nebo 32 bitů nové hodnoty registru EIP. V chráněném režimu instrukce JMP vždy analyzuje slabiku AR z popisovače, na který ukazuje index selektoru v prvních 16 bitech cílové adresy. Může totiž jít o předání řízení do instrukčního segmentu na stejné úrovni oprávnění nebo o přepnutí procesu.

Nastavované příznaky

V případě, že instrukce přepíná proces, mění se obsah všech příznaků. V jiných případech zůstávají příznaky beze změny.

Výjimky ve chráněném režimu

Vzdálené skoky generují výjimky #GP, #NP, #SS a #TS tak, jak to bylo popsáno výše.

Blízké přímé skoky generují #GP(0) tehdy, pokud cílové místo leží vně limitu instrukčního segmentu, při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Blízké nepřímé skoky generují tyto výjimky: Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). #GP se generuje tehdy, leží-li nepřímý offset vně limitu segmentu. Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

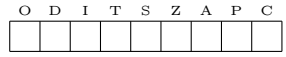
Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

J*cond* — *Conditional Jumps*



Popis:

Pod zkratkou J*cond* se skrývají všechny instrukce podmíněného skoku. Instrukce J*cond* (s výjimkou instrukce JCXZ) testují hodnotu jednoho nebo více jednobitových příznaků příznakového registru.

Odpovídá-li hodnota příznaků kombinaci, která je testována konkrétní instrukcí podmíněného skoku, provede se krátký nebo blízký skok zadaný relativní adresou. Neodpovídá-li, pokračuje se další instrukcí.

Instrukce podmíněného skoku nejčastěji následuje po instrukci CMP, která nastaví příznaky odpovídající rozdílu testovaných operandů. Příznaky jsou konstruovány tak, že operandy vstupující do instrukce CMP mohou být jak čísla bez znaménka, tak i se znaménkem. Byly-li operandy čísla se znaménkem, musí se vybrat instrukce, které jsou dále označeny zkratkou „**z.**“. Byly-li operandy čísla bez znaménka, musí se vybrat instrukce označené „**nz.**“

Instrukce	Popis
J*cond rel8*	Krátký skok při splnění podmínky
J*cond rel16/32*	Blízký skok při splnění podmínky
JCXZ *rel8*	Krátký skok při CX=0
JECXZ *rel8*	Krátký skok při ECX=0

Následuje přehled možných instrukcí podmíněného skoku. Lomítkem jsou odděleny ekvivalentní názvy.

Zkratka instrukce	*Název instrukce (Jump short if …)*	*Výsledek poslední operace …*	*Testovaná podmínka*
JE/JZ	equal	roven	ZF=1
	zero	nulový	
JNE/JNZ	not equal	různý	ZF=0
	not zero	nenulový	
JP/JPE	parity	sudé parity	PF=1

	parity even		
JNP/JPO	not parity	liché parity	PF=0
	parity odd		
JS	sign	záporný	SF=1
JNS	not sign	kladný nebo nulový	SF=0
JC	carry	nastal přenos	CF=1
JNC	not carry	nenastal přenos	CF=0
JO	overflow	nastalo přeplnění	OF=1
JNO	not overflow	nenastalo přeplnění	OF=0
JB/JNAE	below	nz. menší	CF=1
	not above nor equal		
JAE/JNB	above or equal	nz. větší nebo roven	CF=0
	not below		
JBE/JNA	below or equal	nz. menší nebo roven	(CF=1)∨
	not above		∨(ZF=1)
JA/JNBE	above	nz. větší	(CF=0)∧
	not below nor equal		∧(ZF=0)
JL/JNGE	less	z. menší	SF≠OF
	not greater nor equal		
JGE/JNL	greater or equal	z. větší nebo roven	SF=OF
	not less		
JLE/JNG	less or equal	z. menší nebo roven	(ZF=1)∨
	not greater		∨(SF≠OF)
JG/JNLE	greater	z. větší	(ZF=0)∧
	not less nor equal		∧(SF=OF)

Porovnávání typu menší a větší je rozděleno na neznaménkové a znaménkové. Neznaménkové porovnávání (v přehledu uvedeno zkratkou „**nz.**“ a v originále slovy „above“ a „below“) je určeno pro čísla bez znamének (bit nejvyššího řádu se neuvažuje jako znaménkový). Znaménkové porovnávání (zkratka „**z.**“ a v originále „less“ a „greater“) probíhá podle pravidel dvojkového doplňkového kódu s respektováním znamének.

Operace

```
IF podmínka je splněna
THEN
  EIP := EIP + znaménkové_rozšíření(rel8/16/32);
  IF nastavená velikost operandu = 16
  THEN EIP := EIP AND 0000FFFFh;
  FI;
FI;
```

Komentář

Instrukce JCXZ se liší od ostatních instrukcí podmíněného skoku tím, že testuje místo jednobitových příznaků obsah registru CX. Instrukce JECXZ testuje ECX. Pokud je testovaný registr nulový, provede se krátký skok na specifikovanou adresu. Příznaky instrukce nekontroluje, ani nenastavuje. Instrukce se používá jako pomocná pro řízení cyklů. Např. ji lze použít na začátku těla cyklu pro testování, zda řídicí proměnná cyklu (CX) je nulová. Při použití instrukce např. LOOP by nulová hodnota řídicí proměnné vedla k 64 K nebo 4 G iteracím těla cyklu. Instrukce nemá variantu s blízkým skokem. Je vhodné upozornit, že trvá déle než instrukce porovnání a následná instrukce podmíněného skoku.

Výjimky ve chráněném režimu

Ukazuje-li offset vně limitu instrukčního segmentu, generuje se #GP(0).

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

Žádné.

SET*cond* — *Byte Set on Condition*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce SET*cond* nastaví operand podle obsahu registru příznaků. Vyhovuje-li hodnota příznaků v příznakovém registru podmínce, která je zadána operačním znakem instrukce SET*cond*, nastaví se operand na hodnotu 1. Nevyhovuje-li obsah registru příznaků podmínce, je operand nastaven na hodnotu 0.

Instrukce	Popis
SET*cond r/m8*	nastav na 1 při splnění podmínky

Operace

```
IF podmínka je splněna THEN r/m8 := 1; ELSE r/m8 := 0; FI;
```

Komentář

Podmínky, které lze uvést v instrukci SET*cond*, jsou totožné s podmínkami užívanými instrukcí J*cond*. Přesto je pro pohodlí čtenáře uvedeme (výklad seznamu viz v popisu instrukce J*cond*):

Zkratka instrukce	*Název instrukce (Set if …)*	*Výsledek poslední operace …*	*Testovaná podmínka*
SETE/SETZ	equal zero	roven nulový	ZF=1
SETNE/SETNZ	not equal not zero	různý nenulový	ZF=0
SETP/SETPE	parity parity even	sudé parity	PF=1
SETNP/SETPO	not parity parity odd	liché parity	PF=0
SETS	sign	záporný	SF=1
SETNS	not sign	kladný nebo nulový	SF=0
SETC	carry	nastal přenos	CF=1
SETNC	not carry	nenastal přenos	CF=0
SETO	overflow	nastalo přeplnění	OF=1
SETNO	not overflow	nenastalo přeplnění	OF=0
SETB/SETNAE	below not above nor equal	nz. menší	CF=1
SETAE/SETNB	above or equal not below	nz. větší nebo roven	CF=0
SETBE/SETNA	below or equal not above	nz. menší nebo roven	(CF=1)∨ ∨(ZF=1)
SETA/SETNBE	above not below nor equal	nz. větší	(CF=0)∧ ∧(ZF=0)
SETL/SETNGE	less not greater nor equal	z. menší	SF≠OF
SETGE/SETNL	greater or equal not less	z. větší nebo roven	SF=OF
SETLE/SETNG	less or equal not greater	z. menší nebo roven	(ZF=1)∨ ∨(SF≠OF)
SETG/SETNLE	greater not less nor equal	z. větší	(ZF=0)∧ ∧(SF=OF)

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce je k dispozici od procesoru Intel386.

CALL *Call Procedure*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce CALL předá řízení podprogramu, jehož počáteční adresa je specifikována operandem instrukce, a současně do zásobníku uloží návratovou adresu (tj. adresu ukazující bezprostředně za tuto instrukci CALL). Návrat z podprogramu provede instrukce RET. V chráněném režimu má tato instrukce další významné funkce.

Instrukce	Popis
CALL *rel16*	Přímé blízké volání 16bitové
CALL *r/m16*	Nepřímé blízké volání 16bitové
CALL *ptr16:16*	Přímé vzdálené volání podle 4slabikové adresy
CALL *ptr16:16*	Volání brány pro předání řízení na stejné úrovni
CALL *ptr16:16*	Volání brány pro předání řízení na vyšší úrovni
CALL *ptr16:16*	Volání TSS
CALL *m16:16*	Nepřímé vzdálené volání podle 4slabikové adresy ukazující na adresu v paměti
CALL *m16:16*	Nepřímé vzdálené volání podle 4slabikové adresy brány pro předání řízení na stejné úrovni
CALL *m16:16*	Nepřímé vzdálené volání podle 4slabikové adresy

	brány pro předání řízení na vyšší úrovni
CALL *m16:16*	Nepřímé vzdálené volání podle 4slabikové adresy TSS
CALL *rel32*	Přímé blízké volání 32bitové
CALL *r/m32*	Nepřímé blízké volání 32bitové
CALL *ptr16:32*	Přímé vzdálené volání podle 6slabikové adresy
CALL *ptr16:32*	Volání brány pro předání řízení na stejné úrovni
CALL *ptr16:32*	Volání brány pro předání řízení na vyšší úrovni
CALL *ptr16:32*	Volání TSS
CALL *m16:32*	Nepřímé vzdálené volání podle 6slabikové adresy ukazující na adresu v paměti
CALL *m16:32*	Nepřímé vzdálené volání podle 6slabikové adresy brány pro předání řízení na stejné úrovni
CALL *m16:32*	Nepřímé vzdálené volání podle 6slabikové adresy brány pro předání řízení na vyšší úrovni
CALL *m16:32*	Nepřímé vzdálené volání podle 6slabikové adresy TSS

Operace

```
IF instrukce je relativní volání (* tj. operand rel16, rel32 *)
THEN
  IF velikost operandu = 16
  THEN
    Push(IP);
    EIP := (EIP + rel16) AND 0000FFFFh;
  ELSE  (* velikost operandu = 32 *)
    Push(EIP);
    EIP := EIP + rel32;
  FI;
FI;

IF instrukce je blízké nepřímé volání (* tj. operand r/m16, r/m32 *)
THEN
  IF nastavená velikost operandu = 16
  THEN
    Push(IP);
    EIP := [r/m16] AND 0000FFFFh;
  ELSE (* nastavená velikost operandu = 32 *)
    EIP := [r/m32];
  FI;
FI;

IF (PE=0 OR (PE=1 AND VM=1)) (* reálný nebo V86 režim *)
```

```
   AND instrukce je vzdálené volání
       (* tj. operand m16:16, m16:32, ptr16:16, ptr16:32 *)
THEN
  IF velikost operandu = 16
  THEN
    Push(CS);
    Push(IP);  (* adresa následující instrukce za CALL, 16bitová *)
  ELSE (* velikost operandu = 32 *)
    Push(CS);  (* v horní polovině doplněno na 32 bitů *)
    Push(EIP); (* adresa následující instrukce za CALL, 32bitová *)
  FI;
  IF typ operandu = m16:16 nebo m16:32
  THEN (* nepřímé vzdálené volání v reálném nebo V86 režimu *)
    IF nastavená velikost operandu = 16
    THEN
      CS:IP := [m16:16];
      EIP := EIP AND 0000FFFFh; (* vynulování horní poloviny *)
    ELSE (* nastavená velikost operandu = 32 *)
      CS:EIP := [m16:32];
    FI;
  ELSE (* přímé vzdálené volání v reálném nebo V86 režimu *)
    IF nastavená velikost operandu = 16
    THEN
      CS:IP := ptr16:16;
      EIP := EIP AND 0000FFFFh; (* vynulování horní poloviny *)
    ELSE (* nastavená velikost operandu = 32 *)
      CS:EIP := ptr16:32;
    FI;
  FI;
FI;

IF (PE=1 AND VM=0)  (* Chráněný režim a ne V86 režim *)
   AND instrukce je vzdálené volání
       (* tj. operand m16:16, m16:32, ptr16:16, ptr16:32 *)
THEN
  IF typ operandu = m16:16 nebo m16:32
  THEN (* nepřímé vzdálené volání v chráněném a ne V86 režimu *)
    kontrola přístupu;
    není překročen limit, jinak #GP(0);
  FI;
  cílový selektor CS není neplatný, jinak #GP(0);
  cílový index ze selektoru ukazuje pod limit tabulky,
                                    jinak #GP(nový selektor CS);
```

```
  podle AR slabiky cílového selektoru:
    GOTO PŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT;
    GOTO NEPŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT;
    GOTO BRÁNA-PRO-PŘEDÁNÍ-ŘÍZENÍ;
    GOTO BRÁNA-ZPŘÍSTUPŇUJÍCÍ-TSS;
    GOTO TSS;
    jinak #GP(selektor instr. seg.);  (* chybný obsah AR slabiky *)
FI;

PŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT:
  DPL popisovače je menší nebo rovno CPL, jinak #GP(selektor instr. seg.);
  segment je přítomný, jinak #NP(selektor instr. seg.);
  zásobník má dostatek prostoru na uložení návratové adresy, jinak #SS(0);
  offset musí ukazovat dovnitř limitu segmentu, jinak #GP(0);
  naplň neviditelnou část CS;
  naplň selektor CS;
  naplň EIP;
  IF nastavená velikost operandu = 16 THEN EIP := EIP AND 0000FFFFh; FI;

NEPŘIZPŮSOBITELNÝ-INSTRUKČNÍ-SEGMENT:
  RPL cílového popisovače <= CPL, jinak #GP(selektor instr. seg.);
  DPL popisovače = CPL, jinak #GP(selektor instr. seg.);
  segment je přítomný, jinak #NP(selektor instr. seg.);
  zásobník má dostatek prostoru na uložení návratové adresy, jinak #SS(0);
  offset musí ukazovat dovnitř limitu segmentu, jinak #GP(0);
  naplň neviditelnou část CS;
  naplň selektor CS;
  nastav RPL v CS na CPL;
  naplň EIP;
  IF nastavená velikost operandu = 16 THEN EIP := EIP AND 0000FFFFh; FI;

BRÁNA-PRO-PŘEDÁNÍ-ŘÍZENÍ:
  DPL brány >= CPL, jinak #GP(selektor brány);
  DPL brány >= RPL, jinak #GP(selektor brány);
  brána je označena jako přítomná, jinak #NP(selektor brány);
  kontrola selektoru uloženého v bráně:
    selektor není neplatný, jinak #GP(0);
    selektor ukazuje dovnitř limitu tabulky, jinak #GP(selektor inst. seg.);
    slabika AR označuje instrukční segment, jinak #GP(selektor inst. seg.);
    DPL vybraného popisovače <= CPL, jinak #GP(selektor inst. seg.);
    IF není přizpůsobitelný inst. segment AND DPL < CPL
    THEN GOTO VYŠŠÍ-ÚROVEŇ-OPRÁVNĚNÍ;
    ELSE GOTO STEJNÁ-ÚROVEŇ-OPRÁVNĚNÍ;
```

```
  FI;

VYŠŠÍ-ÚROVEŇ-OPRÁVNĚNÍ:
  vezmi nový selektor SS pro novou úroveň oprávnění z TSS;
    zkontroluj selektor a popisovač nového SS:
      selektor není neplatný, jinak #TS(0);
      index selektoru ukazuje dovnitř limitu, jinak #TS(selektor SS);
      RPL selektoru = DPL instr. segmentu, jinak #TS(selektor SS);
      DPL zásobníku = DPL  instr. segmentu, jinak #TS(selektor SS);
      slabika AR označuje zapisovatelný segment, jinak #TS(selektor SS);
      segment je přítomný, jinak #SS(selektor SS);
    IF nastavená velikost operandu = 32
    THEN
      zásobník má prostor na parametry + 16 B, jinak #SS(selektor SS);
      EIP je uvnitř limitu instr. segmentu, jinak #GP(0);
      naplň SS:eSP z TSS;
      naplň CS:EIP z brány;
    ELSE (* nastavená velikost operandu = 16 *)
      zásobník má prostor na parametry + 8 B, jinak #SS(selektor SS);
      IP je uvnitř limitu instr. segmentu, jinak #GP(0);
      naplň SS:eSP z TSS;
      naplň CS:IP z brány;
    FI;
    naplň neviditelnou část CS;
    naplň neviditelnou část SS;
    vlož vzdálený ukazatel na starý zásobník do nového;
    vezmi počet slov z brány pro předání řízení a maskuj jej na 5 bitů;
    zkopíruj zjištěný počet parametrů ze starého zásobníku do nového;
    vlož návratovou adresu do nového zásobníku;
    nastav CPL na DPL zásobníkového segmentu;
    nastav RPL v CS na CPL;

STEJNÁ-ÚROVEŇ-OPRÁVNĚNÍ:
    IF nastavená velikost operandu = 32
    THEN
      zásobník má prostor na 6 B návratové adresy (doplněno na 8),
             jinak #SS(0);
      EIP je uvnitř limitu instr. segmentu, jinak #GP(0);
      naplň CS:EIP z brány;
    ELSE (* nastavená velikost operandu = 16 *)
      zásobník má prostor na 4 B návratové adresy, jinak #SS(0);
      IP je uvnitř limitu instr. segmentu, jinak #GP(0);
      naplň CS:IP z brány;
```

```
      FI;
      vlož návratovou adresu do zásobníku;
      naplň neviditelnou část CS;
      nastav RPL v CS na CPL;

BRÁNA-ZPŘÍSTUPŇUJÍCÍ-TSS:
  DPL popisovače brány je >= CPL, jinak #TS(selektor brány);
  DPL popisovače brány je >= RPL, jinak #TS(selektor brány);
  příznak přítomnosti je v bráně nastaven, jinak #NP(selektor brány);
  kontrola selektoru uloženého v bráně:
    ukazuje do globálního adresového prostoru, jinak #TS(selektor TSS);
    index ukazuje dovnitř limitu GDT, jinak #TS(selektor TSS);
    slabika AR indikuje neaktivní TSS (nejnižší bity 00001),
                         jinak #TS(selektor TSS);
    TSS musí být přítomný, jinak #NP(selektor TSS);
  přepnutí procesu (s vnořením) podle TSS;
  nové EIP ukazuje dovnitř limitu instrukčního segmentu, jinak #TS(0);

TSS:
  DPL popisovače TSS je >= CPL, jinak #TS(selektor TSS);
  DPL popisovače TSS je >= RPL selektoru TSS, jinak #TS(selektor TSS);
  slabika AR indikuje neaktivní TSS (nejnižší bity 00001),
                         jinak #TS(selektor TSS);
  TSS musí být přítomný, jinak #NP(selektor TSS);
  přepnutí procesu (s vnořením) podle TSS;
  nové EIP ukazuje dovnitř limitu instrukčního segmentu, jinak #TS(0);
```

Komentář

Instrukce CALL s operandy *r/m16*, *r/m32*, *rel16* a *rel32* jsou blízkými voláními, protože nemění a neukládají segmentovou část adresy. Operandy *rel16* a *rel32* jsou přírůstky, které se přičtou k EIP. EIP v okamžiku provádění instrukce ukazuje již na instrukci následující. Je-li nastavena velikost operandu na 16 bitů nebo se zpracovává 16bitový segment, použije se *rel16*. Varianta *rel32* se použije při nastavené velikosti 32 bitů nebo ve 32bitových segmentech. Výsledek se vždy ukládá do 32bitového registru EIP. Používá-li se 16bitová varianta, vynuluje se horních 16 bitů.

Varianty instrukcí CALL *r/m16* a *r/m32* jsou nepřímými voláními podprogramu. Operand specifikuje buď registr, nebo místo v paměti s absolutním offsetem cílové adresy. Mezi *r/m16* a *r/m32* se opět vybírá podle nastavené velikosti operandu.

V obou typech blízkých volání se offset první slabiky instrukce následující po CALL ukládá do zásobníku operací Push. Při použití 16bitového operandu se ukládá 16bitově a při použití 32bitového operandu se ukládá 32bitově. Ze zásobníku jej

operací Pop vybírá instrukce blízkého návratu z podprogramu RET. Instrukce RET musí být stejného typu, jako byla instrukce CALL (tj. blízké či vzdálené volání a 16 či 32bitová).

Instrukce s operandy *ptr16:16* a *ptr16:32* mají v instrukci uloženy 4 nebo 6 slabik segmentu a offsetu cílového místa. Jde tedy o vzdálená volání. Varianty *m16:16* a *m16:32* jsou vzdálená nepřímá volání. Adresa cílového místa je uložena na 4 nebo 6 slabikách v paměti na adrese, na kterou ukazuje operand. V reálném a V86 režimu obsahují cílové adresy vždy 16 bitů nové hodnoty registru CS a 16 nebo 32 bitů nové hodnoty registru EIP. V chráněném režimu instrukce CALL vždy analyzuje slabiku AR z popisovače, na který ukazuje index selektoru v prvních 16 bitech cílové adresy. Může totiž jít o vzdálené volání na stejné nebo vyšší úrovni oprávnění nebo o přepnutí procesu.

Vzdálená volání z 32bitového instrukčního segmentu do 16bitového instrukčního segmentu se mohou provádět pouze z prvních 64 KB 32bitového segmentu. Důvod je ten, že návratová adresa se uloží 16bitová.

Nastavované příznaky

V případě, že instrukce přepíná proces, mění se obsah všech příznaků. V jiných případech zůstávají příznaky beze změny.

Výjimky ve chráněném režimu

Vzdálená volání generují výjimky #GP, #NP, #SS a #TS tak, jak to bylo popsáno výše.

Blízká přímá volání generují #GP(0) tehdy, pokud cílové místo leží vně limitu instrukčního segmentu. Pokud uložením návratové adresy do zásobníku se překročí jeho limit, generuje se #SS(0), při výpadku stránky se generuje #PF, při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Blízká nepřímá volání generují tyto výjimky: Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). #GP se generuje tehdy, leží-li nepřímý offset vně limitu segmentu. Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

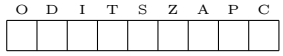
Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

RET *Return From Procedure*



Popis:

Instrukce RET vrací řízení z podprogramu volaného instrukcí CALL. Návratovou adresu převezme z vrcholu zásobníku.

Instrukce	Popis
RET	Blízký návrat
RET	Vzdálený návrat na stejnou úroveň oprávnění
RET	Vzdálený návrat na nižší úroveň oprávnění
RET *imm16*	Blízký návrat s odstraněním parametrů
RET *imm16*	Vzdálený návrat na stejnou úroveň oprávnění s odstraněním parametrů ze zásobníku
RET *imm16*	Vzdálený návrat na nižší úroveň oprávnění s odstraněním parametrů ze zásobníku

Operace

```
IF instrukce blízký návrat
THEN
  IF velikost operandu = 16
  THEN
    Pop(IP);
    EIP := EIP AND 0000FFFFh;
  ELSE  (* velikost operandu = 32 *)
    Push(EIP);
  FI;
  IF instrukce má operand imm16 THEN eSP := eSP + imm16; FI;
FI;

IF (PE=0 OR (PE=1 AND VM=1)) (* reálný nebo V86 režim *)
   AND instrukce je vzdálený návrat
THEN
  IF velikost operandu = 16
  THEN
```

```
    Pop(IP);
    EIP := EIP AND 0000FFFFh;
    Pop(CS);  (* 16bitový Pop *)
  ELSE (* velikost operandu = 32 *)
    Pop(EIP);
    Pop(CS); (* 32bitový Pop, horních 16 bitů nemá význam *)
  FI;
  IF instrukce má operand imm16 THEN eSP := eSP + imm16; FI;
FI;

IF (PE=1 AND VM=0)  (* Chráněný režim a ne V86 režim *)
   AND instrukce je vzdálený návrat
THEN
  IF velikost operandu = 32
  THEN třetí slovo v zásobníku musí být uvnitř limitu, jinak #SS(0);
  ELSE druhé slovo v zásobníku musí být uvnitř limitu, jinak #SS(0);
  FI;
  RPL návratového selektoru >= CPL, jinak #GP(návratový selektor);
  IF RPL návratového selektoru = CPL
  THEN GOTO STEJNÁ-ÚROVEŇ-OPRÁVNĚNÍ;
  ELSE GOTO JINÁ-ÚROVEŇ-OPRÁVNĚNÍ;
  FI;
FI;

STEJNÁ-ÚROVEŇ-OPRÁVNĚNÍ:
  návratový selektor není neplatný, jinak #GP(0);
  index návratového selektoru je uvnitř limitu, jinak #GP(selektor);
  slabika AR označuje instrukční segment, jinak #GP(selektor);
  IF není přizpůsobitelný
  THEN DPL instrukčního segmentu = CPL, jinak #GP(selektor);
  FI;
  IF je přizpůsobitelný
  THEN DPL instrukčního segmentu <= CPL, jinak #GP(selektor);
  FI;
  instrukční segment je přítomný, jinak #NP(selektor);
  slovo na vrcholu zásobníku je uvnitř limitu zásobníku, jinak #SS(0);
  IP ukazuje dovnitř limitu instrukčního segmentu, jinak #GP(0);
  IF velikost operandu = 32
  THEN
    naplň CS:EIP ze zásobníku;
    naplň neviditelnou část CS z popisovače;
    zvyš eSP o 8 + příp. imm16;
  ELSE (* velikost operandu = 16 *)
```

```
    naplň CS:IP ze zásobníku;
    naplň neviditelnou část CS z popisovače;
    zvyš eSP o 4 + příp. imm16;
  FI;

JINÁ-ÚROVEŇ-OPRÁVNĚNÍ:
  IF velikost operandu = 32
  THEN horních (16+imm16) slabik je uvnitř limitu zásobníku, jinak #SS(0);
  ELSE horních (8+imm16) slabik je uvnitř limitu zásobníku, jinak #SS(0);
  FI;
  kontrola návratového selektoru CS a jeho popisovače:
    selektor není neplatný, jinak #GP(0);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
        jinak #GP(selektor);
    slabika AR indikuje instrukční segment, jinak #GP(selektor);
    IF segment je nepřizpůsobivý
    THEN DPL instrukčního segmentu = RPL návratového selektoru,
        jinak #GP(selektor);
    FI;
    IF segment je přizpůsobivý
    THEN DPL instrukčního segmentu <= RPL návratového selektoru,
        jinak #GP(selektor);
    FI;
    segment je přítomný, jinak #NP(selektor);
  kontrola návratového selektoru SS a jeho popisovače:
    selektor není neplatný, jinak #GP(0);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
        jinak #GP(selektor);
    RPL selektoru = RPL návratového CS, jinak #GP(selektor);
    slabika AR indikuje zapisovatelný datový segment,
        jinak #GP(selektor);
    DPL popisovače = RPL návratového CS, jinak #GP(selektor);
    segment je přítomný, jinak #NP(selektor);
  IP ukazuje dovnitř limitu instrukčního segmentu, jinak #GP(0);
  CPL := RPL návratového selektoru CS;
  IF velikost operandu = 32
  THEN
    naplň CS:EIP ze zásobníku;
    nastav RPL z CS na CPL;
    zvyš eSP o 8+imm16;
    naplň SS:eSP ze zásobníku;
  ELSE (* velikost operandu = 16 *)
    naplň CS:IP ze zásobníku;
```

```
  nastav RPL z CS na CPL;
  zvyš eSP o 4+imm16;
  naplň SS:eSP ze zásobníku;
FI;
naplň neviditelnou část CS obsahem popisovače;
naplň neviditelnou část SS obsahem popisovače;
pro každý z registrů ES, FS, GS a DS
DO
  obsah registru je správný pro tuto úroveň oprávnění,
       jinak naplň registr neplatným selektorem (selektor := AR := 0);
  obsah registru je správný když:
    index selektoru ukazuje dovnitř limitu tabulky popisovačů;
    slabika AR indikuje datový nebo čitelný instrukční segment;
    IF segment je datový nebo nepřizpůsobivý
    THEN DPL >= CPL nebo DPL >= RPL;
    FI;
OD;
```

Komentář

Z podprogramu volaného blízkým voláním CALL (v zásobníku je uložena jenom offsetová část návratové adresy) se návrat musí provést pouze instrukcí blízkého návratu RET. Registr CS se provedením instrukce RET nezmění. Rovněž nelze kombinovat 32bitové blízké volání s 16bitovým blízkým návratem a obráceně.

Z podprogramu volaného vzdáleným voláním CALL (v zásobníku je uložen jak segment, tak i offset návratové adresy) se návrat musí provést pouze instrukcí vzdáleného návratu RET (ze zásobníku převezme nejprve offsetovou, pak segmentovou část návratové adresy), která návratovou adresou naplní registry CS:EIP. Rovněž je nutné zachovat kompatibilitu typů (16bitové vzdálené volání s 16bitovým vzdáleným návratem a 32bitové vzdálené volání s 32bitovým vzdáleným návratem).

Za instrukcí RET se může jako přímá hodnota uvést číslo, které udává, kolik slabik (je-li nastavená velikost operandu 16 bitů) nebo slov (je-li nastavená velikost operandu 32 bitů) ze zásobníku se má po vyzvednutí návratové adresy dodatečně odstranit. Této funkce se většinou využívá k odstranění parametrů předávaných podprogramu pomocí zásobníku. Odstranění se provede přičtením k registru eSP.

V chráněném režimu vyvolá instrukce vzdáleného návratu vždy kontrolu obsahu slabiky AR popisovače segmentu, který je návratovou instrukcí adresován. Vracet se lze pouze do instrukčního segmentu, jehož úroveň oprávnění je stejná nebo menší. Návrat do instrukčního segmentu s nižší úrovní oprávnění znamená také výměnu zásobníku a změnu obsahu segmentových registrů DS, ES, FS a GS tehdy, ukazují-li po výměně na segmenty, které již nemohou být adresovány. Potom jsou tyto segmentové registry naplněny neplatným selektorem.

Výjimky ve chráněném režimu

Instrukce generuje výjimky #GP, #NP, #SS a #TS tak, jak to bylo popsáno výše, při výpadku stránky se generuje #PF.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky.

BOUND — *Check Array Index Against Bounds*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce BOUND testuje, zda je hodnota (nejčastěji index pole) v zadaných mezích. Hodnota je číslo se znaménkem.

Instrukce	Popis
BOUND *r16,m16&16*	Kontrola, zda *r16* je v mezích
BOUND *r32,m32&32*	Kontrola, zda *r32* je v mezích

Operace

```
IF (první_operand < [druhý_operand]) OR
        (první_operand > ([druhý_operand + velikost_operandu/8]
THEN výjimka 5;
FI;
```

Komentář

Druhým operandem instrukce je adresa struktury v paměti tvořené dvěma slovy nebo dvojslovy (podle nastavení atributu velikosti operandu). První představuje dolní mez a druhé horní mez. První operand musí být větší nebo roven dolní mezi a současně menší nebo roven horní mezi. Pokud první operand není uvnitř těchto mezí, generuje se výjimka 5 s tím, že se do zásobníku ukládá návratová adresa ukazující na instrukci BOUND, která přerušení způsobila.

Bývá zvykem ukládat datovou strukturu s limity bezprostředně před vlastní pole s tím, že datová struktura s limity se adresuje konstantou vztaženou k začátku pole.

Výjimky ve chráněném režimu

Je-li první operand mimo meze, generuje se výjimka 5. Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Druhý operand musí být paměťový operand. Pokud procesor rozpozná, že instrukce je zakódována se slabikou ModR/M, tj. s registrem jako druhým operandem, generuje se #UD.

Výjimky v reálném režimu

Je-li operand mimo meze, generuje se výjimka 5. Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Je-li druhý operand registr, generuje se výjimka 6.

Výjimky v režimu V86

Stejné výjimky jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce je k dispozici od procesoru Intel286.

ENTER *Make Stack Frame for Procedure Parameters*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce ENTER podporuje předávání parametrů do volaného podprogramu a ukládání lokálních proměnných podprogramu do zásobníku. Má dva parametry. První parametr *lokální_prostor* je počet slabik, které se v zásobníku vyhradí pro uložení lokálních dat. Druhý parametr *úroveň_vnoření* určuje míru statického vnoření volané procedury vzhledem k hlavnímu programu (je to počet ukazatelů do lokálních prostorů nadřazených modulů, pomocí kterých jsou podprogramu tyto prostory zpřístupněny). Hodnota parametru *úroveň_vnoření* musí být v intervalu 0 až 31.

Instrukce	Popis
ENTER *imm16,imm8*	vytvoř oblast pro předání parametrů

Operace

```
úroveň_vnoření := úroveň_vnoření MOD 32;
IF velikost operandu = 16 THEN Push(BP); ELSE Push(EBP); FI;
IF velikost položky zásobníku = 16
THEN ukazatel_rámce = SP;
ELSE ukazatel_rámce = ESP;
FI;
FOR i:=1 TO (úroveň_vnoření - 1)
DO
  IF velikost operandu = 16
  THEN
    IF velikost položky zásobníku = 16
    THEN
      BP := BP - 2;
      Push([BP]); (* uložení slova *)
    ELSE (* velikost položky zásobníku = 32 *)
      EBP := EBP - 2;
      Push([EBP]); (* uložení slova *)
    FI;
  ELSE (* velikost operandu = 32 *)
    IF velikost položky zásobníku = 16
    THEN
      BP := BP - 4;
      Push([BP]); (* uložení dvojslova *)
    ELSE (* velikost položky zásobníku = 32 *)
      EBP := EBP - 4;
      Push([EBP]); (* uložení dvojslova *)
    FI;
  FI;
OD;
IF velikost položky zásobníku = 16
THEN Push(ukazatel_rámce); (* uložení slova *)
THEN Push(ukazatel_rámce); (* uložení dvojslova *)
FI;
IF velikost položky zásobníku = 16
THEN
  BP := ukazatel_rámce;
  SP := SP - lokální_prostor;
ELSE
  EBP := ukazatel_rámce;
  ESP := SP - lokální_prostor;
FI;
```

Komentář

Instrukce ENTER mění obsah registrů BP, SP a obsah zásobníku v těchto krocích:

1. Do zásobníku uloží obsah registru BP (tj. provede PUSH BP). Předpokládá se, že BP v tomto okamžiku ukazuje na začátek lokální datové struktury volajícího modulu.
2. Zapamatuje si momentální obsah registru SP.
3. Do zásobníku vloží *úroveň_vnoření* slov získaných od adresy, na kterou ukazuje obsah BP. Předpokládá se, že se přenáší *úroveň_vnoření* ukazatelů do lokálních prostorů nadřazených modulů.
4. Registr BP naplní zapamatovaným obsahem SP (z bodu 2).
5. V zásobníku vyhradí *lokální_prostor* slabik pro uložení lokálních proměnných tohoto podprogramu. Vyhrazení se provede tak, že se SP sníží o hodnotu *lokální_prostor*.

Odstranění struktur definovaných v zásobníku instrukcí ENTER lze provést instrukcí LEAVE.

Použití ENTER je ukázáno na tomto příkladu (viz obr. 12.6): z hlavního modulu je volán podprogram A a z podprogramu A je volán podprogram B. Podprogram A má svůj lokální prostor velikosti x slabik a B velikosti y slabik. Každému z volaných podprogramů se navíc předávají pomocí zásobníku parametry. Na obrázku je popsána posloupnost jen těch instrukcí, které modifikují obsah zásobníku. Postupně se měnící obsahy registru BP jsou očíslovány horním indexem. Zásobník je na obrázku plněn směrem shora dolů.

Výjimky ve chráněném režimu

#SS(0) se generuje v okamžiku, kdyby obsah SP nebo ESP překročil limit zásobníku. Při výpadku stránky je #PF(výpadek).

Výjimky v reálném režimu

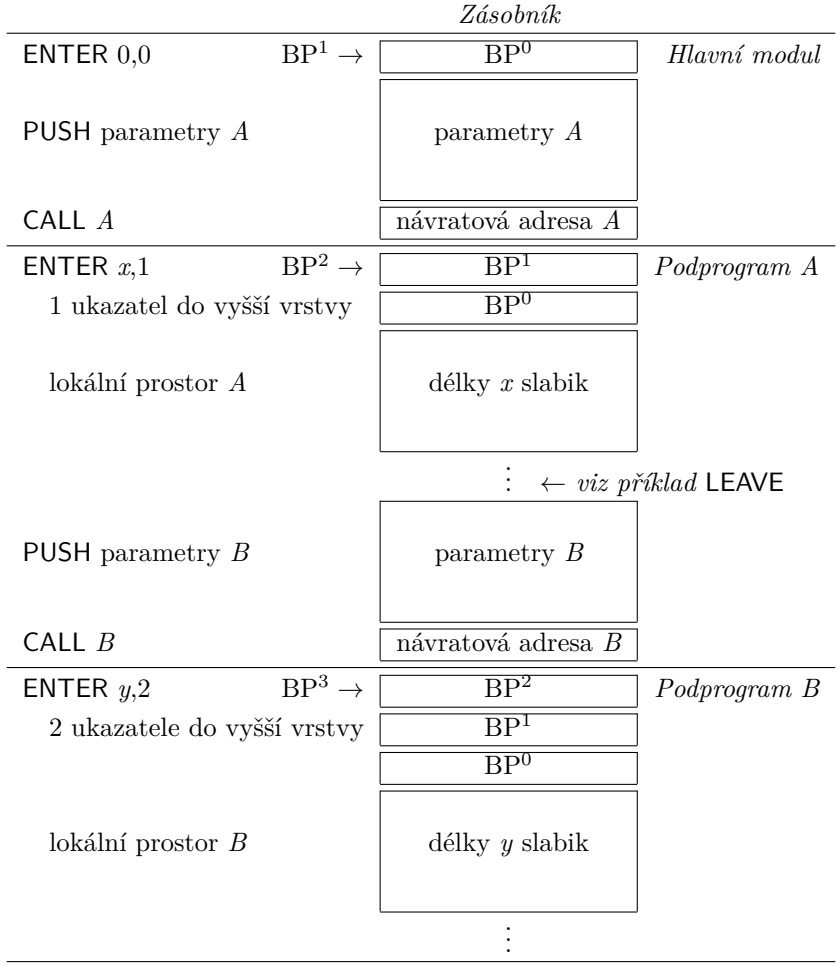
Žádné.

Výjimky v režimu V86

Žádné.

Poznámky k předchozím procesorům

Instrukce je k dispozici od procesoru Intel286.



Obr. 12.6 Příklad použití instrukce ENTER

LEAVE

High Level Procedure Exit

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce LEAVE odstraní v zásobníku strukturu definovanou instrukcí ENTER. Bude pracovat správně jen tehdy, nebyl-li změněn obsah registru BP.

Instrukce	Popis
LEAVE	SP := BP; Pop(BP)

Operace

```
IF velikost položky zásobníku = 16
THEN
  SP := BP;
ELSE (* velikost položky zásobníku = 16 *)
  ESP := EBP;
FI;
IF velikost operandu = 16
THEN
  BP := Pop();
ELSE (* velikost operandu = 32 *)
  EBP := Pop();
FI;
```

Komentář

Předpokládá se, že za instrukcí LEAVE bezprostředně následuje instrukce RET *n*, která ze zásobníku odstraní návratovou adresu a *n* slabik parametrů předávaných podprogramu. Např. bylo-li na obr. 12.6 podprogramu *B* předáno 10 slabik parametrů, potom posloupnost instrukcí:

```
LEAVE
RET    10
```

provede návrat z podprogramu *B* a zásobník převede do úrovně označené: „← *viz příklad* LEAVE“.

Výjimky ve chráněném režimu

#SS(0) se generuje v okamžiku, kdyby obsah SP nebo ESP překročil limit zásobníku. Při výpadku stránky je #PF(výpadek).

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

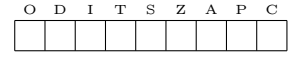
Stejná výjimka jako v reálném režimu.

Poznámky k předchozím procesorům

Instrukce je k dispozici od procesoru Intel286.

12.9 Instrukce pro manipulace s příznakovým registrem

PUSHF — *Push Flag Register onto the Stack*
PUSHFD — *Push Extended Flag Register onto the Stack*



Popis:

Instrukce PUSHF ukládá na vrchol zásobníku registr příznaků FLAGS (viz též instrukci PUSH a tvar registru příznaků na str. 24). Instrukce PUSHFD ukládá 32bitový registr EFLAGS.

Instrukce	Popis
PUSHF	Ulož registr FLAGS do zásobníku
PUSHFD	Ulož registr EFLAGS do zásobníku

Operace

```
IF VM=0 (* není zapnut režim V86 *)
THEN
  IF nastavená velikost operandu = 32
  THEN Push(EFLAGS AND 0FCFFFFh); (* vynuluj bity VM a RF z EFLAGS *)
  ELSE Push(FLAGS);
  FI;
ELSE (* je zapnut režim V86 *)
  IF IOPL=3
  THEN
    IF nastavená velikost operandu = 32
    THEN Push(EFLAGS AND 0FCFFFFh); (* vynuluj bity VM a RF z EFLAGS *)
    ELSE Push(FLAGS);
```

```
    FI;
  ELSE
    #GP(0); (* přerušení aktivující monitor V86 procesu *)
  FI;
FI;
```

Komentář

Instrukce PUSHF sníží ukazatel vrcholu zásobníku o 2 a zkopíruje obsah registru FLAGS na vrchol zásobníku. Instrukce PUSHFD sníží ukazatel vrcholu zásobníku o 4 a zkopíruje obsah registru EFLAGS na vrchol zásobníku.

Výjimky ve chráněném režimu

Pokud nová hodnota SP nebo ESP ukazuje mimo rozsah segmentu se zásobníkem, generuje se #SS(0).

Výjimky v reálném režimu

Žádné. Pokud je SP nebo ESP rovno 1, procesor se zastaví.

Výjimky v režimu V86

Je-li IOPL < 3, generuje se #GP(0). Přerušením je monitoru V86 procesu ponecháno rozhodnutí o provedení či neprovedení operace.

POPF
POPFD

Pop from the Stack into the Flag Register
Pop from the Stack into the Extended Flag Register

Popis:

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Instrukce POP vybere z vrcholu zásobníku hodnotu a uloží ji do registru příznaků (viz též instrukci POP a tvar registru příznaků na str. 24). Instrukce POPFD hodnotu z vrcholu zásobníku ukládá do EFLAGS.

Instrukce	Popis
POPF	Naplň registr FLAGS ze zásobníku
POPFD	Naplň registr EFLAGS ze zásobníku

Operace

```
IF VM=0 (* není zapnut režim V86 *)
THEN
  IF nastavená velikost operandu = 32
  THEN EFLAGS := Pop() AND 277FD7h;
  ELSE FLAGS := Pop();
  FI;
ELSE (* je zapnut režim V86 *)
  IF IOPL=3
  THEN
    IF nastavená velikost operandu = 32
    THEN
      Pomocná := Pop();
      EFLAGS := (EFLAGS AND 1B3000h) OR (Pomocná AND 0FFE4CFFFh);
                  (* bity VM, RF, IOPL, VIP a VIF registru EFLAGS
                     nejsou instrukcí POPFD měněny *)
    ELSE
      FLAGS := Pop();
    FI;
  ELSE
    #GP(0); (* přerušení aktivující monitor V86 procesu *)
  FI;
FI;
```

Komentář

Instrukce POPF zkopíruje slovo z vrcholu zásobníku do registru FLAGS a zvýší ukazatel vrcholu zásobníku o 2. Instrukce POPFD zkopíruje dvojslovo z vrcholu zásobníku do registru EFLAGS a zvýší ukazatel vrcholu zásobníku o 4.

Instrukce hodnotu IOPL v registru příznaků mění pouze při úrovni oprávnění. Hodnota příznaku IF (povolení přerušení) se instrukcí mění jenom při úrovni oprávnění alespoň takové, jaká je v IOPL (v reálném režimu se vše provádí jako na úrovni oprávnění 0). Pokus o změnu hodnoty bitů IOPL a IF nevyvolá generování výjimky – změna uvedených bitů se pouze neprovede.

Nastavované příznaky

Všechny vyjma VM, RF, IOPL, VIF a VIP.

Výjimky ve chráněném režimu

Pokud nová hodnota SP nebo ESP ukazuje mimo rozsah segmentu se zásobníkem, generuje se #SS(0).

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

Je-li IOPL < 3, generuje se #GP(0). Přerušením je monitoru V86 procesu ponecháno rozhodnutí o provedení či neprovedení operace. #GP(0) se generuje rovněž při pokusu o provedení POPF s prefixem měnícím velikost operandu.

STC
STD

Set Carry Flag
Set Direction Flag

O	D	I	T	S	Z	A	P	C
	1							1

Popis:

Instrukce nastavují v registru FLAGS a EFLAGS příznaky CF (STC) a DF (STD) na jedničku.

Instrukce	Popis
STC	CF := 1 (nastavení příznaku přenosu)
STD	DF := 1 (zpracování řetězců odzadu)

Operace

```
CF := 1;    nebo    DF := 1;
```

Komentář

Instrukce negenerují žádné výjimky.

CLC
CLD

Clear Carry Flag
Clear Direction Flag

O	D	I	T	S	Z	A	P	C
	0							0

Popis:

Instrukce nulují v registru FLAGS a EFLAGS příznaky CF (STC) a DF (STD).

Instrukce	Popis
CLC	CF := 0 (nulování příznaku přenosu)
CLD	DF := 0 (zpracování řetězců zepředu)

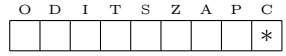
Operace

```
CF := 0;    nebo    DF := 0;
```

Komentář

Instrukce negenerují žádné výjimky.

CMC *Complement Carry Flag*



Popis:

Instrukce CMC invertuje obsah bitu CF v registru příznaků FLAGS nebo EFLAGS. Pokud byla hodnota CF=1, pak nastaví CF=0, a naopak.

Instrukce	Popis
CMC	CF := NOT CF (inverze příznaku přenosu)

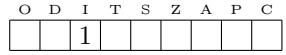
Operace

```
CF := NOT CF;
```

Komentář

Instrukce negeneruje žádné výjimky.

STI *Set Interrupt Flag*



Popis:

Instrukce nastavuje v registru FLAGS a EFLAGS na jedničku příznak IF (povolení maskovatelného přerušení).

Instrukce	Popis
STI	IF := 1

Operace

```
IF PE = 0  (* provádění v reálném režimu *)
THEN
  IF := 1; (* nastavení příznaku povolení přerušení *)
ELSE (* provádění v chráněném režimu *)
  IF VM = 0 (* není zapnut režim V86 *)
  THEN
    IF CPL <= IOPL
    THEN
      IF := 1; (* nastavení příznaku povolení přerušení *)
    ELSE
      #GP(0);
    FI;
  ELSE (* je zapnut režim V86 v rámci chráněného režimu *)
    IF IOPL=3
    THEN
      IF := 1;
    ELSE
      #GP(0); (* výjimka na aktivaci monitoru V86 procesu *)
    FI;
  FI;
FI;
```

Komentář

Následující tabulka shrnuje podmínky a činnost instrukce STI. Políčko obsahující „-“ znamená, že na hodnotě nezáleží; „*ano*“ znamená, že se akce provede, a prázdné políčko znamená, že se akce neprovede.

Podmínky:	PE =	0	1	1	1
	VM =	-	0	0	1
	CPL	-	≤IOPL	>IOPL	-
	IOPL	-	-	-	=3
Akce:	IF := 1	*ano*	*ano*		*ano*
	#GP(0)			*ano*	

Instrukce STI nastavuje příznak IF. Procesor po dokončení **následující** instrukce (pokud tato následující instrukce ponechá příznak IF jedničkový) povoluje (tj. reaguje) na vnější maskovatelná přerušení.

Přerušení se povoluje až po provedení následující instrukce, protože se předpokládá, že za instrukcí STI následuje RET. Přerušení se v tomto případě uplatní až po návratu z podprogramu. Jeden z důvodů je šetření kapacity zásobníku. Pokud po instrukci STI následuje bezprostředně CLI, přerušení se nepovolí.

Instrukce nemá vliv na NMI a vnitřní přerušení a výjimky. Tzn. nastane-li NMI nebo vnitřní přerušení během instrukce STI, uplatní se ihned po jejím dokončení.

Výjimky ve chráněném režimu

Je-li CPL>IOPL, generuje se #GP(0).

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

#GP(0) při IOPL<3.

CLI *Clear Interrupt Flag*

O	D	I	T	S	Z	A	P	C
		0						

Popis:

Instrukce nuluje v registru FLAGS a EFLAGS příznak IF (zakázání maskovatelného přerušení).

Instrukce	Popis
CLI	IF := 0

Operace

```
IF PE = 0  (* provádění v reálném režimu *)
THEN
  IF := 0; (* nulování příznaku povolení přerušení *)
ELSE (* provádění v chráněném režimu *)
  IF VM = 0 (* není zapnut režim V86 *)
  THEN
    IF CPL <= IOPL
    THEN
      IF := 0; (* nulování příznaku povolení přerušení *)
    ELSE
      #GP(0);
    FI;
  ELSE (* je zapnut režim V86 v rámci chráněného režimu *)
    IF IOPL=3
    THEN
      IF := 0;
```

```
    ELSE
      #GP(0); (* výjimka na aktivaci monitoru V86 procesu *)
    FI;
  FI;
FI;
```

Komentář

Následující tabulka shrnuje podmínky a činnost instrukce CLI. Políčko obsahující „-“ znamená, že na hodnotě nezáleží; „*ano*“ znamená, že se akce provede a prázdné políčko znamená, že se akce neprovede.

Podmínky:	PE =	0	1	1	1
	VM =	-	0	0	1
	CPL	-	≤IOPL	>IOPL	-
	IOPL	-	-	-	=3
Akce:	IF := 1	*ano*	*ano*		
	#GP(0)			*ano*	*ano*

Instrukce CLI nuluje příznak IF, pokud současná úroveň oprávnění je alespoň taková, jaká je nastavena v IOPL. Procesor přestává reagovat na vnější maskovatelná přerušení ihned po nastavení příznaku IF na nulu, tj. zhruba na konci této instrukce. Přerušení lze povolit instrukcí STI.

Výjimky ve chráněném režimu

Je-li CPL>IOPL, generuje se #GP(0).

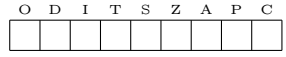
Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

#GP(0) při IOPL<3.

LAHF *Load Flags into AH Register*



Popis:

Instrukce LAHF kopíruje nižších 8 bitů příznakového registru do registru AH (viz tvar registru příznaků na str. 24).

Instrukce	Popis
LAHF	AH := po bitech: SF ZF xx AF xx PF xx CF

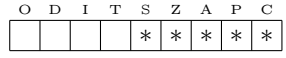
Operace

```
AH := SF ZF xx AF xx PF xx CF
```

Komentář

Příznak SF se ukládá do bitu nejvyššího řádu a CF do nejnižšího. Hodnota xx je nedefinována. Instrukce negeneruje žádné výjimky.

SAHF *Store AH Register into Flags Register*



Popis:

Instrukce SAHF naplní dolních 8 bitů příznakového registru hodnotou uloženou v registru AH (viz tvar registru příznaků na str. 24).

Instrukce	Popis
SAHF	SF ZF xx AF xx PF xx CF := AH

Operace

```
SF ZF xx AF xx PF xx CF := AH;
```

Komentář

Příznak SF se plní bitem nejvyššího řádu a CF nejnižším. Bity odpovídající pozicím xx se ignorují. Instrukce negeneruje žádné výjimky.

12.10 Přerušovací systém

INT *Call to Interrupt Procedure*

INTO *Call to Interrupt Procedure if Overflow*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce INT generuje programové přerušení. Jednoslabiková přímá hodnota v operandu instrukce (v intervalu 0 až 255) je v reálném režimu použita jako index do tabulky přerušovacích vektorů (viz obr. 4.2 na str. 45), kde je uložena 4slabiková adresa rutiny, obsluhující přerušení odkazované operandem. V chráněném režimu je použita jako index do tabulky IDT (viz str. 91), kde může být 8slabikový popisovač tří typů: brána pro přepnutí procesu, brána pro maskující přerušení a brána pro nemaskující přerušení (Trap). V rámci této instrukce se do zásobníku uloží registr příznaků a návratová adresa, která ukazuje na instrukci následující za INT (v chráněném režimu příp. i další údaje). Vynulují se příznaky IF a TF a naplní se registry CS:eIP ukazatelem z přerušovacího vektoru nebo z popisovače (viz též str. 45). Návrat z přerušení se provádí instrukcí IRET.

Instrukce se někdy používá pro volání služeb operačního systému nebo do počítače vestavěného programového vybavení (jde o zvláštní formu vzdáleného volání podprogramu, z něhož se návrat provádí instrukcí IRET). Navíc, oproti instrukci vzdáleného volání se instrukcí INT do zásobníku ukládá registr příznaků.

Instrukce INTO je jednoslabiková instrukce (operační znak 0CEh) generující přerušení číslo 4, pokud je nastaven příznak OF v příznakovém registru. Instrukce využívá programové vybavení pro jednoduché testování přeplnění po provedení aritmetických operací.

Další variantou je opět jednoslabiková instrukce INT 3 s operačním znakem 0CCh. Tato se používá pro aktivaci ladicího systému (ladicí bod – Breakpoint). Ostatní instrukce INT *n* jsou dvouslabikové.

Instrukce	Popis
INT 3	přerušení 3 - trap pro ladicí program
INT 3	přerušení 3 - chráněný režim: na stejnou úroveň
INT 3	přerušení 3 - chráněný režim: na vyšší úroveň
INT 3	přerušení 3 - z V86 na úroveň 0
INT 3	přerušení 3 - chráněný režim: přes bránu přepínání procesů
INT *imm8*	číslo přerušení zadáno přímou hodnotou
INT *imm8*	přerušení - chráněný režim: na stejnou úroveň
INT *imm8*	přerušení - chráněný režim: na vyšší úroveň
INT *imm8*	přerušení - z V86 na úroveň 0

INT *imm8*	přerušení - chráněný režim: přes bránu přepínání procesů
INTO	přerušení 4 - pokud je OF=1
INTO	přerušení 4 - chráněný režim: na stejnou úroveň
INTO	přerušení 4 - chráněný režim: na vyšší úroveň
INTO	přerušení 4 - z V86 na úroveň 0
INTO	přerušení 4 - chráněný režim: přes bránu přepínání procesů

Operace

Následující algoritmy platí jak pro výše uvedené instrukce, tak i pro vnější přerušení a výjimky.

```
IF PE = 0  (* jsme v reálném režimu *)
THEN
  CALL REÁLNÝ-REŽIM;
ELSE
  CALL CHRÁNĚNÝ-REŽIM;
  IF přerušení ukazuje na bránu pro přepnutí procesu
  THEN
    CALL BRÁNA-PRO-PŘEPNUTÍ-PROCESU;
  ELSE
    CALL BRÁNA-PRO-PŘERUŠENÍ-NEBO-TRAP;
    IF instrukční segment je nepřizpůsobivý AND DPL<CPL
    THEN
      IF VM=0
      THEN CALL NA-VYŠŠÍ-ÚROVEŇ-OPRÁVNĚNÍ;
      ELSE CALL Z-REŽIMU-V86;
      FI;
    ELSE (* PE=1, brána přerušení/trap, DPL >= CPL *)
      IF instrukční segment je přizpůsobitelný OR DPL=CPL
      THEN CALL NA-STEJNOU-ÚROVEŇ-OPRÁVNĚNÍ;
      ELSE #GP(selektor CS + EXT); (* PE=1, brána přerušení/trap, DPL>CPL *)
      FI;
    FI;
  FI;
FI;
END;

REÁLNÝ-REŽIM PROC
  Push(FLAGS);
  IF := 0; (* zákaz dalších přerušení *)
  TF := 0; (* vypnutí trasování *)
  Push(CS);
  Push(IP);
```

```
  (* do zásobníku se neukládají chybové kódy *)
  CS := Tabulka přerušovacích vektorů[číslo přerušení * 4].selektor;
  IP := Tabulka přerušovacích vektorů[číslo přerušení * 4].offset;
REÁLNÝ-REŽIM ENDPROC

CHRÁNĚNÝ-REŽIM PROC
  číslo přerušení ukazuje dovnitř limitu IDT,
         jinak #GP(číslo přerušení * 8 + 2 + EXT);
  slabika AR indikuje bránu pro přerušení, bránu pro trap nebo bránu
         pro přepínání procesů, jinak #GP(číslo přerušení * 8 + 2 + EXT);
  IF přerušení je programové (* tj. způsobené INT n, INT 3, INTO *)
  THEN
    IF DPL popisovače brány < CPL
    THEN
      #GP(číslo přerušení * 8 + 2 + EXT); (* PE=1, DPL<CPL, softw. přer. *)
    FI;
  FI;
  brána je označena jako přítomná, jinak #NP(číslo přerušení * 8 + 2 + EXT);
CHRÁNĚNÝ-REŽIM ENDPROC

BRÁNA-PRO-PŘEPNUTÍ-PROCESU PROC
  kontrola selektoru ukazujícího na TSS uloženého v bráně:
    je v GDT, jinak #TS(selektor TSS);
    index ukazuje dovnitř limitu GDT, jinak #TS(selektor TSS);
    slabika AR indikuje neaktivní TSS (nejnižší bity jsou 00001),
          jinak #TS(selektor TSS);
    TSS je přítomný, jinak #NP(selektor TSS);
  PŘEPNUTÍ PROCESŮ s vnořením;
  IF přerušení bylo způsobeno výpadkem s chybovým kódem
  THEN
    limit zásobníku dovoluje uložit o 2 slabiky více, jinak #SS(0);
    Push(chybový kód);
  FI;
  IP ukazuje dovnitř limitu segmentu CS, jinak #GP(0);
BRÁNA-PRO-PŘEPNUTÍ-PROCESU ENDPROC

BRÁNA-PRO-PŘERUŠENÍ-NEBO-TRAP PROC
  kontrola selektoru a popisovače CS z popisovače brány;
    selektor je platný, jinak #GP(EXT);
    selektor ukazuje dovnitř limitu tabulky popisovačů,
         jinak #GP(selektor+EXT);
    slabika AR popisovače indikuje instrukční segment,
         jinak #GP(selektor+EXT);
```

```
    segment je přítomný, jinak #NP(selektor+EXT);
BRÁNA-PRO-PŘERUŠENÍ-NEBO-TRAP ENDPROC

NA-VYŠŠÍ-ÚROVEŇ-OPRÁVNĚNÍ PROC
  (* PE=1 (chráněný režim), brána pro přerušení nebo brána pro trap
     ukazující na instrukční segment s DPL<CPL (na vyšší úrovni
     oprávnění) nepřizpůsobitelný, VM=0 (není V86 režim)  *)
  kontrola selektoru a popisovače zásobníku nové úrovně oprávnění
            v rámci tohoto TSS:
    selektor je platný, jinak #TS(EXT);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
            jinak #TS(selektor SS + EXT);
    RPL selektoru = DPL instrukčního segmentu, jinak #TS(selektor SS + EXT);
    DPL zásobníkového segmentu = DPL instrukčního segmentu,
            jinak #TS(selektor SS + EXT);
    popisovač ukazuje na zapisovatelný datový segment,
            jinak #TS(selektor SS + EXT);
    segment je přítomný, jinak #SS(selektor SS + EXT);
  IF brána je 32bitová
  THEN nový zásobník má prostor na 20 slabik, jinak #SS(0)
  ELSE nový zásobník má prostor na 10 slabik, jinak #SS(0)
  FI;
  IP ukazuje dovnitř limitu segmentu CS, jinak #GP(0);
  naplň SS a eSP hodnotami z TSS;
  IF brána je 32bitová
  THEN CS:EIP := selektor:offset z brány;
  ELSE CS:IP := selektor:offset z brány;
  FI;
  naplň nevidit. část CS popisovačem;
  naplň nevidit. část SS popisovačem;
  IF brána je 32bitová
  THEN
    Push(segment:offset starého zásobníku); (* 3 slova jsou doplněna na 4 *)
    Push(EFLAGS);
    Push(segment:offset návrat. adresy); (* 3 slova jsou doplněna na 4 *)
  ELSE
    Push(segment:offset starého zásobníku); (* 2 slova *)
    Push(FLAGS);
    Push(segment:offset návrat. adresy); (* 2 slova *)
  FI;
  CPL := DPL nového instrukčního segmentu;
  RPL z CS := CPL;
  IF brána pro přerušení THEN IF:=0; (* zákaz přerušení *) FI;
```

```
  TF := 0;
  NT := 0;
NA-VYŠŠÍ-ÚROVEŇ-OPRÁVNĚNÍ ENDPROC

Z-REŽIMU-V86 PROC
  (* PE=1 (chráněný režim), brána pro přerušení nebo brána pro trap
     ukazující na instrukční segment s DPL<CPL (na vyšší úrovni
     oprávnění) nepřizpůsobitelný, VM=1 (přerušení z režimu V86)  *)
  kontrola selektoru a popisovače zásobníku nové úrovně oprávnění
             v rámci tohoto TSS:
    selektor je platný, jinak #TS(EXT);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
             jinak #TS(selektor SS + EXT);
    RPL selektoru = DPL instrukčního segmentu, jinak #TS(selektor SS + EXT);
    DPL zásobníkového segmentu = DPL instrukčního segmentu,
             jinak #TS(selektor SS + EXT);
    popisovač ukazuje na zapisovatelný datový segment,
             jinak #TS(selektor SS + EXT);
    segment je přítomný, jinak #SS(selektor SS + EXT);
  IF brána je 32bitová
  THEN nový zásobník má prostor na 20 slabik, jinak #SS(0)
  ELSE nový zásobník má prostor na 10 slabik, jinak #SS(0)
  FI;
  IP ukazuje dovnitř limitu segmentu CS, jinak #GP(0);
  IF IOPL < 3
  THEN
    #GP(0); (* přerušení pro monitor procesu V86:
               PE=1, brána přerušení/trap, DPL<CPL, VM=1, IOPL<3 *)
  ELSE (* IOPL=3 *)
    IF DPL brány = 3
    THEN
      IF cílové CPL <> 0
      THEN #GP(0);
      ELSE
        PomocnýEFLAGS := EFLAGS;
        VM := 0;
        TF := 0;
        IF brána pro přerušení THEN IF := 0; FI;
        PomocnýSS := SS;
        PomocnýESP := ESP;
        SS := TSS.SS0; (* se změnou úrovně oprávnění na 0 změň i zásobník *)
        ESP := TSS.ESP0;
        Push(GS); (* doplněno na 2 slova *)
```

```
        Push(FS); (* doplněno na 2 slova *)
        Push(DS); (* doplněno na 2 slova *)
        Push(ES); (* doplněno na 2 slova *)
        GS := 0; (* naplnění neplatným selektorem *)
        FS := 0;
        DS := 0;
        ES := 0;
        Push(PomocnýSS); (* doplněno na 2 slova *)
        Push(PomocnýESP);
        Push(PomocnýEFLAGS);
        Push(CS); (* doplněno na 2 slova *)
        Push(EIP);
        CS:EIP := selektor:offset z brány;
      FI;
    ELSE (* DPL brány < 3 *)
      #GP(0);
    FI;
  FI;
Z-REŽIMU-V86 ENDPROC

NA-STEJNOU-ÚROVEŇ-OPRÁVNĚNÍ PROC
  (* PE=1 (chráněný režim), brána pro přerušení nebo brána pro trap
     ukazující na instrukční segment s DPL=CPL (na stejné úrovni
     oprávnění) nebo přizpůsobitelný  *)
  IF brána je 32bitová
  THEN nový zásobník má prostor na 10 slabik, jinak #SS(0)
  ELSE nový zásobník má prostor na 6 slabik, jinak #SS(0)
  FI;
  IF přerušení bylo způsobeno výjimkou s chybovým kódem
  THEN
    limit zásobníku dovoluje uložit o 2 slabiky více, jinak #SS(0);
  FI;
  IP ukazuje dovnitř limitu segmentu CS, jinak #GP(0);
  IF brána je 32bitová
  THEN
    Push(EFLAGS);
    Push(segment:offset návrat. adresy); (* 3 slova doplněna na 4 *)
    CS:EIP := selektor:offset z brány;
  ELSE (* 16bitová brána *)
    Push(FLAGS);
    Push(segment:offset návrat. adresy); (* 2 slova *)
    CS:EIP := selektor:offset z brány;
  FI;
```

```
  naplň nevidit. část CS popisovačem;
  RPL z CS := CPL;
  Push(chybový kód); (* je-li nějaký *)
  IF brána pro přerušení THEN IF:=0; FI;
  TF := 0;
  NT := 0;
NA-STEJNOU-ÚROVEŇ-OPRÁVNĚNÍ ENDPROC
```

Komentář

Tabulka na obr. 12.7 shrnuje, za jakých podmínek se provádějí akce popisované ve výše uvedeném algoritmu. V horní části tabulky jsou uvedeny podmínky, v dolní akce. Každé „*ano*“ v horní polovině znamená provedení procedury. Číslo, které je za „*ano*“ uvedeno, značí pořadí, v jakém se procedura provádí. Symbol „-“ označuje, že na hodnotě nezáleží; prázdné políčko sděluje, že se akce neprovádí.

Použijete-li instrukci INT n v režimu V86 (VM=1), testuje se hodnota IOPL. Je-li IOPL<3, generuje se výjimka porušení obecné ochrany. Je-li IOPL=3, DPL z brány pro přerušení je 3 a přerušení je směrováno na úroveň 0 (cílové CPL=0), potom se přerušení provede.

Firma Intel si vyhrazuje prvních 32 položek tabulky přerušovacích vektorů a IDT pro použití procesorem[2].

Výjimky ve chráněném režimu

Generují se #GP, #NP, #SS a #TS tak, jak bylo popsáno výše.

Výjimky v reálném režimu

Žádné. Pokud je před provedením INT nebo INTO SP rovno 1, 3 nebo 5, procesor se zastaví.

Výjimky v režimu V86

Je-li IOPL<3, generuje se #GP(0).

[2] Počítače kompatibilní s IBM PC toto z historických důvodů nedodržují.

PE =	0	1	1	1	1	1	1	1
VM =	-	-	-	-	-	0	1	1
IOPL =	-	-	-	-	-	-	<3	=3
DPL a CPL	-	DPL< CPL	-	DPL< CPL	DPL= CPL or přizp.	DPL< CPL and nepřizp.	-	-
Typ přerušení	-	softw.	-	-	-	-	-	-
Typ brány	-	-	přepn. procesu	přeruš. trap	přeruš. trap	přeruš. trap	přeruš. trap	přeruš. trap
REÁLNÝ REŽIM	*ano*							
CHRÁNĚNÝ REŽIM		ano_1	ano_1	ano_1	ano_1	ano_1	ano_1	ano_1
BRÁNA PRO PŘEPNUTÍ PROCESU			ano_2					
BRÁNA PRO PŘERUŠENÍ NEBO TRAP				ano_2	ano_2	ano_2	ano_2	ano_2
NA VYŠŠÍ ÚROVEŇ OPRÁVNĚNÍ						ano_3		
Z REŽIMU V86								ano_3
NA STEJNOU ÚROVEŇ OPRÁVNĚNÍ					ano_3			
#GP		ano_2		ano_3			ano_3	

Obr. 12.7 Provádění instrukce INT

IRET *Interrupt Return*
IRETD *Interrupt Return*

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Popis:

Instrukce IRET provádí návrat z programové rutiny vyvolané přerušením nebo instrukcí INT. V reálném režimu z vrcholu zásobníku instrukce přebírá návratovou adresu CS:IP a obsah registru FLAGS (viz též str. 45), čímž se vrátí zpět do přerušeného programu.

V chráněném režimu závisí činnost instrukce IRET na nastavení příznaku NT (Nested Task) v příznakovém registru. Je-li nulový, instrukce se vrací do přerušeného modulu bez přepnutí procesu. Vracet se lze pouze do stejné nebo nižší úrovně

oprávnění. Pokud se vrací do nižší úrovně oprávnění, musí instrukce nastavit i nový obsah ukazatele vrcholu zásobníku a SS odpovídající nové úrovni oprávnění.

Pokud je příznak NT v chráněném režimu nastaven na jedničku, potom instrukce provádí návrat s přepnutím procesu. Procesu, ze kterého se vracíme, bylo předáno řízení instrukcí CALL nebo INT, přičemž tyto instrukce uložily do TSS návratový selektor. Stav procesu, který je instrukcí IRET opouštěn, se ukládá do TSS proto, aby bylo možné v něm později pokračovat.

Instrukce	Popis
IRET	návrat z přerušení: z reálného nebo V86 režimu
IRET	návrat z přerušení: vzdál. návrat a výběr příznaků
IRET	návrat z přerušení: do nižší úrovně
IRET	návrat z přerušení: do jiného procesu (NT=1)
IRETD	návrat z přerušení: vzdál. návrat a výběr příznaků
IRETD	návrat z přerušení: do nižší úrovně
IRETD	návrat z přerušení: do jiného procesu (NT=1)

Operace

```
IF PE = 0  (* jsme v reálném režimu *)
THEN GOTO REÁLNÝ-REŽIM;
ELSE GOTO CHRÁNĚNÝ-REŽIM;
FI;

REÁLNÝ-REŽIM:
  IF velikost operandu = 32 (* instrukce = IRETD *)
  THEN EIP := Pop(); (* 4 slabiky *)
  ELSE (* instrukce = IRET *)
    IP := Pop(); (* 2 slabiky *)
  FI;
  CS := Pop(); (* 2 slabiky *)
  IF velikost operandu = 32 (* instrukce = IRETD *)
  THEN
    Pop(); (* 2 slabiky doplňující CS na 4 *)
    EFLAGS := Pop(); (* 4 slabiky *)
  ELSE (* instrukce = IRET *)
    FLAGS := Pop(); (* 2 slabiky *)
  FI;
  END;

CHRÁNĚNÝ-REŽIM:
  IF VM=1
  THEN (* PE=1, VM=1: režim V86 v rámci chráněného režimu *)
```

```
    GOTO NÁVRAT-Z-V86;
  ELSE
    IF NT=1
    THEN (* PE=1, VM=0, NT=1: návrat s přepnutím procesu *)
      GOTO NÁVRAT-S-PŘEPNUTÍM-PROCESU;
    ELSE (* PE=1, VM=0, NT=0 *)
      IF VM v obrazu příznak. registru uloženém v zásobníku = 1
        THEN GOTO NÁVRAT-DO-V86;
        ELSE GOTO NÁVRAT;
      FI;
    FI;
  FI;

NÁVRAT-Z-V86:
  IF IOPL=3 (* PE=1, VM=1, IOPL=3 *)
  THEN
    IF velikost operandu = 16
    THEN
      IP := Pop(); (* 2 slabiky *)
      CS := Pop();
      FLAGS := Pop();
    ELSE (* velikost operandu = 32 *)
      EIP := Pop(); (* 4 slabiky *)
      CS := Pop();
      EFLAGS := Pop(); (* příznaky VM, IOPL, VIP a VIF instrukce nemění *)
    FI;
  ELSE #GP(0); (* trap do monitoru V86 procesu: PE=1, VM=1, IOPL<3 *)
  FI;
  END;

NÁVRAT-DO-V86: (* přerušeno bylo ve V86 režimu *)
  horních 36 slabik zásobníku je uvnitř limitu, jinak #SS(0);
  uložený eIP ukazuje dovnitř limitu instruk. segmentu, jinak #GP(0);
  EFLAGS := SS:[ESP+8];
  EIP := Pop();
  CS := Pop(); (* ve tvaru 8086, protože jdeme do V86 *)
  Pop(); (* odstraň již převzaté příznaky *)
  PomocnýESP := Pop();
  PomocnýSS := Pop();
  ES := Pop(); (* 4 slabiky, z nichž 2 horní jsou nevýznamné *)
  DS := Pop(); (* 4 slabiky, z nichž 2 horní jsou nevýznamné *)
  FS := Pop(); (* 4 slabiky, z nichž 2 horní jsou nevýznamné *)
  GS := Pop(); (* 4 slabiky, z nichž 2 horní jsou nevýznamné *)
```

```
  SS:ESP := PomocnýSS:PomocnýESP;
  (* Pokračuje běh ve V86 režimu *)
  END;

NÁVRAT-S-PŘEPNUTÍM-PROCESU: (* PE=1, VM=0, NT=1 *)
  Kontrola zpětného ukazatele (selektoru) uloženého v TSS, který je
            adresován právě používaným registrem TR:
    selektor ukazuje do GDT, jinak #TS(nový selektor TSS);
    index ukazuje dovnitř limitu GDT, jinak #TS(nový selektor TSS);
    slabika AR indikuje aktivní TSS, jinak #TS(nový selektor TSS);
    TSS je přítomný, jinak #NP(selektor TSS);
  PŘEPNUTÍ PROCESŮ bez vnoření;
  právě ukončený proces označ jako neaktivní;
  instrukční registr ukazuje dovnitř limitu segmentu CS, jinak #GP(0);
  END;

NÁVRAT: (* PE=1, VM=0, NT=1, nevrací se do V86 *)
  IF velikost operandu = 32
  THEN třetí slovo v zásobníku ještě patří do limitu, jinak #SS(0);
  ELSE druhé slovo v zásobníku ještě patří do limitu, jinak #SS(0);
  FI;
  RPL návratového CS >= CPL, jinak #GP(návratový selektor);
  IF RPL návratového selektoru = CPL
  THEN GOTO NÁVRAT-DO-STEJNÉ-ÚROVNĚ;
  ELSE GOTO NÁVRAT-DO-NIŽŠÍ-ÚROVNĚ;
  FI;

NÁVRAT-DO-STEJNÉ-ÚROVNĚ: (* PE=1, VM=0, NT=1, nevrací se do V86, RPL=CPL *)
  IF velikost operandu = 32
  THEN
    horních 12 slabik zásobníku je uvnitř limitu, jinak #SS(0);
    návratový CS selektor (uložen na eSP+4) je platný, jinak #GP(0);
  ELSE
    horních 6 slabik zásobníku je uvnitř limitu, jinak #SS(0);
    návratový CS selektor (uložen na eSP+2) je platný, jinak #GP(0);
  FI;
  index návratového selektoru ukazuje dovnitř limitu tabulky
         popisovačů, jinak #GP(návratový selektor);
  slabika AR indikuje proveditelný segment, jinak #GP(návratový selektor);
  IF není přizpůsobitelný
  THEN DPL instrukčního segmentu=CPL, jinak #GP(návratový selektor);
  ELSE DPL>CPL, jinak #GP(návratový selektor);
  FI;
```

```
  segment je přítomný, jinak #NP(návratový selektor);
  instrukční registr ukazuje dovnitř limitu instr. segmentu, jinak #GP(0);
  IF velikost operandu = 32
  THEN
    naplň CS:EIP ze zásobníku;
    naplň nevidit. část registru CS popisovačem;
    naplň EFLAGS ze zásobníku;
    eSP := eSP + 12;
  ELSE
    naplň CS:IP ze zásobníku;
    naplň nevidit. část registru CS popisovačem;
    naplň FLAGS ze zásobníku;
    eSP := eSP + 6;
  FI;
  END;

NÁVRAT-DO-NIŽŠÍ-ÚROVNĚ:
  IF velikost operandu = 32
  THEN horních 20 slabik zásobníku je uvnitř limitu, jinak #SS(0);
  ELSE horních 10 slabik zásobníku je uvnitř limitu, jinak #SS(0);
  FI;
  Kontrola návratového selektoru CS a jeho popisovače:
    selektor je platný, jinak #GP(0);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
           jinak #GP(návratový selektor);
    slabika AR indikuje instrukční segment, jinak #GP(návratový selektor);
    IF není přizpůsobitelný
    THEN
      DPL = RPL selektoru CS, jinak #GP(návratový selektor);
    ELSE (* je přizpůsobitelný *)
      DPL > CPL, jinak #GP(návratový selektor);
    FI;
    segment je přítomný, jinak #NP(návratový selektor);
  Kontrola návratového selektoru SS a jeho popisovače:
    selektor je platný, jinak #GP(0);
    index selektoru ukazuje dovnitř limitu tabulky popisovačů,
           jinak #GP(SS selektor);
    RPL selektoru = RPL návratového selektoru CS, jinak #GP(SS selektor);
    slabika AR indikuje zapisovatelný datový segment,
           jinak #GP(SS selektor);
    DPL zásobníkového segmentu = RPL návratového selektoru CS,
           jinak #GP(SS selektor);
    segment se zásobníkem je přítomný, jinak #NP(SS selektor);
```

```
instrukční registr ukazuje dovnitř limitu inst. segmentu, jinak #GP(0);
IF velikost operandu = 32
THEN
  naplň CS:EIP ze zásobníku;
  naplň EFLAGS ze zásobníku (eSP+8);
ELSE
  naplň CS:IP ze zásobníku;
  naplň FLAGS ze zásobníku (eSP+4);
FI;
naplň SS:eSP ze zásobníku;
CPL := RPL návratového selektoru CS;
naplň nevidit. část registru CS popisovačem;
naplň nevidit. část registru SS popisovačem;
FOR každý z ES, FS, GS a DS
DO
  IF současná hodnota není pro novou úroveň platná
  THEN naplň registr neplatným selektorem;
  FI;
  Aby byl obsah registru pro novou úroveň platný, musí být splněno:
    index selektoru ukazuje dovnitř limitu tabulky popisovačů;
    slabika AR indikuje datový nebo čitelný instrukční segment;
    IF segment je datový nebo nepřizpůsobitelný instrukční
    THEN DPL musí být > CPL nebo DPL musí být < RPL;
OD;
END;
```

Výjimky ve chráněném režimu

#GP, #SS a #NP se generují tak, jak bylo uvedeno výše. Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

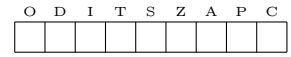
Pokud libovolná část operandu je vybírána vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #GP(0) se generuje při IOPL<3, #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

12.11 Cykly

LOOP — *Unconditional Loop Control*
LOOPcond — *Conditional Loop Control*



Popis:

Instrukce LOOP podporuje konstruování nepodmíněných cyklů. Používá registr CX jako čítač počtu průchodů cyklem. Činnost instrukce si vysvětlíme na příkladu:

```
        MOV  CX,počet_průchodů
Opakuj:
         .
         .          ; Tělo cyklu.
        LOOP Opakuj  ; CX:=CX-1 a je-li CX nenulové
                     ;   provede se SHORT skok na Opakuj.
        ; Zde se pokračuje, je-li po odečtení jedničky CX=0.
```

Při provádění instrukce LOOP je nejdříve snížen obsah registru CX o jedničku a pak je CX testován. V případě, že CX je různý od nuly, provede se krátký skok na návěští uvedené za instrukcí. Návěští reprezentuje 8bitovou hodnotu se znaménkem, která určuje vzdálenost cílového návěští od instrukce LOOP. Proto vzdálenost návěští musí být v rozmezí -128 až 127 slabik. V případě, že CX je roven nule, pokračuje zpracování následující instrukcí.

Instrukce podmíněného provádění cyklu LOOP*cond* je rozšířením instrukce LOOP. Tyto instrukce provedou krátký skok na návěští tehdy, je-li po odečtení jedničky CX$\neq$0 a zároveň je splněna zadaná podmínka.

Instrukce	Popis
LOOP *rel8*	Čítač-1, krátký skok při Čítač$\neq$0
LOOPE *rel8*	Čítač-1, krátký skok při Čítač$\neq$0 AND ZF=1
LOOPZ *rel8*	Čítač-1, krátký skok při Čítač$\neq$0 AND ZF=1
LOOPNE *rel8*	Čítač-1, krátký skok při Čítač$\neq$0 AND ZF=0
LOOPNZ *rel8*	Čítač-1, krátký skok při Čítač$\neq$0 AND ZF=0

Operace

```
IF velikost adresy = 16 THEN Čítač je CX; ELSE Čítač je ECX; FI;
Čítač := Čítač - 1;
IF instrukce <> LOOP
THEN
  IF (instrukce = LOOPE) OR (instrukce = LOOPZ)
```

```
  THEN PodmínkaSkoku := (ZF=1) AND (Čítač<>0);
  FI;
  IF (instrukce = LOOPNE) OR (instrukce = LOOPNZ)
  THEN PodmínkaSkoku := (ZF=0) AND (Čítač<>0);
  FI;
FI;
IF PodmínkaSkoku
THEN
  IF velikost operandu = 16
  THEN
    IP := IP + znaménkové_rozšíření(rel8);
  ELSE (* velikost operandu = 32 *)
    EIP := EIP + znaménkové_rozšíření(rel8);
  FI;
FI;
```

Komentář

Při snižování hodnoty Čítače o jedničku se nenastavují příznaky. Je-li atribut velikosti adresy nastaven na 16 bitů, používá se jako čítač registr CX. Je-li adresový atribut 32 bitů, použije se ECX.

Instrukce LOOP trvá déle než posloupnost dvou instrukcí, z nichž první sníží Čítač a druhá hodnotu Čítače testuje.

Výjimky ve chráněném režimu

Pokud vypočtený offset leží mimo limit segmentu, generuje se #GP(0).

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

Žádné.

12.12 Ovládání V/V

IN

Input from Port

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce IN přenáší data o délce slabiky, slova nebo dvojslova ze V/V brány do registru AL, AX nebo EAX. Druhý operand určuje adresu V/V brány.

Instrukce	Popis
IN AL,*imm8*	Čti slabiku z brány *imm8* do AL
IN AX,*imm8*	Čti slovo z brány *imm8* do AX
IN EAX,*imm8*	Čti dvojslovo z brány *imm8* do EAX
IN AL,DX	Čti slabiku z brány DX do AL
IN AX,DX	Čti slovo z brány DX do AX
IN EAX,DX	Čti dvojslovo z brány DX do EAX

Operace

```
IF (PE=1) AND ((VM=1) OR (CPL>IOPL))
THEN (* režim V86 nebo chráněný režim s CPL>IOPL *)
  IF NOT Povolení_V/V ( V/V_adresa, šířka_registru )
  THEN #GP(0);
  FI;
FI;
registr := [V/V_adresa]; (* AL,AX,EAX := 1,2,4 slabiky čtené ze V/V bran *)
```

Komentář

První operand instrukce je střádač, do něhož se uloží přečtená informace. Na místě prvního operandu smí být pouze AL, AX a EAX. Druhý operand je V/V adresa. Zde může být zadána buď přímá 8bitová hodnota bez znaménka (0 až 255), nebo registr DX (obsahující hodnotu 0 až 65 535). Pokud je použita přímá 8bitová hodnota, je vyšší slabika adresy brány nulová. Přenáší-li se slovo, uloží se do cílového registru obsahy bran na adresách *druhý_operand* a *druhý_operand+1*.

V režimu V86 a v chráněném režimu při CPL>IOPL se před provedením instrukce testuje mapa V/V bran v TSS procesu, který instrukci vydává. Pokud bit odpovídající bráně (v případě přenosu slova a dvojslova: bity odpovídající branám) operaci nepovoluje, generuje se #GP(0).

Výjimky ve chráněném režimu

#GP(0) se generuje při nepovolení operace na zadané bráně mapou V/V bran v TSS.

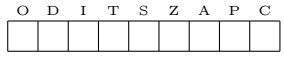
Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

#GP(0) se generuje při nepovolení operace na zadané bráně mapou V/V bran v TSS.

OUT *Output to Port*



Popis:

Instrukce OUT přenáší data ze střádače AL, AX nebo EAX do V/V brány. Adresu brány určuje první operand.

Instrukce	Popis
OUT *imm8*,AL	Zapiš slabiku z AL do brány *imm8*
OUT *imm8*,AX	Zapiš slovo z AX do brány *imm8*
OUT *imm8*,EAX	Zapiš dvojslovo z EAX do brány *imm8*
OUT DX,AL	Zapiš slabiku z AL do brány DX
OUT DX,AX	Zapiš slovo z AX do brány DX
OUT DX,EAX	Zapiš dvojslovo z EAX do brány DX

Operace

```
IF (PE=1) AND ((VM=1) OR (CPL>IOPL))
THEN (* režim V86 nebo chráněný režim s CPL>IOPL *)
  IF NOT Povolení_V/V ( V/V_adresa, šířka_registru )
  THEN #GP(0);
  FI;
FI;
[V/V_adresa] := registr; (* AL,AX,EAX *)
```

Komentář

Další informace viz instrukce IN.

12.13 Další instrukce přesunů dat

XLAT — *Table Lookup Translation*
XLATB — *Table Lookup Translation*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce XLAT transformuje obsah AL podle tabulky. Před provedením instrukce musí být v DS:eBX uložena adresa tabulky a v registru AL index (neznaménkové číslo 0 až 255), který se sečte s obsahem registru eBX. Slabikou z vypočtené adresy se naplní opět registr AL. Instrukce se používá pro transformaci mezi kódy.

Instrukce	Popis
XLAT *m8*	AL := DS:[eBX + neznaménkové AL]
XLATB	AL := DS:[eBX + neznaménkové AL]

Operace

```
IF velikost adresy = 16
THEN
  AL := [BX + neznaménkové_rozšíření(AL)]
ELSE (* velikost adresy = 32 *)
  AL := [EBX + neznaménkové_rozšíření(AL)]
FI;
```

Komentář

Registr BX se používá tehdy, je-li adresový atribut nastaven na 16 bitů. Registr EBX při adresovém atributu nastaveném na 32 bitů.

Instrukce XLATB, která je bez operandu, se používá tehdy, pokud tabulka adresovaná offsetem BX nebo EBX leží v segmentu pokrývaném registrem DS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

LEA — *Load Effective Address*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce LEA ukládá do prvního operandu efektivní (offsetovou) část adresy druhého operandu, na jehož místě je uložena adresa objektu v paměti.

Instrukce	Popis
LEA *r16,m*	do *r16* ulož efektivní adresu *m*
LEA *r32,m*	do *r32* ulož efektivní adresu *m*

Operace

```
IF     velikost operandu = 16 AND velikost adresy = 16
THEN     r16 := 16bitová_adresa(m);
ELSEIF velikost operandu = 16 AND velikost adresy = 32
THEN     r16 := zkraťˇ_na_16_bitů(32bitová_adresa(m));
ELSEIF velikost operandu = 32 AND velikost adresy = 16
THEN     r32 := neznaménkové_rozšíření(16bitová_adresa(m));
ELSEIF velikost operandu = 32 AND velikost adresy = 32
THEN     r32 := 32bitová_adresa(m);
FI;
```

Komentář

Hodnota `velikost operandu` je určena typem prvního operandu. Hodnota `velikost adresy` je určena hodnotou adresového atributu instrukčního segmentu.

Výjimky ve chráněném režimu

#UD se generuje, je-li druhý operand registr.

Výjimky v reálném režimu

Je-li druhý operand registr, generuje se výjimka 6.

Výjimky v režimu V86

Je-li druhý operand registr, generuje se výjimka 6.

BSWAP *Byte Swap*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce BSWAP zamění pořadí slabik v 32bitovém operandu do tvaru „**Big-Endian**“.

Instrukce	Popis
BSWAP *r32*	zaměň slabiky do tvaru Big-Endian

Operace

```
Pomocná := r32;
r32(7..0) := Pomocná(31..24);
r32(15..8) := Pomocná(23..16);
r32(23..16) := Pomocná(15..8);
r32(31..24) := Pomocná(7..0);
```

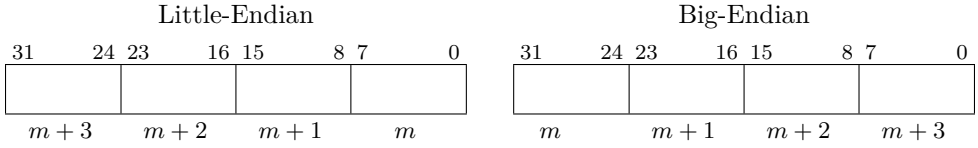
Komentář

Procesory INTEL ukládají data delší než jedna slabika tak, že slabika nejnižšího řádu leží na nejnižší adrese (viz obr. na str. 18). Tento formát bývá často pojmenován „**Little-Endian**“.

Instrukce BSWAP zamění pořadí slabik v 32bitovém operandu do tvaru „**Big-Endian**“, ve kterém je na nejnižší adrese uložena slabika nejvyššího řádu (viz obr. 12.8). Instrukce pomáhá programátorovi při tvorbě programového vybavení, které má sdílet datové struktury (např. databáze) spolu s procesory ukládajícími data ve tvaru Big-Endian (např. se sálovými počítači IBM). Záměnu pořadí slabik v 16bitovém slově lze provést instrukcí XCHG. Je-li instrukce BSWAP použita s 16bitovým operandem, operace se neprovede a obsah registru zůstává nezměněn.
Instrukce negeneruje žádné výjimky.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel486.



Obr. 12.8 Srovnání tvaru Little-Endian a Big-Endian

XADD *Exchange and Add*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce XADD naplní druhý operand obsahem prvního a naplní první operand součtem původního obsahu druhého operandu a prvního operandu.

Instrukce	Popis
XADD *r/m8,r8*	zaměň a sečti

Operace

```
Pomocná := druhý_operand + první_operand;
druhý_operand := první_operand;
první_operand := Pomocná;
```

Komentář

Instrukce může být použita s prefixem LOCK.

Nastavované příznaky

Příznaky se nastavují shodně s instrukcí ADD.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel486.

12.14 Řetězcové instrukce

Instrukce ze skupiny řetězcových instrukcí pracují zpravidla se dvěma místy v paměti. Zdrojové místo v paměti bývá adresováno dvojicí DS:eSI a cílové místo dvojicí ES:eDI. Poněvadž řetězce jsou určeny adresami uloženými v těchto registrech, instrukce nepotřebují operandy. Řetězcové instrukce pracují se slabikami, s 16bitovými slovy nebo 32bitovými dvojslovy.

Jsou-li v instrukci operandy, používá je asembler pro kontrolu typů a pro generování prefixu měnícího implicitní segmentový registr a pro generování instrukce určené pro práci se slabikami, slovy nebo se dvojslovy. Instrukce s názvem končícím písmenem „B“ nevyžaduje operandy, protože předpokládá, že obsah dvojice DS:eSI a ES:eDI ukazuje na slabikovou strukturu. Instrukce, jejíž název končí „W“, pracuje s 16bitovými slovy. Instrukce, jejíž název končí „D“, pracuje s 32bitovými dvojslovy. Čtvrtý typ instrukcí (nekončící „B“, „W“, ani „D“) vyžaduje operandy.

Po provedení instrukce se upravuje obsah eSI a eDI tak, aby ukazoval na další zpracovávaný prvek. Obsah eSI a eDI se zvyšuje nebo zmenšuje o velikost zpracovávaného prvku (o 1 pro slabiku, o 2 pro slovo a o 4 pro dvojslovo). Zda se řetězce zpracovávají zepředu nebo zezadu, určuje momentální hodnota příznaku DF (Direction Flag) z příznakového registru. Hodnota DF=0 způsobí zvětšování obsahu eSI a eDI po provedení operace a hodnota DF=1 zmenšování obsahu eSI a eDI o velikost zpracovávaného prvku. Příznak DF se modifikuje instrukcemi STD a CLD.

Dále uvedené instrukce zpracovávají vždy jenom jeden prvek. Po provedení operace nastaví ukazatele na prvek další. Chceme-li zpracovat naráz celý řetězec prvků, můžeme použít prefix REP nebo REP*cond* (viz též str. 270).

MOVS/MOVSB/MOVSW/MOVSD *Move String*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce MOVS přesouvá obsah slabiky (slova, dvojslova) z adresy DS:eSI do paměťového místa s adresou ES:eDI.

Instrukce	Popis
MOVS *m8,m8*	přesuň slabiku z [eSI] na ES:[eDI]
MOVS *m16,m16*	přesuň slovo z [eSI] na ES:[eDI]
MOVS *m32,m32*	přesuň dvojslovo z [eSI] na ES:[eDI]
MOVSB	přesuň slabiku z DS:[eSI] na ES:[eDI]
MOVSW	přesuň slovo z DS:[eSI] na ES:[eDI]
MOVSD	přesuň dvojslovo z DS:[eSI] na ES:[eDI]

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: SI pro zdrojový index a DI pro cílový index;
ELSE pro adresaci použij: ESI pro zdrojový index a EDI pro cílový index;
FI;
IF (instrukce = MOVSB) OR (instrukce má slabikové operandy)
THEN
  [cílový index] := [zdrojový index]; (* slabikové přiřazení *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = MOVSW) OR (instrukce má slovní operandy)
THEN
  [cílový index] := [zdrojový index]; (* slovní přiřazení *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = MOVSD) OR (instrukce má dvojslovní operandy) *)
  [cílový index] := [zdrojový index]; (* dvojslovní přiřazení *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
zdrojový index := zdrojový index + Přírůstek;
cílový index := cílový index + Přírůstek;
```

Komentář

Instrukce může být uvedena buď bez operandů (MOVSB/MOVSW/MOVSD), nebo s operandy (MOVS). Pokud není v instrukci uveden explicitně u zdrojového operandu segmentový registr, je použit DS. Cílový operand musí být vždy adresován segmentovým registrem ES. Při použití instrukce s operandy je překladačem kontrolován typ operandů a podle délky (slabika, slovo, dvojslovo) je generována odpovídající instrukce. Rovněž tak je uveden případný prefix změny segmentového registru, je-li jiný než implicitní. Jiný význam operandy nemají.

Je-li příznak DF=0, pak jsou obsahy obou registrů SI a DI po provedení přenosu zvětšeny, v opačném případě se zmenšují. Hodnota, o kterou se obsah registrů zvětšuje nebo zmenšuje, je 1 pro slabikové operandy, 2 pro slovní operandy a 4 pro dvojslovní operandy.

Instrukce může mít prefix REP pro opakované zpracování eCX elementů (viz popis REP v závěru této podkapitoly).

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

CMPS/CMPSB/CMPSW/CMPSD *Compare String*

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Instrukce CMPS porovnává slabiku (slovo, dvojslovo) cílového operandu na adrese ES:eDI se zdrojovým operandem na adrese DS:eSI.

Instrukce	Popis
CMPS *m8,m8*	porovnej slabiku z [eSI] s ES:[eDI]
CMPS *m16,m16*	porovnej slovo z [eSI] s ES:[eDI]
CMPS *m32,m32*	porovnej dvojslovo z [eSI] s ES:[eDI]
CMPSB	porovnej slabiku z DS:[eSI] s ES:[eDI]
CMPSW	porovnej slovo z DS:[eSI] s ES:[eDI]
CMPSD	porovnej dvojslovo z DS:[eSI] s ES:[eDI]

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: SI pro zdrojový index a DI pro cílový index;
ELSE pro adresaci použij: ESI pro zdrojový index a EDI pro cílový index;
FI;
IF (instrukce = CMPSB) OR (instrukce má slabikové operandy)
THEN
  FLAGS := podle ([zdrojový index] - [cílový index]); (* slabikové porovnání *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = CMPSW) OR (instrukce má slovní operandy)
THEN
  FLAGS := podle ([zdrojový index] - [cílový index]); (* slovní porovnání *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = CMPSD) OR (instrukce má dvojslovní operandy) *)
  FLAGS := podle ([zdrojový index] - [cílový index]); (* dvojslovní porovnání *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
zdrojový index := zdrojový index + Přírůstek;
cílový index := cílový index + Přírůstek;
```

Komentář

Porovnání se provede tak, že se odečte cílový operand od zdrojového, tj. (obsah podle DS:eSI) − (obsah podle ES:eDI). V této instrukci, na rozdíl od konvencí firmy Intel, je zaměněno pořadí zdrojového a cílového operandu.

Podle výsledku rozdílu se nastaví obsah příznakového registru a hodnota cílového operandu se nemění. Po provedení porovnání se aktualizují registry eSI a eDI.

Další podrobnosti viz MOVS.

SCAS/SCASB/SCASW *Compare String Data*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Popis:

Instrukce SCAS porovnává prvek řetězce na adrese ES:eDI s obsahem registru AL, AX nebo EAX.

Instrukce	Popis
SCAS *m8*	porovnej slabiku z AL s ES:[eDI]
SCAS *m16*	porovnej slovo z AX s ES:[eDI]
SCAS *m32*	porovnej dvojslovo z EAX s ES:[eDI]
SCASB	porovnej slabiku z AL s ES:[eDI]
SCASW	porovnej slovo z AX s ES:[eDI]
SCASD	porovnej dvojslovo z EAX s ES:[eDI]

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: DI pro cílový index;
ELSE pro adresaci použij: EDI pro cílový index;
FI;
IF (instrukce = SCASB) OR (instrukce má slabikové operandy)
THEN
  FLAGS := podle (AL - [cílový index]); (* slabikové porovnání *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = SCASW) OR (instrukce má slovní operandy)
THEN
  FLAGS := podle (AX - [cílový index]); (* slovní porovnání *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = SCASD) OR (instrukce má dvojslovní operandy) *)
  FLAGS := podle (EAX - [cílový index]); (* dvojslovní porovnání *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
cílový index := cílový index + Přírůstek;
```

Komentář

Je-li prvkem řetězce slabika, pracuje se s registrem AL, je-li prvkem slovo, použije se registr AX, je-li prvkem dvojslovo, použije se registr EAX. Porovnání se provede tak, že se od obsahu registru AL (AX, EAX) odečte obsah prvku řetězce na adrese ES:eDI. Podle výsledku se nastaví příznakový registr, rozdíl se nikam neukládá. Po provedení porovnání se aktualizuje registr eDI.

Další podrobnosti viz MOVS.

LODS/LODSB/LODSW/LODSD *Load String*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce LODS naplní registr AL, AX nebo EAX obsahem prvku řetězce uloženého na adrese DS:eSI.

Instrukce	Popis
LODS *m8*	naplň AL slabikou z [eSI]
LODS *m16*	naplň AX slovem z [eSI]
LODS *m32*	naplň EAX dvojslovem z [eSI]
LODSB	naplň AL slabikou z DS:[eSI]
LODSW	naplň AX slovem z DS:[eSI]
LODSD	naplň EAX dvojslovem z DS:[eSI]

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: SI pro zdrojový index
ELSE pro adresaci použij: ESI pro zdrojový index
FI;
IF (instrukce = LODSB) OR (instrukce má slabikové operandy)
THEN
  AL := [zdrojový index]; (* slabikový přenos *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = LODSW) OR (instrukce má slovní operandy)
THEN
  AX := [zdrojový index]; (* slovní přenos *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = LODSD) OR (instrukce má dvojslovní operandy) *)
  EAX := [zdrojový index]; (* dvojslovní přenos *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
zdrojový index := zdrojový index + Přírůstek;
```

Komentář

Další podrobnosti viz MOVS.

STOS/STOSB/STOSW/STOSD *Store String Data*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce STOS uloží obsah registru AL, AX nebo EAX do prvku řetězce na adrese ES:eDI.

Instrukce	Popis
STOS *m8*	ulož slabiku z AL do ES:[eDI]
STOS *m16*	ulož slovo z AX do ES:[eDI]
STOS *m32*	ulož dvojslovo z EAX do ES:[eDI]
STOSB	ulož slabiku z AL do ES:[eDI]
STOSW	ulož slovo z AX do ES:[eDI]
STOSD	ulož dvojslovo z EAX do ES:[eDI]

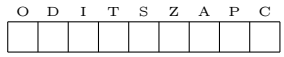
Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: DI pro cílový index;
ELSE pro adresaci použij: EDI pro cílový index;
FI;
IF (instrukce = STOSB) OR (instrukce má slabikové operandy)
THEN
  [cílový index] := AL; (* slabikový přenos *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = STOSW) OR (instrukce má slovní operandy)
THEN
  [cílový index] := AX; (* slovní přenos *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = STOSD) OR (instrukce má dvojslovní operandy) *)
  [cílový index] := EAX; (* dvojslovní přenos *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
cílový index := cílový index + Přírůstek;
```

Komentář

Další podrobnosti viz MOVS.

INS/INSB/INSW/INSD *Input from Port to String*



Popis:

Instrukce INS přenáší data ze vstupní brány zadané registrem DX a ukládá je do prvku řetězce na adrese ES:eDI.

Instrukce	Popis
INS *m8,DX*	čti slabiku z brány DX do ES:[eDI]
INS *m16,DX*	čti slovo z brány DX do ES:[eDI]
INS *m32,DX*	čti dvojslovo z brány DX do ES:[eDI]
INSB	čti slabiku z brány DX do ES:[eDI]
INSW	čti slovo z brány DX do ES:[eDI]
INSD	čti dvojslovo z brány DX do ES:[eDI]

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: DI pro cílový index;
ELSE pro adresaci použij: EDI pro cílový index;
FI;
IF (PE=1) AND ((VM=1) OR (CPL>IOPL))
THEN (* režim V86 nebo chráněný režim s CPL>IOPL *)
  IF NOT Povolení_V/V ( V/V_adresa, šířka_registru )
  THEN #GP(0);
  FI;
FI;
IF (instrukce = INSB) OR (instrukce má slabikové operandy)
THEN
  [cílový index] := brána DX; (* slabikový přenos *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = INSW) OR (instrukce má slovní operandy)
THEN
  [cílový index] := brána DX; (* slovní přenos *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = INSD) OR (instrukce má dvojslovní operandy) *)
  [cílový index] := brána DX; (* dvojslovní přenos *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
cílový index := cílový index + Přírůstek;
```

Komentář

Instrukce INS nedovoluje adresovat bránu přímou hodnotou. Další podrobnosti viz MOVS a IN.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel286.

OUTS/OUTSB/OUTSW/OUTSD *Output String*

Popis:

O	D	I	T	S	Z	A	P	C

Instrukce OUTS přenese prvek řetězce z adresy DS:eSI do výstupní brány zadané registrem DX.

Instrukce	Popis
OUTS *DX,m8*	přenes slabiku z [eSI] do brány DX
OUTS *DX,m16*	přenes slovo z [eSI] do brány DX
OUTS *DX,m32*	přenes dvojslovo z [eSI] do brány DX
OUTSB	přenes slabiku z [eSI] do brány DX
OUTSW	přenes slovo z [eSI] do brány DX
OUTSD	přenes dvojslovo z [eSI] do brány DX

Operace

```
IF velikost adresy = 16
THEN pro adresaci použij: SI pro zdrojový index
ELSE pro adresaci použij: ESI pro zdrojový index
FI;
IF (PE=1) AND ((VM=1) OR (CPL>IOPL))
THEN (* režim V86 nebo chráněný režim s CPL>IOPL *)
  IF NOT Povolení_V/V ( V/V_adresa, šířka_registru )
  THEN #GP(0);
  FI;
FI;
IF (instrukce = OUTSB) OR (instrukce má slabikové operandy)
THEN
  brána DX := [zdrojový index]; (* slabikový přenos *)
  IF DF=0 THEN Přírůstek:=1; ELSE Přírůstek:=-1; FI;
ELSEIF (instrukce = OUTSW) OR (instrukce má slovní operandy)
THEN
```

```
  brána DX := [zdrojový index]; (* slovní přenos *)
  IF DF=0 THEN Přírůstek:=2; ELSE Přírůstek:=-2; FI;
ELSE (* (instrukce = OUTSD) OR (instrukce má dvojslovní operandy) *)
  brána DX := [zdrojový index]; (* dvojslovní přenos *)
  IF DF=0 THEN Přírůstek:=4; ELSE Přírůstek:=-4; FI;
FI;
zdrojový index := zdrojový index + Přírůstek;
```

Komentář

Instrukce OUTS nedovoluje adresovat bránu přímou hodnotou. Další podrobnosti viz MOVS a OUT.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel286.

REP/REP*cond* — *Repeat Following String Operation*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukční prefix REP lze aplikovat na řetězcové instrukce. Uvedení prefixu před instrukcí odstartuje její opakované provádění. Při každé iteraci se sníží obsah eCX o jedničku. Nulová hodnota eCX ukončí opakování.

Podmíněné varianty prefixu REP*cond* testují navíc ještě příznak ZF a má význam je používat s instrukcemi CMPS nebo SCAS, které jej nastavují.

Operace

```
IF velikost adresy = 16
THEN jako Čítač používej CX;
ELSE (* velikost adresy = 32 *) jako čítač používej ECX;
FI;
WHILE Čítač <> 0
DO
  Test, nežádá-li někdo o přerušení. Pokud ano, přerušení se uplatní.
  Jedno provedení požadované instrukce.
  Čítač := Čítač - 1; (* nemění se příznaky *)
  IF instrukce je CMPSx nebo SCASx
  THEN
    IF (prefix je REP, REPE nebo REPZ) AND (ZF=0)
    THEN END; (* ukončení opakování *)
```

```
    ELSE
      IF (prefix je REPNE nebo REPNZ) AND (ZF=1)
      THEN END; (* ukončení opakování *)
      FI;
    FI;
  FI;
OD;
```

Komentář

REP a REP*cond* jsou pouze symbolické zápisy skutečných instrukčních prefixů, které bývají uvedeny před operačním znakem instrukce. Operační znaky diskutovaných prefixů jsou: F3h (REP, REPE, REPZ) a F2h (REPNE, REPNZ).

Prefixy REP se aplikují vždy jenom na jednu instrukci. Pokud potřebujete opakovat skupinu instrukcí, použijte např. instrukci pro tvorbu cyklů LOOP. Použití prefixů si ukažme na dvou jednoduchých příkladech:

```
    LDS  SI,adresa_zdrojového_řetězce
    LES  DI,adresa_cílového_řetězce
    MOV  CX,počet_prvků_řetězce
    REP  MOVSW   ; Zkopíruje prvky řetězce typu slovo.
```

```
    LDS  SI,adresa_zdrojového_řetězce
    LES  DI,adresa_cílového_řetězce
    MOV  CX,počet_prvků_řetězce
    REPE CMPSB  ; Srovnává obsahy dvou řetězců typu slabika.
    JE   Shodné ; Řetězce mají shodný obsah.
    ; Řetězce se liší.
```

Použijete-li varianty REP INS a REP OUT, nemusejí všechny V/V brány a připojená zařízení zvládnout tak vysokou přenosovou rychlost.

Nekombinujte dohromady prefix REP a instrukci LOOP. Návrháři procesoru nezaručují definovanou činnost. Rovněž tak není definováno chování při použití prefixu REP s jinou instrukcí než ze skupiny řetězcových.

Nastane-li během provádění instrukce CMPS nebo SCAS s prefixem REPNE výpadek stránky, obnoví se obsah EFLAGS do podoby před zahájením instrukce. Protože obě instrukce nepoužívají registr příznaků pro vstup, může procesor po zavedení stránky instrukci zopakovat.

12.15 Instrukce pro práci s bity

BSF

Bit Scan Forward

O	D	I	T	S	Z	A	P	C
?			?	?	*	?	?	?

Popis:

Instrukce BSF prohlíží druhý operand po bitech počínaje bitem 0 a hledá první výskyt nenulového bitu. Instrukce BSF prohlíží bity v objektu směrem nahoru.

Instrukce	Popis
BSF *r16,r/m16*	prohledávej slovo *r/m16* vpřed
BSF *r32,r/m32*	prohledávej dvojslovo *r/m32* vpřed

Operace

```
IF r/m = 0
THEN
  ZF := 1;
  r := nedefinovaná hodnota;
ELSE
  Pomocná := 0;
  ZF := 0;
  WHILE Bit[r/m,Pomocná] = 0
  DO
    Pomocná := Pomocná + 1;
    r := Pomocná;
  OD;
FI;
```

Komentář

Pokud jsou všechny bity druhého operandu nulové, je nastaven příznak ZF na nulu. V opačném případě je ZF=1 a první operand obsahuje číslo prvního nenulového bitu ve druhém operandu.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce této skupiny jsou zavedeny od procesoru Intel386.

BSR *Bit Scan Reverse*

O	D	I	T	S	Z	A	P	C
?			?	?	*	?	?	?

Popis:

Instrukce BSR prohlíží druhý operand po bitech počínaje nejvyšším a hledá první výskyt nenulového bitu. Instrukce BSR prohlíží bity v objektu směrem dolů.

Instrukce	Popis
BSR *r16,r/m16*	prohledávej slovo *r/m16* vzad
BSR *r32,r/m32*	prohledávej dvojslovo *r/m32* vzad

Operace

```
IF r/m = 0
THEN
  ZF := 1;
  r := nedefinovaná hodnota;
ELSE
  Pomocná := velikost operandu - 1;
  ZF := 0;
  WHILE Bit[r/m,Pomocná] = 0
  DO
    Pomocná := Pomocná - 1;
    r := Pomocná;
  OD;
FI;
```

Komentář

Další podrobnosti viz BSF.

BT/BTC/BTR/BTS *Bit Test*

O	D	I	T	S	Z	A	P	C
								*

Popis:

Skupina zahrnuje čtyři instrukce BT, BTC, BTR a BTS. Instrukce BT kopíruje jeden bit prvního operandu do příznaku CF. Číslo bitu je určeno druhým operandem (nejnižší bit je označen 0). Instrukce BTC pak navíc hodnotu bitu prvního operandu invertuje, instrukce BTR vynuluje a BTS nastaví na 1.

Instrukce	Popis
BT *r/m16,r16*	CF := bit
BT *r/m32,r32*	CF := bit
BT *r/m16,imm8*	CF := bit
BT *r/m32,imm8*	CF := bit
BTC *r/m16,r16*	CF := bit; bit := NOT bit
BTC *r/m32,r32*	CF := bit; bit := NOT bit
BTC *r/m16,imm8*	CF := bit; bit := NOT bit
BTC *r/m32,imm8*	CF := bit; bit := NOT bit
BTR *r/m16,r16*	CF := bit; bit := 0
BTR *r/m32,r32*	CF := bit; bit := 0
BTR *r/m16,imm8*	CF := bit; bit := 0
BTR *r/m32,imm8*	CF := bit; bit := 0
BTS *r/m16,r16*	CF := bit; bit := 1
BTS *r/m32,r32*	CF := bit; bit := 1
BTS *r/m16,imm8*	CF := bit; bit := 1
BTS *r/m32,imm8*	CF := bit; bit := 1

Operace

```
CF := Bit[první_operand, druhý_operand];
CASE instrukce OF
  BTC: Bit[první_operand, druhý_operand] :=
                NOT Bit[první_operand, druhý_operand];
  BTR: Bit[první_operand, druhý_operand] := 0;
  BTS: Bit[první_operand, druhý_operand] := 1;
```

Komentář

Číslo zpracovávaného bitu se zadává buď přímou hodnotou, nebo ve všeobecném registru. Přímá hodnota může v instrukci být pouze 8bitová. Z operandu je potom převzata hodnota MOD 32. Z toho plyne, že nejvyšší číslo bitu je 31 a že v bitových řetězcích lze touto instrukcí zpracovat pouze první slovo nebo dvojslovo.

Některé překladače však přímou hodnotu druhého operandu vyšší než 31 zpracují tak, že odpovídající slabikový posuv přičtou k prvnímu operandu a ve druhém operandu ponechají opět hodnotu MOD 32.

Instrukce pracuje vždy minimálně se slovem nebo dvojslovem, i když testujete bit pouze v dolní slabice. Tento fakt může sebou nést problém při potřebě testovat bit slabiky, na kterou je např. mapována V/V brána. Operace zasáhne vždy slabiky alespoň 2 a při nepozornosti může vyvolat nežádoucí akci na bráně 1 nebo 3 dalších adres.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Instrukce jsou zavedeny od procesoru Intel386.

12.15.1 Instrukce BCD aritmetiky

Instrukce BCD (Binary Coded Decimal – dvojkově kódované desítkové číslo) aritmetiky pracují se dvěma typy dat. V obou typech budeme pracovat s pojmem **BCD číslice**, což je číslice v intervalu 0 až 9 uložená ve čtyřech bitech. Tzn., že se v těchto bitech **nesmí** vyskytnout kombinace v intervalu 10 až 15. Čtyři bity (tj. polovina slabiky) čísel 0 až 3 nebo 4 až 7 budeme nazývat **půlslabika** (Nibble).

Nezhuštěný tvar (Unpacked Decimal) je tvar, ve kterém je v jedné slabice uložena jedna BCD číslice. Číslice je uložena v dolní půlslabice (v dolních 4 bitech), horní půlslabika musí být pro operace násobení a dělení nulová, pro ostatní operace může mít libovolný obsah.

Zhuštěný tvar (Packed Decimal) je tvar, ve kterém jsou v jedné slabice uloženy dvě BCD číslice. Každá číslice v jedné půlslabice. Číslice vyššího řádu je uložena ve vyšší půlslabice. Slabika ve zhuštěném BCD tvaru může tedy obsahovat číslo v intervalu 0 až 99.

Instrukce BCD aritmetiky používají příznak AF (Auxiliary Carry) příznakového registru. Do tohoto příznaku se ukládá přenos z nejnižší do vyšší půlslabiky (tedy vždy z bitu 3 do bitu 4, bez ohledu na šířku operandu).

Procesor přímo nenabízí instrukce pro operace (sčítání, násobení atd.) v BCD kódu, poskytuje pouze sadu instrukcí, které připraví data v BCD kódu do tvaru vhodného pro zpracování klasickými (ne BCD) instrukcemi a které po zpracování výsledek převedou zpět do BCD kódu. Instrukce této skupiny nemají operand a negenerují výjimky.

AAA — *ASCII Adjust after Adition*

O	D	I	T	S	Z	A	P	C
?				?	?	*	?	*

Popis:

Instrukce AAA se používá následně po ADD, která sčítala dvě BCD číslice v nezhuštěném tvaru a součet uložila v binárním tvaru do registru AL. V registru AL na vstupu této instrukce může teoreticky být číslo v intervalu 0 až 19.

Operace

```
ALpřenos := AL > 0F9h; (* 1 při splnění podmínky *)
IF ((AL AND 0Fh) > 9) OR (AF = 1)
THEN
  AL := (AL + 6) AND 0Fh;
  AH := AH + 1 + ALpřenos;
  AF := 1;
  CF := 1;
ELSE
  AF := 0;
  CF := 0;
  AL := AL AND 0Fh;
FI;
```

Komentář

AAA převádí binární obsah AL na BCD číslici a případný přenos do vyššího BCD řádu následovně:

Je-li nastaven příznak AF nebo číslo v AL je > 9, zvýší se obsah AH o jedničku, AL se zvýší o 6 (konverze zpět na BCD číslici), vynuluje se horní půlslabika AL a nastaví se AF=CF=1.

Není-li nastaven příznak AF a zároveň číslo v AL je < 10, nemění se obsah AX a nastaví se AF=CF=0.

Pro převedení BCD číslice v AL na ASCII znak je možné po instrukci AAA použít instrukci např. `OR AL,30H`.

AAD

ASCII Adjust AX before Division

Popis:

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	?

Instrukce AAD převádí dvě BCD číslice v nezhuštěném tvaru z registru AX (v AH je číslice vyššího řádu) na binární číslo do registru AX.

Operace

```
AL := (AH * imm8 + AL) AND 0FFh;
AH := 0;
```

Komentář

Převod se provádí tak, že hodnota registru AH vynásobená číslem 10 se přičte k obsahu registru AL. Registr AH se nuluje. Tato instrukce se většinou používá před instrukcí DIV.

Hodnota `imm8` je přímý operand, uvedený jako druhá slabika instrukce ihned za operačním znakem. Z pohledu překladačů však bývá operační znak této instrukce dvouslabikový, přičemž druhá slabika je vždy 0Ah. Proto programátor nemá v asembleru možnost této instrukci operand normálním způsobem zadat.

AAM *ASCII Adjust AX after Multiplication*

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	?

Popis:

Instrukce AAM se používá po instrukci násobení dvou BCD číslic v nezhuštěném tvaru, která uložila výsledek do registru AX. AAM převádí binární hodnotu AL (protože výsledek násobení nemůže být větší než rozsah zobrazení slabiky) do dvou BCD číslic uložených v registrech AH (vyšší řád) a AL.

Operace

```
regAL := AL;
AH := regAL / imm8;
AL := regAL MOD imm8;
```

Komentář

Obsah registru AX dělí číslem 10, celá část výsledku se uloží do registru AH a zbytek po dělení do registru AL. Poznámku k hodnotě `imm8` viz u AAD.

AAS *ASCII Adjust AL after Subtraction*

O	D	I	T	S	Z	A	P	C
?				?	?	*	?	*

Popis:

Instrukce AAS se používá po instrukci odečítání dvou BCD číslic v nezhuštěném tvaru, která uložila výsledek do registru AL. AAS převádí výsledek v AL na dekadickou číslici, uloženou ve spodních čtyřech bitech registru AL.

Operace

```
ALvýpůjčka := AL < 6; (* 1 při splnění podmínky *)
IF ((AL AND 0Fh) > 9) OR (AF = 1)
THEN
  AL := (AL - 6) AND 0Fh;
  AH := AH - 1 - ALpřenos;
  AF := 1;
  CF := 1;
ELSE
  AF := 0;
  CF := 0;
  AL := AL AND 0Fh;
FI;
```

Komentář

Převod probíhá podobně jako u instrukce AAA. Je-li nastaven příznak AF nebo číslo v AL je > 9, sníží se obsah AH o jedničku, AL se sníží o 6 (konverze zpět na BCD číslici), vynuluje se horní půlslabika AL a nastaví se AF=CF=1. Není-li nastaven příznak AF a zároveň číslo v AL je < 10, nemění se obsah AX a nastaví se AF=CF=0.

DAA *Decimal Adjust AL after Adition*

Popis:

O	D	I	T	S	Z	A	P	C
?				*	*	*	*	*

Instrukce DAA se užívá po instrukci sečtení dvou BCD čísel ve zhuštěném tvaru, která uložila výsledek do registru AL. DAA převádí binární obsah registru AL na dvě BCD číslice ve zhuštěném tvaru a ukládá jej zpět do AL.

Operace

```
IF ((AL AND 0Fh) > 9) OR (AF = 1)
THEN
  AL := AL + 6;
FI;
IF ((AL AND 0F0h) > 90h) OR (CF = 1)
THEN
  AL := AL + 60h;
  CF := 1;
FI;
```

Komentář

Pokud je hodnota dolní půlslabiky AL > 9 nebo je nastaven příznak AF, pak je zvýšen obsah AL o 6 a příznak AF je nastaven na 1.

Následně pokud je výsledek předcházející operace >9Fh nebo je nastaven příznak CF, pak je zvýšen obsah AH o 60h a CF se nastaví na 1.

DAS *Decimal Adjust AL after Subtraction*

O	D	I	T	S	Z	A	P	C
?				*	*	*	*	*

Popis:
Instrukce DAS se používá po instrukci odečtení dvou BCD čísel ve zhuštěném tvaru, která uložila výsledek do registru AL. Instrukce DAS převádí binární obsah registru AL na dvě BCD číslice ve zhuštěném tvaru a ukládá jej do AL.

Operace

```
PomCF := 0;
PomAL := AL;
IF ((PomAL AND 0Fh) > 9) OR (AF = 1)
THEN
  AF := 1;
  AL := AL - 6;
  PomCF := (AL < 0) OR CF;
FI;
IF (PomAL > 99h) OR (CF = 1)
THEN
  AL := AL - 60h;
  PomCF := 1;
FI;
CF := PomCF;
```

Komentář

Pokud je hodnota dolní půlslabiky AL > 9 nebo je nastaven příznak AF, pak je zmenšen obsah AL o 6 a příznak AF je nastaven na 1.

Následně pokud je výsledek předcházející operace >9Fh nebo je nastaven příznak CF, pak je zmenšen obsah AH o 60h a CF se nastaví na 1.

12.16 Řídicí instrukce

CPUID *CPU Identification*

O	D	I	T	S	Z	A	P	C

Popis:
Instrukce CPUID vrací informace o procesoru, na kterém se právě provádí. Typ informací je určen obsahem registru EAX.

Instrukce	Popis
CPUID	vrať identifikaci procesoru podle EAX

Operace

```
CASE EAX OF

  0: EAX := hv; (* Pentium má konstantu hv = 1 *)
               (* 0 <= vstupní EAX <= hv *)
     EBX := identifikace výrobce;
     EDX := identifikace výrobce;
     ECX := identifikace výrobce;

  1: EAX[3..0] := ID;
     EAX[7..4] := Model;
     EAX[11..8] := Rodina; (* 5 pro Pentium *)
     EAX[31..12] je rezervováno pro budoucí použití;
     EBX je rezervováno pro budoucí použití; (* 0 *)
     ECX je rezervováno pro budoucí použití; (* 0 *)
     EDX := příznaky vlastností;

  > hv: (* vyhrazeno pro budoucí použití, nedefinovaný obsah registrů *)

ESAC;
```

Komentář

Předá-li se instrukci řízení s nulovou hodnotou registru EAX, vrací se v EAX nejvyšší hodnota EAX, kterou instrukce podporuje. Procesor Pentium vrací 1. V registrech EBX, EDX a ECX se vrací identifikace výrobce. U procesorů Intel má identifikační řetězec obsah: „GenuineIntel“. Přesněji:

```
EBX := 756E6547h; (* "Genu" s G v dolní půlslabice BL *)
EDX := 49656E69h; (* "ineI" s i v dolní půlslabice DL *)
ECX := 6C65746Eh; (* "ntel" s n v dolní půlslabice CL *)
```

Předá-li se instrukci řízení s jedničkovou hodnotou registru EAX, vrací se v bitech EAX[3..0] identifikační číslo procesoru, v EAX[7..4] číslo modelu (první model má číslo 1) a v EAX[11..8] číslo rodiny procesorů, což pro Pentium je 5. Ostatní bity EAX jsou rezervovány stejně jako EBX a ECX. Pentium nastavuje ještě do registru EDX příznaky vlastností – nastavuje na hodnotu 1BFh. Jednotlivé bity EDX mají tento význam:

```
EDX[0]     na čipu je FPU
EDX[6..1]  běžně nedokumentováno
EDX[7]     umožňuje výjimku 18 (Machine Check)
EDX[8]     instrukce CMPXCHG8B
EDX[31..9] rezervováno
```

Položka označená „běžně nedokumentováno“ je popsána až ve zvláštních dodatcích, které vydává firma Intel k dokumentaci procesoru Pentium. Položka nemusí mít ve všech modelech stejný význam.
Instrukce negeneruje žádné výjimky.

Poznámky k předchozím procesorům

Instrukce je podporována od procesoru Pentium.

NOP *No Operation*

O	D	I	T	S	Z	A	P	C

Popis:
Instrukce NOP neprovádí žádnou akci. Má délku 1 slabiky nesoucí operační znak 90h (stejný operační znak má instrukce XCHG eAX,eAX). Hojně se užívá tam, kde je potřeba doplnit slabiku na zarovnání adresy následující instrukce na hodnotu dělitelnou 2 či 4. Jejím provedením se v procesoru mění pouze obsah eIP.

HLT *Halt CPU*

O	D	I	T	S	Z	A	P	C

Popis: **CPL=0**
Instrukce HLT zastaví provádění instrukcí a uvede procesor do stavu, ze kterého jej vyvede pouze NMI (nemaskovatelné přerušení), vnější přerušení (je-li povoleno) nebo signál RESET. Je-li činnost obnovena pomocí přerušení, tak se tímto přerušením do zásobníku uloží adresa instrukce následující za instrukcí HLT. Návratem z přerušovací rutiny (IRET) pokračuje provádění instrukcí za HLT.

Výjimky ve chráněném režimu

Instrukce HLT se smí provádět pouze na úrovni oprávnění 0. Na jiné úrovni se generuje #GP(0).

Výjimky v reálném režimu

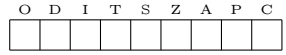
Žádné.

Výjimky v režimu V86

Generuje #GP(0).

ESC *Escape*

Popis:



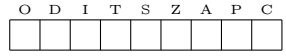
Pomocí prefixu ESC procesor rozpoznává, že jde o instrukci určenou spolupracujícímu koprocesoru (např. FPU).

Příkazy pro koprocesor jsou součástí instrukcí pro procesor. Liší se pouze tím, že jsou uvedeny vzorkem ESC. Jeho výskyt procesoru oznamuje, že tuto instrukci nemá zpracovat, ale že ji má předat koprocesoru včetně operandů. Činnost obou procesorů je možno synchronizovat pomocí instrukce WAIT.

ESC není součásti asemblerů, protože jde jenom o posloupnost pěti bitů (11011) zařazených před příkaz koprocesoru. Příkazy klasické FPU zpravidla překladače zpracovat umějí.

WAIT *Wait*

Popis:



Instrukce WAIT uvede procesor do čekacího stavu na numerickou výjimku #NM. Instrukce WAIT zadaná po ESC zajistí, aby program mohl převzít jakoukoli numerickou výjimku (i chybovou) dříve, než začne analyzovat výsledek. Instrukce FWAIT je synonymem instrukce WAIT.

Výjimky ve chráněném režimu

Je-li nastaveno MP a TS v CR0, generuje se #NM.

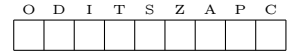
Výjimky v reálném režimu

Je-li nastaveno MP a TS v CR0, generuje se výjimka 7.

Výjimky v režimu V86

Je-li nastaveno MP a TS v CR0, generuje se #NM.

LOCK *Assert LOCK Signal Prefix*



Popis:

Instrukční prefix LOCK aktivuje signál $\overline{\text{LOCK}}$ po dobu zpracovávání instrukce bezprostředně následující za prefixem. Pokud je v systému více procesorů připojených na společnou sběrnici, pak tento signál zabezpečuje procesoru výlučný přístup k této sběrnici. Význam má u instrukcí typu přečti-změň-zapiš a testuj-nastav, např. BTS.

Prefix LOCK se smí použít pouze u těchto instrukcí (u všech instrukcí má význam pouze tehdy, mají-li operand v paměti):

ADD, OR, ADC, SBB, AND, SUB, XOR, NOT, NEG, INC, DEC,
BTS, BTR, BTC,
XCHG ($\overline{\text{LOCK}}$ generuje automaticky),
CMPXCHG, XADD

Použití tohoto prefixu s jinými instrukcemi způsobí přerušení #UD (nedefinovaný operační znak). Prefix LOCK svoji funkci plní bez ohledu na zarovnání operačního znaku a operandů v paměti.

Výjimky ve chráněném režimu

Použije-li se prefix s jinou než vyjmenovanou instrukcí, generuje se #UD. Další výjimky mohou být generovány následnou instrukcí.

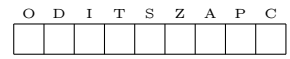
Výjimky v reálném režimu

Použije-li se prefix s jinou než vyjmenovanou instrukcí, generuje se výjimka 6. Další výjimky mohou být generovány následnou instrukcí.

Výjimky v režimu V86

Použije-li se prefix s jinou než vyjmenovanou instrukcí, generuje se #UD. Další výjimky mohou být generovány následnou instrukcí.

RDMSR *Read from Model Specific Register*



Popis: **CPL=0**

Instrukce RDMSR předává informace technického rázu, které poskytuje procesor Pentium. Instrukce čte registry Model Specific Registers (MSR) , které řídí testování, trasování, monitorování výkonnosti a kontrolu chyb procesoru.

Instrukce	Popis
RDMSR	podle ECX vrať hodnotu MSR do EDX&EAX

Operace

```
EDX&EAX := MSR[ECX];
```

Komentář

Hodnota v ECX určuje jeden z 64bitových registrů Model Specific Registers procesoru Pentium. Obsah se zkopíruje do dvojice EDX&EAX. Do EDX se uloží vyšší bity a do EAX nižší.

Pro procesor Pentium jsou zveřejněny tyto registry:

Číslo	**Jméno registru**	**Popis registru**
0	Machine Check Address	Uchovává adresy cyklu, který generoval výjimku.
1	Machine Check Type	Uchovává typ cyklu, který generoval výjimku.
0Eh	Test Register 12 (TR12)	Registr pro sledování předvídání skoků.

Informace o dalších registrech určených k testování a monitorování výkonu procesoru jsou popsány ve zvláštních dodatcích, které zveřejňuje firma Intel. Popis registrů 0 a 1 hledejte na str. 104. Popis registru 0Eh (TR12) hledejte na str. 156.

Má-li adresovaný MSR méně než 64 bitů, potom je hodnota ostatních bitů v EDX&EAX nedefinována. Čísla registrů 03h, 0Fh a čísla větší než 13h jsou rezervována. Tato čísla nepoužívejte.

Výjimky ve chráněném režimu

#GP(0) se generuje při pokusu o provedení instrukce na jiné úrovni oprávnění než 0, nebo hodnota v ECX neurčuje implementovaný MSR.

Výjimky v reálném režimu

#GP pokud hodnota v ECX neurčuje implementovaný MSR. Instrukci lze provést v reálném režimu.

Výjimky v režimu V86

#GP(0), instrukce je privilegovaná.

Poznámky k předchozím procesorům

Instrukce je podporována od procesoru Pentium.

WRMSR *Write to Model Specific Register*

O	D	I	T	S	Z	A	P	C

Popis: **CPL=0**

Instrukce WRMSR zapisuje do jednoho z Model Specific Registers, který řídí testování, trasování, monitorování výkonnosti a kontrolu chyb procesoru.

Instrukce	Popis
WRMSR	podle ECX zapiš hodnotu EDX&EAX do MSR

Operace

```
MSR[ECX] := EDX&EAX;
```

Komentář

Další podrobnosti viz RDMSR.

RSM *Resume from System Management Mode*

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Popis:

Instrukce RSM vrací řízení do programu přerušeného přerušením System Management Mode.

Instrukce	Popis
RSM	obnov běh přerušeného programu

Komentář

Instrukce obnoví stav procesoru podle informací, které byly uloženy při přepnutí do režimu System Management Mode. Instrukcí však není změněn obsah registrů Model Specific Register. Pokud se při obnovování původního stavu detekuje chyba, procesor se zastaví. Detekovány mohou být tyto chyby:

- hodnota uložená v poli „Báze záznamu o registrech“ není adresou zarovnanou na 32 KB,
- některý z rezervovaných bitů CR4 je nastaven na 1,
- nepovolená kombinace bitů v CR0 (PG=1 a PE=0) nebo (NW=1 a CD=0).

Další podrobnosti viz v kapitole 9.

Výjimky ve chráněném režimu

Při pokusu o zadání instrukce mimo System Management Mode se generuje #UD.

Výjimky v reálném režimu

Při pokusu o zadání instrukce mimo System Management Mode se generuje #UD.

Výjimky v režimu V86

Při pokusu o zadání instrukce mimo System Management Mode se generuje #UD.

Poznámky k předchozím procesorům

Instrukce je podporována od procesoru Pentium.

LDS/LES/LFS/LGS/LSS *Load Full Pointer*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce plní dvojici registrů. Jeden z dvojice je segmentový registr a druhý je jeden z všeobecných registrů.

Instrukce	Popis
LDS *r16,m16:16*	naplň DS:*r16* ukazatelem z paměti
LDS *r32,m16:32*	naplň DS:*r32* ukazatelem z paměti
LSS *r16,m16:16*	naplň SS:*r16* ukazatelem z paměti
LSS *r32,m16:32*	naplň SS:*r32* ukazatelem z paměti
LES *r16,m16:16*	naplň ES:*r16* ukazatelem z paměti
LES *r32,m16:32*	naplň ES:*r32* ukazatelem z paměti
LFS *r16,m16:16*	naplň FS:*r16* ukazatelem z paměti
LFS *r32,m16:32*	naplň FS:*r32* ukazatelem z paměti
LGS *r16,m16:16*	naplň GS:*r16* ukazatelem z paměti
LGS *r32,m16:32*	naplň GS:*r32* ukazatelem z paměti

Operace

```
CASE instrukce OF
  LSS: Sreg je SS; (* Naplň SS registr *)
  LDS: Sreg je DS; (* Naplň DS registr *)
  LES: Sreg je ES; (* Naplň ES registr *)
  LFS: Sreg je FS; (* Naplň FS registr *)
  LGS: Sreg je GS; (* Naplň GS registr *)
ESAC;
IF velikost operandu = 16
THEN
  r16 := [efektivní adresa]; (* 16bitový přenos *)
  Sreg := [efektivní adresa + 2]; (* 16bitový přenos *)
  (* v chráněném režimu ještě naplnění neviditelné části Sreg *)
ELSE (* velikost operandu je 32 bitů *)
  r32 := [efektivní adresa]; (* 32bitový přenos *)
  Sreg := [efektivní adresa + 4]; (* 16bitový přenos *)
  (* v chráněném režimu ještě naplnění neviditelné části Sreg *)
FI;
```

Komentář

Instrukce čtou hodnotu ukazatele z paměti z adresy zadané druhým operandem. Od této adresy jsou uloženy 2 nebo 4 slabiky offsetu a 2 slabiky segmentu. Ukazatelem se naplní dvojice registrů: segmentový a offsetový. Segmentový registr je určen jménem instrukce a offsetový registr je zadán prvním operandem. Typ prvního operandu (16/32 bitů) určuje, zda se offset uložený v paměti interpretuje jako 16 nebo 32bitový.

Součástí instrukce je také naplnění neviditelných částí segmentových registrů obsahem popisovače podle naplněného selektoru. Do registrů DS, ES, FS a GS může být zapsán i neplatný selektor. Při plnění segmentového registru se provádějí tyto akce:

```
IF plní se SS
THEN
  selektor není neplatný, jinak #GP(0);
  index selektoru ukazuje dovnitř limitu tabulky popisovačů,
        jinak #GP(selektor);
  RPL selektoru = CPL, jinak #GP(selektor);
  slabika AR indikuje zapisovatelný datový segment, jinak #GP(selektor);
  DPL z AR = CPL, jinak #GP(selektor);
  segment je přítomný, jinak #SS(selektor);
  naplň SS selektorem;
```

```
  naplň SS popisovačem;
FI;

IF plní se DS, ES, FS nebo GS platným selektorem
THEN
  index selektoru ukazuje dovnitř limitu tabulky popisovačů,
        jinak #GP(selektor);
  slabika AR indikuje datový nebo čitelný kódový segment,
        jinak #GP(selektor);
  IF datový nebo nepřizpůsobitelný instrukční segment
  THEN
    RPL a zároveň CPL <= DPL ze slabiky AR, jinak #GP(selektor);
  FI;
  segment je přítomný, jinak #NP(selektor);
  naplň segm. registr selektorem a bity RPL;
  naplň segm. registr popisovačem;
FI;

IF plní se DS, ES, FS nebo GS neplatným selektorem
THEN
  naplň segm. registr selektorem;
  vynuluj bit platnosti v nevidit. části;
FI;
```

Výjimky ve chráněném režimu

Pokud je SS plněn neplatným selektorem, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. Je-li druhý operand registr, generuje se #UD.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Je-li druhý operand registr, generuje se výjimka 6.

Výjimky v režimu V86

Stejné výjimky jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

ARPL — *Adjust RPL Field of Selector*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukce ARPL umožňuje nastavit pole RPL (bity 1 a 0) selektoru uloženého v prvním operandu na nižší úroveň oprávnění z obou zadaných operandů.

Instrukce	Popis
ARPL *r/m16,r16*	nastav RPL *r/m16* ne menší než RPL *r16*

Operace

```
IF RPL prvního_operandu < RPL druhého_operandu
THEN
  ZF := 1;
  RPL prvního_operandu := RPL druhého_operandu;
ELSE
  ZF := 0;
FI;
```

Komentář

Druhý operand obsahuje libovolnou hodnotu. Je-li hodnota RPL (bity 1 a 0) prvního operandu menší než hodnota bitů 1 a 0 druhého operandu, je nastaven příznak ZF na jedničku a pole RPL prvního operandu je zvýšeno na hodnotu nejnižších dvou bitů druhého operandu. Jinak je příznak ZF nulován a první operand zůstane nezměněn.

Instrukce ARPL je určena pro operační systém, ne pro aplikační programy. Zajistí, aby selektor jako parametr podprogramu nevyžadoval vyšší úroveň oprávnění, než dovoluje volající. Druhým operandem instrukce zpravidla bývá registr CS obsahující CPL volajícího.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Při použití instrukce v reálném režimu se generuje výjimka 6, protože instrukce není rozpoznána.

Výjimky v režimu V86

Při použití instrukce ve V86 režimu se generuje výjimka 6, protože instrukce není rozpoznána.

CLTS — *Clear Task Switched Flag*

Popis: **CPL=0**

O	D	I	T	S	Z	A	P	C

Instrukce CLTS nuluje bit TS (Task Switch) registru CR0 (nebo MSW).

Instrukce	Popis
CLTS	vynuluj bit Task Switch v CR0

Operace

```
TS v CR0 := 0;
```

Komentář

Bit TS nastavuje procesor při každém přepnutí procesu a má význam pro spolupráci procesoru s koprocesory (např. FPU) následovně:

- Před provedením každé instrukce ESC se generuje trap v případě, že je nastaven bit TS.
- Před provedením každé instrukce WAIT se generuje trap v případě, že je nastaven bit MP a současně bit TS.

Význam spočívá v zaregistrování změny kontextu procesoru a provedení podobných změn v koprocesoru. Jakmile se přepne proces po zahájení provádění ESC, je potřebné uložit kontext v koprocesoru. To stačí provést těsně před zahájením další instrukce ESC. Tuto operaci provede rutina obsluhující přerušení.
CLTS je privilegovaná instrukce, kterou v chráněném režimu lze provést pouze s nejvyšší úrovní oprávnění (CPL=0).

Výjimky ve chráněném režimu

Při pokusu o provedení CLTS na jiné úrovni oprávnění než 0 se generuje #GP(0).

Výjimky v reálném režimu

Žádné. (Instrukce je v reálném režimu povolena.)

Výjimky v režimu V86

#GP(0); instrukce je privilegovaná.

Poznámky k předchozím procesorům

Registr MSW byl zaveden v procesoru Intel286.

INVD *Invalidate Cache*

Popis: **CPL=0**

O	D	I	T	S	Z	A	P	C

Instrukce prohlásí za neplatný obsah celé interní vyrovnávací paměti procesoru a signalizuje případné externí vyrovnávací paměti, že má udělat totéž.

Instrukce	Popis
INVD	prohlaš obsah IVP za neplatný

Operace

```
prohlaš obsah interní vyrovnávací paměti za neplatný;
signalizace externí vyrovnávací paměti, že se má udělat totéž;
```

Komentář

Instrukce se musí používat opatrně, protože nezapisuje do paměti změněné položky interní vyrovnávací paměti. Vyprázdnění (tj. zápis změněných položek do paměti a zneplatnění obsahu) lze provést instrukcí WBINVD.
Instrukce nečeká na dokončení zneplatňování obsahu externí vyrovnávací paměti. Činnost této instrukce může být v budoucích typech procesorů změněna.

Výjimky ve chráněném režimu

Instrukce je privilegovaná, při pokusu o její provedení na jiné úrovni oprávnění než 0 se generuje #GP(0).

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

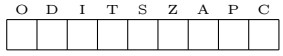
#GP(0); instrukce je privilegovaná.

Poznámky k předchozím procesorům

Instrukce byla poprvé zavedena v procesoru Intel486. V Intel486 byla totožná s WBINVD, protože všechny zápisy do IVP byly ihned zapisovány i do paměti (Write-through).

WBINVD *Write-Back and Invalidate Cache*

Popis: **CPL=0**



Instrukce vyprázdní obsah celé interní vyrovnávací paměti procesoru a signalizuje případné externí vyrovnávací paměti, že má udělat totéž. Operace vyprázdnění zahrnuje poznačení změněných (Write-back) položek do paměti a zneplatnění obsahu.

Instrukce	Popis
WBINVD	vyprázdni IVP

Operace

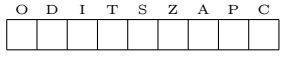
```
vyprázdni interní vyrovnávací paměť s uložením změněných položek;
signalizace externí vyrovnávací paměti, že se má udělat zpětný zápis;
signalizace externí vyrovnávací paměti, že se má zneplatnit obsah;
```

Komentář

Další podrobnosti viz INVD.

INVLPG *Invalidate TLB Entry*

Popis: **CPL=0**



Instrukce INVLPG zneplatňuje jednu položku TLB. Položka (její klíčová část) je určena operandem instrukce. Pokud je položka nalezena, je jí příslušný bit V vynulován.

Instrukce	Popis
INVLPG m	zneplatni položku TLB

Operace

```
zneplatni odpovídající položku TLB;
```

Výjimky ve chráněném režimu

Instrukce je privilegovaná, při pokusu o její provedení na jiné úrovni oprávnění než 0 se generuje #GP(0). Je-li operandem registr, generuje se #UD.

Výjimky v reálném režimu

Žádné.

Výjimky v režimu V86

#GP(0); instrukce je privilegovaná.

Poznámky k předchozím procesorům

Instrukce byla poprvé zavedena v procesoru Intel486.

LAR *Load Access Rights Byte*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukce LAR naplní první operand obsahem slabiky *přístupová práva* popisovače segmentu, který je adresován selektorem uloženým ve druhém operandu.

Instrukce	Popis
LAR *r16,r/m16*	*r16* := *r/m16* maskováno FF00
LAR *r32,r/m32*	*r32* := *r/m32* maskováno 00FxFF00

Komentář

Před naplněním registru prvního operandu se kontroluje, nepřekračuje-li selektor limit GDT nebo LDT a je-li DPL popisovače větší nebo rovné EPL (tj. maximu CPL a RPL ze zdrojového operandu). Jsou-li tyto podmínky splněny, naplní se horní slabika prvního operandu slabikou přístupových práv popisovače, nuluje se dolní slabika a nastaví se ZF=1. Není-li podmínka splněna, nastaví se ZF=0.

Jsou-li použity 16bitové registry, ukládá se slabika přístupových práv do horní poloviny slova. Ve 32bitové variantě se ukládá celé druhé dvojslovo popisovače maskované hodnotou 00FxFF00, kde x znamená, že příslušné 4 bity jsou nedefinovány.

Instrukce umí pracovat se všemi instrukčními a datovými segmenty. Následující tabulka shrnuje povolené typy popisovačů:

Typ	Název popisovače	Povolené ?
0	nepoužito	nepovolené
1	16bitové neaktivní TSS	povolené
2	LDT	povolené
3	16bitové aktivní TSS	povolené
4	16bitová brána pro pro předání řízení	povolené
5	brána zpřístupňující TSS	povolené
6	16bitová brána pro maskující přerušení (trap)	nepovolené
7	16bitová brána pro nemaskující přerušení	nepovolené
8	nepoužito	nepovolené
9	32bitové neaktivní TSS	povolené
A	nepoužito	nepovolené
B	32bitové aktivní TSS	povolené
C	32bitová brána pro předání řízení	povolené
D	nepoužito	nepovolené
E	32bitová brána pro maskující přerušení (trap)	nepovolené
F	32bitová brána pro nemaskující přerušení	nepovolené

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Výjimka 6; instrukce není v reálném režimu rozpoznána.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu.

LSL *Load Segment Limit*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukce LSL naplní první operand limitem segmentu, jehož selektor je uložen ve druhém operandu, a nastaví příznak ZF na jedničku tehdy, je-li přístup k segmentu

povolen (CPL má odpovídající úroveň oprávnění vzhledem k DPL segmentu). Jsouli přístupem k segmentu porušena přístupová práva, je ZF nulován a obsah prvního operandu zůstane nezměněn.

Instrukce	Popis
LSL *r16,r/m16*	*r16* := limit pro selektor *r/m16*
LSL *r32,r/m32*	*r32* := limit pro selektor *r/m32*

Komentář

Limit je do registru ukládán vždy s jednotkou 1 B (nikoli 4 KB). Pokud je limit v popisovači uložen s jednotkou 4 KB, instrukce jej převede na jednotku 1 B následovně: hodnotu posune doleva o 12 bitů a k výsledku logicky přičte hodnotu 00000FFFh.

Do 32bitového registru se ukládá všech 32 bitů limitu rozbaleného na jednotku 1 B. Do 16bitového registru se opět ukládá rozbaleně s tím, že horních 16 bitů se ignoruje. Další podrobnosti viz LAR.

LGDT
LIDT

Load Global Descriptor Table Register
Load Interrupt Descriptor Table Register

Popis: **CPL=0**

O	D	I	T	S	Z	A	P	C

Instrukce LGDT (LIDT) plní obsah registru GDTR (IDTR) limitem a bází segmentu, v němž je uložena tabulka GDT (IDT). Registr se plní obsahem operandu, který má vždy délku 6 slabik. První dvě slabiky musejí obsahovat 16bitový limit segmentu a tři (při 16bitovém atributu velikosti operandu) nebo čtyři (při 32bitovém atributu velikosti operandu) následující slabiky obsahují bázi segmentu.

Instrukce	Popis
LGDT *m16&32*	naplň GDTR hodnotou *m*
LIDT *m16&32*	naplň IDTR hodnotou *m*

Operace

```
IF instrukce = LIDT
THEN
  IF velikost operandu = 16
  THEN IDTR.Limit:Báze := m16:24 (* 24 bitů báze *)
  ELSE IDTR.Limit:Báze := m16:32 (* 32 bitů báze *)
```

```
  FI;
ELSE (* instrukce = LGDT *)
  IF velikost operandu = 16
  THEN GDTR.Limit:Báze := m16:24 (* 24 bitů báze *)
  ELSE GDTR.Limit:Báze := m16:32 (* 32 bitů báze *)
  FI;
FI;
```

Výjimky ve chráněném režimu

Při pokusu o provedení instrukce na jiné úrovni oprávnění než 0 se generuje #GP(0). Je-li operand registr, generuje se #UD.

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek).

Výjimky v reálném režimu

Instrukce je v reálném režimu povolena. Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky.

SGDT — *Store Global Descriptor Table Register*
SIDT — *Store Interrupt Descriptor Table Register*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce SGDT (SIDT) uloží obsah registru GDTR (IDTR) do operandu délky 6 slabik. První dvě slabiky jsou naplněny 16bitovým limitem segmentu, následující tři (při 16bitovém atributu velikosti operandu) nebo čtyři (při 32bitovém atributu velikosti operandu) slabiky bází segmentu. Při 16bitovém atributu velikosti operandu je obsah poslední slabiky báze nedefinován.

Instrukce	Popis
SGDT *m16&32*	ulož GDTR do paměti
SIDT *m16&32*	ulož IDTR do paměti

Operace

```
IF instrukce = SIDT
THEN
  IF velikost operandu = 16
  THEN  m16:24 := IDTR.Limit:Báze (* 24 bitů báze *)
  ELSE  m16:32 := IDTR.Limit:Báze (* 32 bitů báze *)
  FI;
ELSE (* instrukce = SGDT *)
  IF velikost operandu = 16
  THEN m16:24 := GDTR.Limit:Báze (* 24 bitů báze *)
  ELSE m16:32 := GDTR.Limit:Báze (* 32 bitů báze *)
  FI;
FI;
```

Výjimky ve chráněném režimu

Je-li operand registr, generuje se #UD. Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Instrukce je v reálném režimu povolena. Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky, při nezarovnaném přístupu k paměťovému operandu se generuje #AC.

LLDT *Load Local Descriptor Table Register*

O	D	I	T	S	Z	A	P	C

Popis: **CPL=0**

Privilegovaná instrukce LLDT naplní registr LDTR obsahem operandu. 16bitový operand obsahuje selektor ukazující do GDT, v němž je uložen popisovač segmentu LDT. Z popisovače je automaticky naplněna neviditelná část LDTR bází a limitem segmentu tabulky LDT.

Instrukce	Popis
LLDT *r/m16*	naplň LDTR selektorem

Operace

```
LDTR := r/m16; (* naplnění selektorem *)
naplň neviditelnou část LDTR popisovačem;
```

Komentář

Jestliže operand obsahuje neplatný (nulový) selektor, je registr LDTR označen jako neplatný a všechny instrukce odvolávající se na lokální adresový prostor přes tento registr (vyjma LAR, VERR, VERW, VERW nebo LSL) vyvolají #GP. Atribut velikosti operandu nemá na činnost instrukce vliv.

Výjimky ve chráněném režimu

Při pokusu o provedení instrukce na jiné úrovni oprávnění než 0 se generuje #GP(0). Je-li operand registr, generuje se #UD. Neukazuje-li selektor do GDT nebo není-li položka popisovač LDT, generuje se #GP(selektor). #NP(selektor) se generuje při nepřítomnosti segmentu s LDT.

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek).

Výjimky v reálném režimu

Výjimka 6, instrukce není v reálném režimu rozpoznána.

Výjimky v režimu V86

Výjimka 6, instrukce není v reálném režimu povolena.

SLDT *Store Local Descriptor Table Register*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce SLDT uloží 16bitový obsah viditelné části registru LDTR (selektor) do operandu. Selektor ukazuje do GDT.

Instrukce	Popis
SLDT *r/m16*	ulož slektor LDTR do operandu

Operace

```
r/m16 := LDTR; (* naplnění selektorem *)
```

Komentář

Je-li operand 32bitový registr, je výsledek zapsán do dolních 16 bitů a horních 16 bitů je nedefinovaných. Do paměti se zapisuje vždy právě 16 bitů.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Výjimka 6, instrukce není v reálném režimu rozpoznána.

Výjimky v režimu V86

Výjimka 6, instrukce není v reálném režimu povolena.

LMSW *Load Machine Status Word*

Popis: **CPL=0**

O	D	I	T	S	Z	A	P	C

Instrukce LMSW naplní MSW hodnotou uloženou v operandu. Označení MSW je dodržováno pro kompatibilitu s procesorem Intel286. Ve vyšších typech procesorů MSW představuje dolní slovo registru CR0.

Instrukce	Popis
LMSW *r/m16*	naplň MSW operandem

Operace

```
MSW := r/m16 (* 16 bitů uloženo do MSW *)
```

Komentář

Instrukce smí být v reálném režimu použita pro přepnutí do chráněného režimu. V tomto případě musí za instrukcí LMSW následovat instrukce blízkého skoku pro vyprázdnění fronty předzpracovaných instrukcí.

Pomocí instrukce LMSW nelze přepnout procesor zpět do reálného režimu (opět pro kompatibilitu s Intel286) a nelze jí nastavovat bity PG, ET a NE. Na činnost instrukce nemá vliv nastavení atributu velikosti operandu. Od procesoru Intel386 je k dispozici instrukce MOV CR0, která plní celý registr CR0.

Výjimky ve chráněném režimu

Při pokusu o provedení instrukce na jiné úrovni oprávnění než 0 se generuje #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek).

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

#GP(0), instrukce je privilegovaná.

Poznámky k předchozím procesorům

Registr MSW byl zaveden v procesoru Intel286.

SMSW *Store Machine Status Word*

Popis:

O	D	I	T	S	Z	A	P	C

Instrukce SMSW ukládá obsah registru MSW do operandu. MSW je dolní slovo CR0 (viz též LMSW).

Instrukce	Popis
SMSW *r/m16*	ulož MSW do operandu

Operace

```
r/m16 := MSW (* 16 bitů uloženo do r/m *)
```

Komentář

Instrukci lze provést v reálném režimu a na libovolné úrovni oprávnění chráněného režimu.

Je-li operand 32bitový registr, je výsledek zapsán do dolních 16 bitů a horních 16 bitů je nedefinovaných. Do paměti se zapisuje vždy právě 16 bitů.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Poznámky k předchozím procesorům

Registr MSW byl zaveden v procesoru Intel286.

LTR — *Load Task Registr*

Popis: **CPL=0**

O	D	I	T	S	Z	A	P	C

Privilegovaná instrukce LTR naplní registr TR obsahem operandu (selektorem). Příslušný TSS je tím označen jako právě aktivní. Instrukce však nevyvolá přepnutí procesu.

Instrukce	Popis
LTR *r/m16*	TR := selektor z *r/m16*

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek).

Při pokusu o provedení na jiné úrovni oprávnění než 0 se generuje #GP(0). #GP(selektor) se vyvolá tehdy, neukazuje-li selektor na neaktivní TSS. #NP(selektor) se generuje, není-li segment s TSS přítomný.

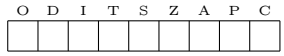
Výjimky v reálném režimu

Výjimka 6; instrukce není v reálném režimu rozpoznána.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu.

STR *Store Task Register*



Popis:

Instrukce STR ukládá obsah registru TR (selektor) do operandu.

Instrukce	Popis
STR *r/m16*	*r/m16* := selektor z TR

Komentář

Je-li operand 32bitový registr, je výsledek zapsán do dolních 16 bitů a horních 16 bitů je nedefinovaných. Do paměti se zapisuje vždy právě 16 bitů.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Výjimka 6; instrukce není v reálném režimu rozpoznána.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu.

MOV *Move to/from Control Registers*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Popis: **CPL=0**

Instrukce plní nebo ukládá řídicí registry CR0, CR2, CR3 a CR4 z nebo do libovolného všeobecného registru.

Instrukce	Popis
MOV CR*x*,*r32*	CR*x* := *r32*
MOV *r32*,CR*x*	*r32* := CR*x*

Komentář

Bez ohledu na velikost operandu se vždy přenáší 32 bitů. Číslo řídicího registru je zapsáno v poli **reg** slabiky ModR/M. Pole **mod** obsahuje hodnotu 11. Pole **r/m** identifikuje všeobecný registr.

Výjimky ve chráněném režimu

Při pokusu o provedení instrukce na jiné úrovni oprávnění než 0 se generuje #GP(0). Totéž se generuje při pokusu nastavit na 1 rezervované bity v CR4.

Výjimky v reálném režimu

Výjimka 13 při pokusu nastavit na 1 rezervované bity v CR4.

Výjimky v režimu V86

#GP(0) při pokusu o provedení instrukce.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel386.

MOV *Move to/from Debug Registers*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Popis: **CPL=0**

Instrukce plní nebo ukládá ladicí registry DR*i* z nebo do libovolného všeobecného registru.

Instrukce	Popis
MOV DR*x*,*r32*	DR*x* := *r32*
MOV *r32*,DR*x*	*r32* := DR*x*

Operace

```
IF ((DE = 1) AND (operand je DR4 nebo DR5))
THEN
  #UD;
ELSE
  první operand := druhý operand;
```

Komentář

Bez ohledu na velikost operandu se vždy přenáší 32 bitů. Číslo řídicího registru je zapsáno v poli **reg** slabiky ModR/M. Pole **mod** obsahuje hodnotu 11. Pole **r/m** identifikuje všeobecný registr.

Je-li bit DE (Debugging Extensions) v registru CR4 nulový, potom ladicí systém pracuje kompatibilně s procesory Intel386 a Intel486 (mj. lze používat registry DR4 a DR5). Je-li DE nastaven, generuje se při přístupu k DR4 nebo DR5 výjimka #UD.

Výjimky ve chráněném režimu

Při pokusu o provedení instrukce na jiné úrovni oprávnění než 0 se generuje #GP(0). Při použití DR4 nebo DR5 současně s DE=0 se generuje #UD.

Výjimky v reálném režimu

Při použití DR4 nebo DR5 současně s DE=0 se generuje #UD.

Výjimky v režimu V86

#GP(0) při pokusu o provedení instrukce.

Poznámky k předchozím procesorům

Instrukce je zavedena od procesoru Intel386.

VERR
VERW

Verify a Segment for Reading
Verify a Segment for Writing

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukce VERR (VERW) kontroluje, zda současná úroveň oprávnění procesu (CPL) je dostatečná pro čtení (zápis) segmentu, jehož selektor je uložen v operandu instrukce. Nebudou-li čtením (zápisem) specifikovaného segmentu porušena přístupová práva, je příznak ZF nastaven na jedničku. Při porušení přístupových práv je

ZF vynulován. Instrukce provádí stejné kontroly jako při plnění registrů DS, ES, FS a GS.

Instrukce	Popis
VERR *r/m16*	kontrola selektoru *r/m16*
VERW *r/m16*	kontrola selektoru *r/m16*

Operace

```
IF segment o selektoru r/m16 je přístupný na současné úrovni oprávnění
     AND ((segment je čitelný při VERR)
                OR (segment je zapisovatelný při VERW))
THEN ZF := 1;
ELSE ZF := 0;
FI;
```

Komentář

Instrukce kontroluje splnění těchto podmínek: popisovač GDT nebo LDT vybraný selektorem musí být uvnitř limitu segmentu příslušné tabulky, popisovač musí ukazovat na datový nebo instrukční segment (nikoli na systémový segment s TSS, LDT nebo bránu), při použití instrukce VERR musí být povoleno čtení segmentu (při VERW zápis), musí být splněno DPL≥Max(CPL,RPL) (pouze v případě, že se jedná o instrukční segment s C=1, je tato podmínka splněna vždy).

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

Výjimky v reálném režimu

Při použití instrukce v reálném režimu se generuje výjimka 6, protože instrukce není rozpoznána.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. Při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC.

12.17 Instrukce FPU

F2XM1

Compute $2^x - 1$

Instrukce nahradí obsah ST hodnotou $(2^{ST} - 1)$. ST musí být v intervalu $-1 <$ST< 1.

Instrukce	Popis
F2XM1	nahradí obsah ST hodnotou $(2^{ST} - 1)$

Operace

ST := $(2^{ST} - 1)$

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Leží-li operand mimo povolený rozsah, je výsledek instrukce nedefinován.

FABS *Absolute Value*

Instrukce vynuluje znaménkový bit ST. Kladnou hodnotu ponechá beze změny a zápornou změní na kladnou.

Instrukce	Popis
FABS	nahraď ST absolutní hodnotou

Operace

```
znaménkový bit ST := 0;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FADD/FADDP/FIADD *Add*

Instrukce přičte k cílovému operandu obsah zdrojového.

Instrukce	Popis
FADD *m32real*	Přičti *m32real* k ST
FADD *m64real*	Přičti *m64real* k ST
FADD ST,ST(i)	Přičti ST(i) k ST
FADD ST(i),ST	Přičti ST k ST(i)
FADDP ST(i),ST	Přičti ST k ST(i) a Pop ST
FADDP	Přičti ST k ST(1) a Pop ST
FIADD *m32int*	Přičti *m32int* k ST
FIADD *m16int*	Přičti *m16int* k ST

Operace

```
cíl := cíl + zdroj;
IF instrukce = FADDP THEN Pop ST; FI;
```

Komentář

Operand na vrcholu zásobníku lze zdvojit instrukcí FADD ST,ST(0)

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, D, I, IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát.

FBLD *Load Binary Coded Decimal*

Instrukce převede zdrojový operand v BCD na extended-real formát a vloží jej do zásobníku.

Instrukce	Popis
FBLD *m80dec*	vlož *m80dec* do zásobníku

Operace

```
sniž ukazatel vrcholu zásobníku;
ST(0) := operand;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce nekontroluje, zda vstupní hodnota neobsahuje nepovolené číslice (tj. mimo 0-9). Pokud ano, bude hodnota nedefinována. Položka ST(7) musí být prázdná, jinak se generuje výjimka.

FBSTP *Store Binary Coded Decimal and Pop*

Instrukce převede hodnotu v ST na zhuštěný BCD kód, uloží ji do paměti a vybere ST ze zásobníku.

Instrukce	Popis
FBSTP *m80dec*	ulož ST do paměti a vyber ST ze zásobníku

Operace

```
cíl := ST(0);
pop ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

FCHS *Change Sign*

Instrukce invertuje znaménko ST. Kladné změní na záporné a obráceně.

Instrukce	Popis
FCHS	změň znaménko ST

Operace

```
znam. bit ST := NOT znam. bit ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FCLEX/FNCLEX *Clear Exceptions*

Instrukce nuluje příznaky ze stavového registru FPU (viz též str. 39).

Instrukce	Popis
FCLEX	zkontroluj chybové podmínky FP a potom smaž příznaky
FNCLEX	smaž příznaky bez kontroly

Operace

```
SW[0..7] := 0;
SW[15] := 0;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FCOM/FCOMP/FCOMPP *Compare Real*

Instrukce porovnává dvě reálná čísla: číslo na vrcholu zásobníku a operand, kterým může být registr nebo paměťový operand.

Instrukce	Popis
FCOM *m32real*	Porovnej ST s *m32real*
FCOM *m64real*	Porovnej ST s *m64real*
FCOM ST(i)	Porovnej ST s ST(i)
FCOM	Porovnej ST s ST(1)
FCOMP *m32real*	Porovnej ST s *m32real* a pop ST
FCOMP *m64real*	Porovnej ST s *m64real* a pop ST
FCOMP ST(i)	Porovnej ST s ST(i) a pop ST
FCOMP	Porovnej ST s ST(1) a pop ST
FCOMPP	Porovnej ST s ST(1) a pop ST dvakrát

Operace

```
CASE vztah operandů OF
  neporovnatelné: C3,C2,C0 := 111;
  ST > operand: C3,C2,C0 := 000;
  ST < operand: C3,C2,C0 := 001;
  ST = operand: C3,C2,C0 := 100;
ESAC;
IF instrukce je FCOMP THEN pop ST; FI;
IF instrukce je FCOMPP THEN pop ST; pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 viz výše.

Příznak FPU	EFlags
C0	CF
C1	žádný
C2	PF
C3	ZF

Numerické výjimky

D, I, IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

FCOS *Cosine*

Instrukce nahradí obsah ST hodnotou *cos*(ST). ST vyjádřené v radiánech musí být v intervalu $|\mathrm{ST}| < 2^{63}$.

Instrukce	Popis
FCOS	Nahraď ST hodnotou *cos*(ST)

Operace

```
IF operand je v rozmezí
THEN
  C2 := 0;
  ST := cos(ST);
ELSE
  C2 := 1;
FI;
```

Modifikované příznaky FPU

C1, C2 podle popisu na str. 35. C0 a C3 jsou nedefinovány.

Numerické výjimky

P, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Pokud je operand mimo stanovený rozsah, nastaví se příznak C2 a ST zůstává nezměněno. Programátor musí vstupní hodnotu redukovat vydělením příslušným násobkem 2π.

FDECSTP *Decrement Stack-Top Pointer*

Instrukce sníží o jedničku 3bitový ukazatel vrcholu zásobníku ve stavovém slově FPU.

Instrukce	Popis
FDECSTP	sniž ukazatel vrcholu zásobníku FPU

Operace

```
IF ukazatel = 0
THEN ukazatel := 7;
ELSE ukazatel := ukazatel - 1;
FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Význam instrukce spočívá v rotování zásobníku. Instrukce nemodifikuje jeho obsah.

FDIV/FDIVP/FIDIV *Divide*

Instrukce dělí obsah vrcholu zásobníku operandem a podílem nahradí vrchol zásobníku.

Instrukce	Popis
FDIV *m32real*	Vyděl ST:=ST/*m32real*
FDIV *m64real*	Vyděl ST:=ST/*m64real*
FDIV ST,ST(i)	Vyděl ST:=ST/ST(i)
FDIV ST(i),ST	Vyděl ST(i):=ST(i)/ST
FDIVP ST(i),ST	Vyděl ST(i):=ST(i)/ST a Pop ST
FDIVP	Vyděl ST(1):=ST(1)/ST a Pop ST
FIDIV *m32int*	Vyděl ST:=ST/*m32int*
FIDIV *m16int*	Vyděl ST:=ST/*m16int*

Operace

```
FDIV cíl,zdroj
cíl := cíl / zdroj;
IF instrukce je FDIVP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, Z, D, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát. Trvání instrukce závisí na hodnotě pole PC (Precision Control). Čím je nastavena vyšší přesnost, tím instrukce déle trvá. Při přesnosti 53 bitů trvá 33 tiků hodin, při přesnosti 24 bitů trvá 19 tiků.

FDIVR/FDIVRP/FIDIVR *Divide*

Instrukce dělí operand obsahem vrcholu zásobníku a podíl uloží do cílového operandu.

Instrukce	Popis
FDIVR *m32real*	Vyděl ST:=*m32real*/ST
FDIVR *m64real*	Vyděl ST:=*m64real*/ST
FDIVR ST,ST(i)	Vyděl ST:=ST(i)/ST
FDIVR ST(i),ST	Vyděl ST(i):=ST/ST(i)
FDIVRP ST(i),ST	Vyděl ST(i):=ST/ST(i) a Pop ST
FDIVRP	Vyděl ST(1):=ST/ST(1) a Pop ST
FIDIVR *m32int*	Vyděl ST:=*m32int*/ST
FIDIVR *m16int*	Vyděl ST:=*m16int*/ST

Operace

```
FDIVR cíl,zdroj
cíl := zdroj / cíl;
IF instrukce je FDIVRP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, Z, D, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát. Trvání instrukce závisí na hodnotě pole PC (Precision Control). Čím je nastavena vyšší přesnost, tím instrukce déle trvá. Při přesnosti 53 bitů trvá 33 tiků hodin, při přesnosti 24 bitů trvá 19 tiků.

FFREE *Free FP Register*

Instrukce označí registr jako prázdný.

Instrukce	Popis
FFREE ST(i)	

Operace

```
TAG(i) := 11b;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce nemodifikuje obsah registru a nemodifikuje nastavení vrcholu zásobníku.

FICOM/FICOMP *Compare Integer*

Instrukce porovnává dvě celá čísla: číslo na vrcholu zásobníku a operand, kterým může být registr nebo paměťový operand.

Instrukce	Popis
FICOM *m16int*	Porovnej ST s *m16int*
FICOM *m32int*	Porovnej ST s *m32int*
FICOMP *m16int*	Porovnej ST s *m16int* a pop ST
FICOMP *m32int*	Porovnej ST s *m32int* a pop ST

Operace

```
CASE vztah operandů OF
  neporovnatelné: C3,C2,C0 := 111;
  ST > operand: C3,C2,C0 := 000;
  ST < operand: C3,C2,C0 := 001;
  ST = operand: C3,C2,C0 := 100;
ESAC;
IF instrukce je FICOMP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 viz výše.

Příznak FPU	**EFlags**
C0	CF
C1	žádný
C2	PF
C3	ZF

Numerické výjimky

D, I, IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát.

FILD *Load Integer*

Instrukce konvertuje operand z celého čísla se znaménkem na extended-real a uloží jej do zásobníku FPU.

Instrukce	Popis
FILD *m16int*	vlož *m16int* do zásobníku FPU
FILD *m32int*	vlož *m32int* do zásobníku FPU
FILD *m64int*	vlož *m64int* do zásobníku FPU

Operace

```
sniž ukazatel vrcholu zásobníku FPU;
ST(0) := operand;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

ST(7) musí být před provedením instrukce prázdný.

FINCSTP *Increment Stack-Top Pointer*

Instrukce zvýší o jedničku 3bitový ukazatel vrcholu zásobníku ve stavovém slově FPU.

Instrukce	Popis
FINCSTP	zvyš ukazatel vrcholu zásobníku FPU

Operace

```
IF ukazatel = 7
THEN ukazatel := 0;
ELSE ukazatel := ukazatel + 1;
FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Význam instrukce spočívá v rotování zásobníku. Instrukce nemodifikuje jeho obsah.

FINIT/FNINIT *Initialize FPU*

Instrukce uvede FPU do počátečního stavu.

Instrukce	Popis
FINIT	Inicializace po kontrole na chybové podmínky FPU
FNINIT	Inicializace bez kontroly

Operace

```
CW := 037Fh; (* Řídicí slovo *)
SW := 0; (* Stavové slovo *)
TW := FFFFh; (* Tag slovo *)
FEA := 0; FDS := 0; (* Ukazatele dat *)
FIP := 0; FOP := 0; FCS := 0; (* Ukazatele instrukcí *)
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné;

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámky k předchozím procesorům

V Pentiu instrukce FINIT a FNINIT nulují, na rozdíl od Intel387 a Intel486, také chybové ukazatele.

FIST/FISTP *Store Integer*

Instrukce konvertuje vrchol zásobníku ve formátu extended-real na celé číslo se znaménkem na extended-real a uloží jej do operandu.

Instrukce	Popis
FIST *m16int*	ulož ST do *m16int*
FIST *m32int*	ulož ST do *m32int*
FISTP *m16int*	ulož ST do *m16int* a pop ST
FISTP *m32int*	ulož ST do *m32int* a pop ST
FISTP *m64int*	ulož ST do *m64int* a pop ST

Operace

```
ST(0) := operand;
IF instrukce je FISTP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

ST(7) musí být před provedením instrukce prázdný.

FLD *Load Real*

Instrukce vloží operand do zásobníku FPU.

Instrukce	Popis
FLD *m32real*	vlož *m32real* do zásobníku FPU
FLD *m64real*	vlož *m64real* do zásobníku FPU
FLD *m80real*	vlož *m80real* do zásobníku FPU

Operace

```
sniž ukazatel vrcholu zásobníku FPU;
ST(0) := operand;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

D, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Pokud je operandem registr FPU, je jeho číslo provedením instrukce sníženo o jedničku. Instrukce FLD ST(0) zdvojí hodnotu na vrcholu zásobníku. ST(7) musí být před provedením instrukce prázdný. Ukládaná hodnota se konvertuje do formátu extended-real.

FLD1/FLDL2T/FLDL2E/FLDPI/ FLDLG2/FLDLN2/FLDZ *Load Constant*

Instrukce vloží do zásobníku FPU konstantu.

Instrukce	Popis
FLD1	vlož $+1,0$
FLDL2T	vlož $\log_2 10$
FLDL2E	vlož $\log_2 e$
FLDPI	vlož π
FLDLG2	vlož $\log_{10} 2$
FLDLN2	vlož $\log_e 2$
FLDZ	vlož $+0,0$

Operace

```
sniž ukazatel vrcholu zásobníku FPU;
ST(0) := konstanta;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

ST(7) musí být před provedením instrukce prázdný.

FLDCW *Load Control Word*

Instrukce přepíše obsah řídicího slova FPU hodnotou paměťového operandu.

Instrukce	Popis
FLDCW *m2byte*	naplň řídicí slovo FPU

Operace

```
řídicí slovo FPU := operand;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

FLDENV *Load FPU Environment*

Naplní FPU jeho prostředím (Environment) z bloku slabik v paměti, který byl již dříve zapsán instrukcí FSTENV nebo FNSTENV.

Instrukce	Popis
FLDENV *m14/28byte*	naplní FPU jeho prostředím

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou naplněny.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Blok prostředí FPU, kterým se FPU plní, obsahuje řídicí slovo, stavové slovo, doplňující slovo a datový a instrukční chybový ukazatel. Velikost operandu je určena hodnotou atributu velikosti operandu pro daný instrukční segment.

FMUL/FMULP/FIMUL *Multiply*

Instrukce vynásobí cílový operand obsahem zdrojového.

Instrukce	Popis
FMUL *m32real*	ST:=ST**m32real*
FMUL *m64real*	ST:=ST**m64real*
FMUL ST,ST(i)	ST:=ST*ST(i)
FMUL ST(i),ST	ST(i):=ST(i)*ST
FMULP ST(i),ST	ST(i):=ST(i)*ST a Pop ST
FMULP	ST(1):=ST(1)*ST a Pop ST
FIMUL *m32int*	ST:=ST**m32int*
FIMUL *m16int*	ST:=ST**m16int*

Operace

```
cíl := cíl * zdroj;
IF instrukce = FMULP THEN Pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, D, I.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát.

FNOP *No Operation*

Instrukce nemění stav FPU.

Instrukce	Popis
FNOP	Žádná akce.

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FPATAN *Partial Arctangent*

Instrukce vypočítá hodnotu arkustangentu podílu ST(1)/ST a výsledek v radiánech uloží do ST(1) a potom provede pop ST. Výsledek má stejné znaménko jako operand v ST(1).

Instrukce	Popis
FPATAN	Nahraď ST(1) hodnotou arctan(ST(1)/ST) a pop ST.

Operace

```
ST(1) := arctan(ST(1)/ST);
pop ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce nemá omezení na hodnotu argumentů. Instrukci lze použít i na realizaci jiných funkcí: např. $\arcsin(x)$ je $\arctan(x/\sqrt{1-x^2})$.

FPREM *Partial Remainder*

Instrukce vypočte zbytek po dělení ST/ST(1) a výsledek uloží do ST. Znaménko zbytku je stejné jako znaménko dělence v ST.

Instrukce	Popis
FPREM	ST := zbytek po dělení ST/ST(1)

Operace

```
EXPDIV := exponent(ST) - exponent(ST(1));
IF EXPDIV < 64
THEN
  Q := celá část výsledku oříznutím desetinné (ST/ST(1));
  ST := ST - (ST(1)*Q);
  C2 := 0;
  C0,C1,C3 := tři bity nejnižšího řádu z Q; (* Q2,Q1,Q0 *)
ELSE
  C2 := 1:
  N := hodnota mezi 32 a 63;
  QQ := celá část výsledku oříznutím desetinné ((ST/ST(1))/2^(EXPDIF-N));
  ST := ST - (ST(1)*QQ*2^(EXPDIF-N);
FI;
```

Modifikované příznaky FPU

C0, C1, C2, C3 podle popisu na str. 35.

Numerické výjimky

U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce nepracuje podle normy IEEE Std 754. Podle této normy pracuje instrukce FPREM1. Instrukce FPREM se zachovává pro kompatibilitu s 8087 a Intel287.

Pokud instrukce dosáhla dostatečného výsledku (tj. zbytek je menší než oba moduly), nuluje příznak C2. Pokud je C2 jedničkové, vrací se pouze tzv. *částečný zbytek*. Exponent částečného zbytku je menší než exponent originálního dělence alespoň o 32. Programové vybavení by potom mělo instrukci opakovat (použitím částečného zbytku v ST jako dělence) do té doby, než bude C2 nulové.

Instrukce má význam na redukci argumentů periodických funkcí. Po dokončení redukce jsou k dispozici tři nejnižší bity podílu v příznacích C3,C1 a C0. Toto je důležité pro redukci tangenciální funkce (použitím modulu $\pi/4$), protože tím získáme původní úhel v jednom z osmi sektorů kruhu.

FPREM1 *Partial Remainder*

Instrukce vypočte zbytek po dělení ST/ST(1) a výsledek uloží do ST.

Instrukce	Popis
FPREM1	ST := zbytek po dělení ST/ST(1)

Operace

```
EXPDIV := exponent(ST) - exponent(ST(1));
IF EXPDIV < 64
THEN
  Q := celá část výsledku zaokrouhlením desetinné (ST/ST(1));
  ST := ST - (ST(1)*Q);
  C2 := 0;
  C0,C1,C3 := tři bity nejnižšího řádu z Q; (* Q2,Q1,Q0 *)
ELSE
  C2 := 1:
  N := hodnota mezi 32 a 63;
  QQ := celá část výsledku oříznutím desetinné ((ST/ST(1))/2^(EXPDIF-N));
  ST := ST - (ST(1)*QQ*2^(EXPDIF-N);
FI;
```

Modifikované příznaky FPU

C0, C1, C2, C3 podle popisu na str. 35.

Numerické výjimky

U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce pracuje podle normy IEEE Std 754.

Pokud instrukce dosáhla dostatečného výsledku (tj. zbytek je menší než oba moduly), nuluje příznak C2. Pokud je C2 jedničkové, vrací se pouze tzv. *částečný zbytek*. Exponent částečného zbytku je menší než exponent originálního dělence alespoň o 32. Programové vybavení by potom mělo instrukci opakovat (použitím částečného zbytku v ST jako dělence) do té doby, než bude C2 nulové.

Instrukce má význam na redukci argumentů periodických funkcí. Po dokončení redukce jsou k dispozici tři nejnižší bity podílu v příznacích C3,C1 a C0. Toto je důležité pro redukci tangenciální funkce (použitím modulu $\pi/4$), protože tím získáme původní úhel v jednom z osmi sektorů kruhu.

FPTAN *Partial Tangent*

Instrukce nahradí ST hodnotou *tan*(ST) a vloží do zásobníku FPU hodnotu 1,0. ST je v radiánech a leží v intervalu $|\mathrm{ST}| < 2^{63}$.

Instrukce	Popis
FPTAN	ST := *tan*(ST); push 1,0

Operace

```
IF operand je v rozmezí
THEN
  C2 := 0;
  ST := tan(ST);
  sniž ukazatel vrcholu zásobníku;
  ST := 1,0;
ELSE
  C2 := 1;
FI;
```

Modifikované příznaky FPU

C1, C2 podle popisu na str. 35. C0 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Pro kompatibilitu s 8087 a Intel287 se udržuje to, že po provedení instrukce se ukládá hodnota 1 do zásobníku a tím se zjednodušuje výpočet ostatních trigonometrických funkcí. Např. kotangens, který je reciproční hodnotou funkce tangens, se může spočítat posloupností instrukcí FPTAN a FDIVR. ST(7) musí být před provedením instrukce prázdný.

FRNDINT *Round to Integer*

Instrukce zaokrouhlí hodnotu ST na celé číslo podle předpisu v poli RC řídicího slova FPU.

Instrukce	Popis
FRNDINT	zaokrouhlení ST na celé číslo

Operace

```
ST := zaokrouhlené ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FRSTOR *Restore FPU State*

Instrukce obnoví stav FPU (zahrnuje prostředí a zásobník registrů) z paměti. Informace byly dříve do paměti zapsány instrukcemi FSAVE nebo FNSAVE.

Instrukce	Popis
FRSTOR *m94/108byte*	obnov stav FPU

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou naplněny.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Blok informací v paměti obsahuje údaje o prostředí (viz FLDENV) a za nimi jsou uloženy obsahy registrů ST až ST(7). Instrukce by měla být provedena ve stejném operačním režimu jako odpovídající FSAVE nebo FNSAVE. Velikost operandu je určena hodnotou atributu velikosti operandu pro daný instrukční segment.

FSAVE/FNSAVE *Store FPU State*

Instrukce uloží stav FPU (zahrnuje prostředí a zásobník registrů) do paměti a provede inicializaci FPU. Informace lze z paměti obnovit instrukcí FRSTOR.

Instrukce	Popis
FSAVE *m94/108byte*	ulož stav FPU po kontrole FP chyb
FNSAVE *m94/108byte*	ulož stav FPU bez kontroly

Operace

```
cíl := stav FPU;
inicializuj FPU; (* stejně jako v FNINIT *)
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou vynulovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Blok informací v paměti obsahuje údaje o prostředí (viz FLDENV) a za nimi jsou uloženy obsahy registrů ST až ST(7). Instrukce by měla být provedena ve stejném operačním režimu jako odpovídající FSAVE nebo FNSAVE. Velikost operandu je určena hodnotou atributu velikosti operandu pro daný instrukční segment.

Instrukce stav neuloží do té doby, než jsou ukončeny všechny aktivity FPU. Pokud za instrukcí ukládající stav FPU do paměti následuje instrukce čtoucí tento obsah, musí jí předcházet FWAIT.

Ukládání stavu FPU se provádí zpravidla v souvislosti s přepínání kontextu procesoru ve víceúlohovém operačním systému.

FSCALE *Scale*

Instrukce interpretuje hodnotu v ST(1) jako celé číslo a přičte tuto hodnotu k exponentu ST. Instrukce je vlastně rychlá násobička nebo dělička hodnotami mocnin čísla 2.

Instrukce	Popis
FSCALE	K exponentu ST přičti ST(1)

Operace

```
ST := ST * 2^(ST(1));
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukci lze použít jako inverzní k FXTRACT. FSCALE však ze zásobníku nevybírá exponentovou část, a proto musí být následována FSTP ST(1).

FSIN *Sine*

Instrukce nahradí ST hodnotou *sin*(ST). ST je v radiánech a leží v intervalu $|ST| < 2^{63}$.

Instrukce	Popis
FSIN	ST := *sin*(ST)

Operace

```
IF operand je v rozmezí
THEN
  C2 := 0;
  ST := sin(ST);
ELSE
  C2 := 1;
FI;
```

Modifikované příznaky FPU

C1, C2 podle popisu na str. 35. C0 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FSINCOS *Sine and Cosine*

Instrukce nahradí ST hodnotou *sin*(ST) a vloží do zásobníku hodnotu *cos*(původního ST). ST je v radiánech a leží v intervalu $|\mathrm{ST}| < 2^{63}$.

Instrukce	Popis
FSINCOS	ST := *sin*(ST); push *cos*(původního ST)

Operace

```
IF operand je v rozmezí
THEN
  C2 := 0;
  Pomocná := cos(ST);
  ST := sin(ST);
  sniž ukazatel vrcholu zásobníku FPU;
  ST := Pomocná;
ELSE
  C2 := 1;
FI;
```

Modifikované příznaky FPU

C1, C2 podle popisu na str. 35. C0 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Rychlejší než FSINCOS je provedení dvou instrukcí FSIN a FCOS.

FSQRT *Square Root*

Instrukce nahradí ST druhou odmocninou ST.

Instrukce	Popis
FSQRT	ST := $\sqrt{ST}$

Operace

```
ST := druhá odmocnina ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Druhá odmocnina -0 je -0.

FST/FSTP *Store Real*

Instrukce zkopíruje obsah ST do operandu, který může být jiný registr nebo paměťový operand. FSTP hodnotu zkopíruje a potom vybere ze zásobníku.

Instrukce	Popis
FST *m32real*	zkopíruj ST do *m32real*
FST *m64real*	zkopíruj ST *m64real*
FST ST(i)	zkopíruj ST do ST(i)
FSTP *m32real*	zkopíruj ST do *m32real* a pop ST
FSTP *m64real*	zkopíruj ST *m64real* a pop ST
FSTP *m80real*	zkopíruj ST *m80real* a pop ST
FSTP ST(i)	zkopíruj ST do ST(i) a pop ST

Operace

```
cíl := ST(0);
IF instrukce je FSTP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li operandem registr, je použit ten, na který operand ukazuje před provedením pop.

FSTCW/FNSTCW *Store Control Word*

Instrukce zapíše obsah řídicího slova FPU do paměťového operandu.

Instrukce	Popis
FSTCW *m2byte*	ulož řídicí slovo FPU po kontrole FP chyb
FNSTCW *m2byte*	ulož řídicí slovo FPU bez kontroly

Operace

```
cíl := řídicí slovo FPU;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

FSTENV/FNSTENV *Store FPU Environment*

Uloží prostředí (Environment) FPU do z bloku slabik v paměti, který lze později použít instrukcí FLDENV.

Instrukce	Popis
FSTENV *m14/28byte*	ulož prostředí FPU po kontrole FP chyb
FNSTENV *m14/28byte*	ulož prostředí FPU bez kontroly

Operace

```
cíl := prostředí FPU;
CW[0..5] := 111111b;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány;

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Blok prostředí FPU, kterým se FPU plní, obsahuje řídicí slovo, stavové slovo, doplňující slovo a datový a instrukční chybový ukazatel. Velikost operandu je určena hodnotou atributu velikosti operandu pro daný instrukční segment.

Instrukce prostředí neuloží do té doby, než jsou ukončeny všechny aktivity FPU. Pokud za instrukcí ukládající prostředí FPU do paměti následuje instrukce čtoucí tento obsah, musí jí předcházet FWAIT.

Instrukce po uložení prostředí nastavuje všechny bity masky výjimek. Ukládání prostředí FPU se provádí zpravidla v souvislosti s obsluhou výjimek, protože obslužné rutiny potřebují mít přístup k ukazatelům FP chyb.

FSTSW/FNSTSW *Store Status Word*

Instrukce zapisují současnou hodnotu stavového slova FPU do operandu, který smí být 2slabikový paměťový operand nebo registr AX.

Instrukce	Popis
FSTSW *m2byte*	ulož stav do paměti po kontrole FP chyb
FSTSW AX	ulož stav do AX po kontrole FP chyb
FNSTSW *m2byte*	ulož stav do paměti bez kontroly
FNSTSW AX	ulož stav do AX bez kontroly

Operace

```
cíl := stavové slovo;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

Pokud je cíl uvnitř nezapisovatelného segmentu, generuje se #GP(0). Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce se používají při podmíněných skocích (po srovnávání, po instrukcích FPREM, FPREM1 nebo FXAM). Po provedení instrukce FNSTSW AX je obsah AX ihned k dispozici. Stavové slovo se vztahuje k výsledku předchozí prováděné instrukce.

FSUB/FSUBP/FISUB *Subtract*

Instrukce odčítá od vrcholu zásobníku operand a rozdílem nahradí vrchol zásobníku.

Instrukce	Popis
FSUB *m32real*	ST:=ST–*m32real*
FSUB *m64real*	ST:=ST–*m64real*
FSUB ST,ST(i)	ST:=ST–ST(i)
FSUB ST(i),ST	ST(i):=ST(i)–ST
FSUBP ST(i),ST	ST(i):=ST(i)–ST a Pop ST
FSUBP	ST(1):=ST(1)–ST a Pop ST
FISUB *m32int*	ST:=ST–*m32int*
FISUB *m16int*	ST:=ST–*m16int*

Operace

```
FSUB cíl,zdroj
cíl := cíl - zdroj;
IF instrukce je FSUBP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, D, I, IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát.

FSUBR/FSUBRP/FISUBR *Reverse Subtract*

Instrukce odčítá od operandu obsah vrcholu zásobníku a rozdíl uloží do cílového operandu.

Instrukce	Popis
FSUBR *m32real*	ST:=*m32real*–ST
FSUBR *m64real*	ST:=*m64real*–ST
FSUBR ST,ST(i)	ST:=ST(i)–ST
FSUBR ST(i),ST	ST(i):=ST–ST(i)
FSUBRP ST(i),ST	ST(i):=ST–ST(i) a Pop ST
FSUBRP	ST(1):=ST–ST(1) a Pop ST
FISUBR *m32int*	ST:=*m32int*–ST
FISUBR *m16int*	ST:=*m16int*–ST

Operace

```
FSUBR cíl,zdroj
cíl := zdroj - cíl;
IF instrukce je FSUBRP THEN pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, D, I, IS.

Výjimky ve chráněném režimu

Chybná efektivní adresa operandu v paměti uložená v registrech CS, DS, ES, FS a GS vyvolá #GP(0), v registru SS vyvolá #SS(0). Při výpadku stránky je #PF(výpadek), při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Pokud libovolná část operandu leží vně intervalu efektivních adres 0 až 0FFFF, generuje se výjimka 13. Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

Stejná výjimka jako v reálném režimu. #PF(výpadek) se generuje při výpadku stránky a při nezarovnaném přístupu k paměťovému operandu procesu s CPL=3 se generuje #AC. #NM při nastaveném EM nebo TS v CR0.

Poznámka

Je-li zdrojový operand v paměti, automaticky se převádí na extended-real formát.

FTST *Test*

Instrukce srovnává číslo na vrcholu zásobníku s hodnotou 0,0.

Instrukce	Popis
FTST	Porovnej ST s 0,0

Operace

```
CASE vztah operandů OF
  neporovnatelné: C3,C2,C0 := 111;
  ST > 0: C3,C2,C0 := 000;
  ST < 0: C3,C2,C0 := 001;
  ST = 0: C3,C2,C0 := 100;
ESAC;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 viz výše.

Příznak FPU	**EFlags**
C0	CF
C1	žádný
C2	PF
C3	ZF

Numerické výjimky

D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Znaménko nuly se ignoruje ($-0,0 = +0,0$).

FUCOM/FUCOMP/FUCOMPP *Compare Real*

Instrukce porovnává dvě reálná čísla: číslo na vrcholu zásobníku a operand, kterým musí být registr. Pokud není uveden žádný operand, srovnává se ST s ST(1).

Instrukce	Popis
FUCOM ST(i)	Porovnej ST s ST(i)
FUCOM	Porovnej ST s ST(1)
FUCOMP ST(i)	Porovnej ST s ST(i) a pop ST
FUCOMP	Porovnej ST s ST(1) a pop ST
FUCOMPP	Porovnej ST s ST(1) a pop ST dvakrát

Operace

```
CASE vztah operandů OF
  neporovnatelné: C3,C2,C0 := 111;
  ST > operand: C3,C2,C0 := 000;
  ST < operand: C3,C2,C0 := 001;
  ST = operand: C3,C2,C0 := 100;
ESAC;
IF instrukce je FUCOMP THEN pop ST; FI;
IF instrukce je FUCOMPP THEN pop ST; pop ST; FI;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 viz výše.

Příznak FPU	**EFlags**
C0	CF
C1	žádný
C2	PF
C3	ZF

Numerické výjimky

D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Znaménko nuly se ignoruje ($-0,0 = +0,0$).

FWAIT *Wait*

Instrukce uvede procesor do stavu čekání na numerickou výjimku. Instrukce je synonymen WAIT – nejde o ESC.

Instrukce	Popis
FWAIT	synonym WAIT

Operace

```
Uveď procesor do stavu čekání na numerickou výjimku před předáním
  řízení další instrukci.
```

Modifikované příznaky FPU

C0, C1, C2 a C3 jsou nedefinovány.

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FXAM *Examine*

Instrukce vrací identifikaci typu objektu uloženého v ST.

Instrukce	Popis
FXAM	oznam typ objektu v ST

Operace

```
C1 := znaménkový bit ST; (* 0 pro kladnou hodnotu, 1 pro zápornou *)
CASE typ objektu v ST OF
  nepodporovaný: C3,C2,C0 := 000;
  NaN: C3,C2,C0 := 001;
  Normal: C3,C2,C0 := 010;
  Infinity: C3,C2,C0 := 011;   (* nekonečno *)
  Zero: C3,C2,C0 := 100;       (* nula *)
  Empty: C3,C2,C0 := 101;      (* prázdný *)
  Denormal: C3,C2,C0 := 110;
```

Modifikované příznaky FPU

C0, C1, C2 a C3 viz výše.

Příznak FPU	EFlags
C0	CF
C1	žádný
C2	PF
C3	ZF

Numerické výjimky

Žádné.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

C1 reprezentuje znaménko obsahu ST(0) bez ohledu na to, zdali je obsah prázdný, nebo ne.

FXCH *Exchange Register Contents*

Instrukce zamění obsah operandu a vrcholu zásobníku. Není-li operand zadán, pracuje se s ST(1).

Instrukce	Popis
FXCH ST(i)	zaměň obsah ST a ST(i)
FXCH	zaměň obsah ST a ST(1)

Operace

```
Pomocná := ST;
ST := operand;
operand := Pomocná;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Řada instrukcí pracuje pouze s vrcholem zásobníku. Potřebujeme-li zpracovat i obsah jiného registru (např. vypočítat odmocninu ST(3)), použijeme instrukci FXCH, např.:

```
FXCH ST(3)
FSQRT
FXCH ST(3)
```

FXTRACT *Extract Exponent and Significand*

Instrukce rozdělí hodnotu v ST na její exponent a mantisu. Exponentem je nahrazen původní operand a mantisa je vložena do zásobníku.

Instrukce	Popis
FXTRACT	

Operace

```
Pomocná := mantisa z ST;
ST := exponent z ST;
sniž ukazatel vrcholu zásobníku FPU;
ST := Pomocná;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

Z, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

Poznámka

Instrukce ponechá v ST mantisu s původním znaménkem jako reálné číslo s exponentem 0 a v ST(1) exponent původní hodnoty opět jako reálné číslo.

Instrukce FXTRACT provádí nadmnožinu operací normou IEEE doporučované funkce $\log_b x$.

Pokud je vstupní operand nulový, instrukce jako exponent vrátí hodnotu $-\infty$ v ST(1) a ST je přiřazena nula se znaménkem odpovídajícím původnímu operandu. Nastaví se příznak dělení nulou. ST(7) musí být před provedením instrukce prázdný.

FYL2X

Compute $y \times \log_2 x$

Instrukce vypočítá hodnotu ST(1):=ST(1) $\times \log_2$ ST a potom provede pop ST. Operand v ST nesmí být záporný nebo nulový.

Instrukce	Popis
FYL2X	ST(1):=ST(1) $\times \log_2$ ST; pop ST

Operace

```
ST(1) := ST(1) * log2(ST);
pop ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, O, Z, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

FYL2XP1

Compute $y \times \log_2(x+1)$

Instrukce vypočítá hodnotu ST(1):=ST(1) $\times \log_2(\mathrm{ST}+1)$ a potom provede pop ST. Operand v ST musí být v intervalu $-(1-(\sqrt{2}/2)) \leq \mathrm{ST} \leq \sqrt{2}-1$ být záporný nebo nulový.

Instrukce	Popis
FYL2XP1	ST(1):=ST(1) $\times \log_2(\mathrm{ST}+1)$; pop ST

Operace

```
ST(1) := ST(1) * log2(ST+1.0);
pop ST;
```

Modifikované příznaky FPU

C1 podle popisu na str. 35. C0, C2 a C3 jsou nedefinovány.

Numerické výjimky

P, U, D, I, IS.

Výjimky ve chráněném režimu

#NM při nastaveném EM nebo TS v CR0.

Výjimky v reálném režimu

Výjimka 7 při nastaveném EM nebo TS v CR0.

Výjimky v režimu V86

#NM při nastaveném EM nebo TS v CR0.

A. Operační znaky instrukcí

Operační znaky instrukcí jsou ve sloupci *Opcode* v šestnáctkovém tvaru. Jsou zapsány v tom pořadí, v jakém jsou zakódovány v instrukci. V témže sloupci jsou ještě tyto další definiční symboly:

- **/digit:** (číslo v intervalu 0 až 7) oznamuje, že ve slabice **ModR/M** je definován pouze operand typu *r/m* (registr nebo operand). Pole **reg** obsahuje číslo rozšiřující definici operačního znaku.
- **/r:** indikuje, že slabika **ModR/M** obsahuje registrový operand a *r/m* operand.
- **cb, cw, cd, cp:** je 1slabiková (cb), 2slabiková (cw), 4slabiková (cd) nebo 6slabiková (cp) hodnota uložená za operačním znakem, která je použita pro specifikaci instrukčního (Code) offsetu a případně nové hodnoty pro instrukční segment.
- **ib, iw, id, ip:** je 1slabikový (ib), 2slabikový (iw), 4slabikový (id) nebo 6slabikový (ip) přímý operand, který je uložen za operačním znakem, slabikou ModR/M nebo SIB. Operační znak určuje, je-li operand číslo se znaménkem. Slova a dvojslova mají slabiky uspořádány od nejnižší po nejvyšší.
- **+rb, +rw, +rd:** je číslo registru v intervalu 0 až 7 přičtené k šestnáctkově zapsané slabice nalevo od znaménka plus ve formě samostatného operačního znaku. Možná čísla jsou:

Číslo	**rb**	**rw**	**rd**
0	AL	AX	EAX
1	CL	CX	ECX
2	DL	DX	EDX
3	BL	BX	EBX
4	AH	SP	ESP
5	CH	BP	EBP
6	DH	SI	ESI
7	BH	DI	EDI

- **+i:** se používá v instrukcích FPU, je-li jeden z operandů registr ST(i) ze sady registrů FPU. Hodnota **i** (v intervalu 0 až 7) je připočtena k předchozí slabice ve formě samostatného operačního znaku.

Ve sloupci *Clocks* jsou údaje o trvání instrukce v počtech hodinových cyklů. Uváděná hodnota je vypočítána za dodržení těchto předpokladů:

- přístup k datům a instrukcím byl obsloužen vyrovnávací pamětí (Cache Hit),
- cíl skokové instrukce je rovněž ve vyrovnávací paměti,
- s instrukcí se neprovádějí zneplatňovací cykly,
- TLB obsahuje všechny adresy pro běh instrukce,
- výpočet efektivní adresy používá bázový registr, který nebyl cílovým registrem pro předchozí instrukci,
- během provádění instrukce nevznikají výjimky,
- nejsou zpoždění vyvolaná zápisovými operacemi.

Ve sloupci *Clocks* se uvádějí tyto symboly:

- **n** reprezentuje počet opakování,
- **m** určuje počet komponent v další prováděné instrukci, kde se jako jedna komponenta počítá celý přírůstek, celý přímý operand, každý prefix a každá další slabika instrukce,
- **pm=** je počet hodinových cyklů ve chráněném režimu (Protected Mode). Tento parametr se zadává tehdy, liší-li se počet hodinových cyklů v chráněném režimu od reálného.

Nastane-li během provádění instrukce výjimka a obslužná rutina se nachází v jiném procesu, prodlouží se doba provádění instrukce o počet hodinových cyklů nutných k přepnutí procesu. Tento parametr závisí na několika faktorech:

- na typu TSS reprezentujícího nový proces (32 nebo 16bitový TSS),
- zdali současný proces běží ve V86,
- zdali nový proces má běžet ve V86,
- zdali všechny požadované údaje jsou ve vyrovnávací paměti,
- zdali je použita brána pro přepnutí procesu nebo brána pro přerušení nebo trap.

Počty cyklů nutných pro přepnutí procesu jsou uvedeny v následující tabulce (v dalším textu označovány ts):

Starý proces	**Nový proces**		
	do 32bit. TSS	do 16bit. TSS	do VM86 TSS
VM86/32/16bit. TSS	85	87	71

r8(/r)			AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
/digit(Opcode)			0	1	2	3	4	5	6	7
REG=			000	001	010	011	100	101	110	111
Efektivní adresa	**Md**	**R/M**	**Hodnoty ModR/M šestnáctkově**							
[BX+SI]	00	000	00	08	10	18	20	28	30	38
[BX+DI]		001	01	09	11	19	21	29	31	39
[BP+SI]		010	02	0A	12	1A	22	2A	32	3A
[BP+DI]		011	03	0B	13	1B	23	2B	33	3B
[SI]		100	04	0C	14	1C	24	2C	34	3C
[DI]		101	05	0D	15	1D	25	2D	35	3D
disp16		110	06	0E	16	1E	26	2E	36	3E
[BX]		111	07	0F	17	1F	27	2F	37	3F
[BX+SI]+disp8	01	000	40	48	50	58	60	68	70	78
[BX+DI]+disp8		001	41	49	51	59	61	69	71	79
[BP+SI]+disp8		010	42	4A	52	5A	62	6A	72	7A
[BP+DI]+disp8		011	43	4B	53	5B	63	6B	73	7B
[SI]+disp8		100	44	4C	54	5C	64	6C	74	7C
[DI]+disp8		101	45	4D	55	5D	65	6D	75	7D
[BP]+disp8		110	46	4E	56	5E	66	6E	76	7E
[BX]+disp8		111	47	4F	57	5F	67	6F	77	7F
[BX+SI]+disp16	10	000	80	88	90	98	A0	A8	B0	B8
[BX+DI]+disp16		001	81	89	91	99	A1	A9	B1	B9
[BP+SI]+disp16		010	82	8A	92	9A	A2	AA	B2	BA
[BP+DI]+disp16		011	83	8B	93	9B	A3	AB	B3	BB
[SI]+disp16		100	84	8C	94	9C	A4	AC	B4	BC
[DI]+disp16		101	85	8D	95	9D	A5	AD	B5	BD
[BP]+disp16		110	86	8E	96	9E	A6	AE	B6	BE
[BX]+disp16		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/CH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

Poznámky:

disp8 znamená 8bitový přírůstek uložený za slabikou ModR/M, který se znaménkově rozšíří a přičte k indexu.

disp16 znamená 16bitový přírůstek uložený za slabikou ModR/M, který se přičte k indexu.

Implicitní segmentový registr je SS pro efektivní adresy obsahující BP a DS pro všechny ostatní.

Obr. A.1 16bitový adresový formát a slabika ModR/M

r8(/r)			AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
/digit(Opcode)			0	1	2	3	4	5	6	7
REG=			000	001	010	011	100	101	110	111
Efektivní adresa	**Mod**	**R/M**	**Hodnoty ModR/M šestnáctkově**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--]		100	04	0C	14	1C	24	2C	34	3C
disp32		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX]	01	000	40	48	50	58	60	68	70	78
disp8[ECX]		001	41	49	51	59	61	69	71	79
disp8[EDX]		010	42	4A	52	5A	62	6A	72	7A
disp8[EBX]		011	43	4B	53	5B	63	6B	73	7B
disp8[--][--]		100	44	4C	54	5C	64	6C	74	7C
disp8[EBP]		101	45	4D	55	5D	65	6D	75	7D
disp8[ESI]		110	46	4E	56	5E	66	6E	76	7E
disp8[EDI]		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX]	10	000	80	88	90	98	A0	A8	B0	B8
disp32[ECX]		001	81	89	91	99	A1	A9	B1	B9
disp32[EDX]		010	82	8A	92	9A	A2	AA	B2	BA
disp32[EBX]		011	83	8B	93	9B	A3	AB	B3	BB
disp32[--][--]		100	84	8C	94	9C	A4	AC	B4	BC
disp32[EBP]		101	85	8D	95	9D	A5	AD	B5	BD
disp32[ESI]		110	86	8E	96	9E	A6	AE	B6	BE
disp32[EDI]		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

Poznámky:

[--][--] znamená, že za slabikou ModR/M následuje SIB.

disp8 znamená 8bitový přírůstek uložený za slabikou SIB, který se znaménkově rozšíří a přičte k indexu.

disp32 znamená 32bitový přírůstek uložený za slabikou SIB, který se přičte k indexu.

Obr. A.2 32bitový adresový formát a slabika ModR/M

r32			EAX	ECX	EDX	EBX	ESP	[*]	ESI	EDI
Base =			0	1	2	3	4	5	6	7
Base =			000	001	010	011	100	101	110	111
Scaled index	**SS**	**Index**	**Hodnoty SIB šestnáctkově**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
nepoužito		100	04	0C	14	1C	24	2C	34	3C
[EBP]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX*2]	01	000	40	48	50	58	60	68	70	78
[ECX*2]		001	41	49	51	59	61	69	71	79
[EDX*2]		010	42	4A	52	5A	62	6A	72	7A
[EBX*2]		011	43	4B	53	5B	63	6B	73	7B
nepoužito		100	44	4C	54	5C	64	6C	74	7C
[EBP*2]		101	45	4D	55	5D	65	6D	75	7D
[ESI*2]		110	46	4E	56	5E	66	6E	76	7E
[EDI*2]		111	47	4F	57	5F	67	6F	77	7F
[EAX*4]	10	000	80	88	90	98	A0	A8	B0	B8
[ECX*4]		001	81	89	91	99	A1	A9	B1	B9
[EDX*4]		010	82	8A	92	9A	A2	AA	B2	BA
[EBX*4]		011	83	8B	93	9B	A3	AB	B3	BB
nepoužito		100	84	8C	94	9C	A4	AC	B4	BC
[EBP*4]		101	85	8D	95	9D	A5	AD	B5	BD
[ESI*4]		110	86	8E	96	9E	A6	AE	B6	BE
[EDI*4]		111	87	8F	97	9F	A7	AF	B7	BF
[EAX*8]	11	000	C0	C8	D0	D8	E0	E8	F0	F8
[ECX*8]		001	C1	C9	D1	D9	E1	E9	F1	F9
[EDX*8]		010	C2	CA	D2	DA	E2	EA	F2	FA
[EBX*8]		011	C3	CB	D3	DB	E3	EB	F3	FB
nepoužito		100	C4	CC	D4	DC	E4	EC	F4	FC
[EBP*8]		101	C5	CD	D5	DD	E5	ED	F5	FD
[ESI*8]		110	C6	CE	D6	DE	E6	EE	F6	FE
[EDI*8]		111	C7	CF	D7	DF	E7	EF	F7	FF

Poznámky:

[*] znamená disp32 bez báze při MOD=00, jinak [EBP]. Adresové módy jsou tyto:

disp32[index]	MOD=00
disp8[EBP][index]	MOD=01
disp32[EBP][index]	MOD=10

Obr. A.3 32bitový adresový formát a slabika SIB

Instrukce	*Opcode*	*Clocks*
AAA	37	3
AAD	D5 0A	10
AAM	D4 0A	18
AAS	3F	3
ADC AL,*imm8*	14 ib	1
ADC AX,*imm16*	15 iw	1
ADC EAX,*imm32*	15 id	1
ADC *r/m8,imm8*	80 /2 ib	1/3
ADC *r/m16,imm16*	81 /2 iw	1/3
ADC *r/m32,imm32*	81 /2 id	1/3
ADC *r/m16,imm8*	83 /2 ib	1/3
ADC *r/m32,imm8*	83 /2 ib	1/3
ADC *r/m8,r8*	10 /r	1/3
ADC *r/m16,r16*	11 /r	1/3
ADC *r/m32,r32*	11 /r	1/3
ADC *r8,r/m8*	12 /r	1/2
ADC *r16,r/m16*	13 /r	1/2
ADC *r32,r/m32*	13 /r	1/2
ADD AL,*imm8*	04 ib	1
ADD AX,*imm16*	05 iw	1
ADD EAX,*imm32*	05 id	1
ADD *r/m8,imm8*	80 /0 ib	1/3
ADD *r/m16,imm16*	81 /0 iw	1/3
ADD *r/m32,imm32*	81 /0 id	1/3
ADD *r/m16,imm8*	83 /0 ib	1/3
ADD *r/m32,imm8*	83 /0 ib	1/3
ADD *r/m8,r8*	00 /r	1/3
ADD *r/m16,r16*	01 /r	1/3
ADD *r/m32,r32*	01 /r	1/3
ADD *r8,r/m8*	02 /r	1/2
ADD *r16,r/m16*	03 /r	1/2
ADD *r32,r/m32*	03 /r	1/2
AND AL,*imm8*	24 ib	1
AND AX,*imm16*	25 iw	1
AND EAX,*imm32*	25 id	1
AND *r/m8,imm8*	80 /4 ib	1/3
AND *r/m16,imm16*	81 /4 iw	1/3
AND *r/m32,imm32*	81 /4 id	1/3
AND *r/m16,imm8*	83 /4 ib	1/3
AND *r/m32,imm8*	83 /4 ib	1/3
AND *r/m8,r8*	20 /r	1/3
AND *r/m16,r16*	21 /r	1/3

AND *r/m32,r32*	21 /r	1/3	
AND *r8,r/m8*	22 /r	1/2	
AND *r16,r/m16*	23 /r	1/2	
AND *r32,r/m32*	23 /r	1/2	
ARPL *r/m16,r16*	63 /r	pm=7	
BOUND *r16,m16&16*	62 /r	8 (v rozmezí), int+32 (mimo meze)	
BOUND *r32,m32&32*	62 /r	8 (v rozmezí), int+32 (mimo meze)	
BSF *r16,r/m16*	0F BC	6-34/6-35	
BSF *r32,r/m32*	0F BC	6-42/6-43	
BSR *r16,r/m16*	0F BD	7-39/6-40	
BSR *r32,r/m32*	0F BD	7-71/7-72	
BSWAP *r32*	0F C8+rd	1	
BT *r/m16,r16*	0F A3	4/9	
BT *r/m32,r32*	0F A3	4/9	
BT *r/m16,imm8*	0F BA /4 ib	4/9	
BT *r/m32,imm8*	0F BA /4 ib	4/9	
BTC *r/m16,r16*	0F BB	4/9	
BTC *r/m32,r32*	0F BB	4/9	
BTC *r/m16,imm8*	0F BA /7 ib	4/9	
BTC *r/m32,imm8*	0F BA /7 ib	4/9	
BTR *r/m16,r16*	0F B3	4/9	
BTR *r/m32,r32*	0F B3	4/9	
BTR *r/m16,imm8*	0F BA /6 ib	4/9	
BTR *r/m32,imm8*	0F BA /6 ib	4/9	
BTS *r/m16,r16*	0F AB	4/9	
BTS *r/m32,r32*	0F AB	4/9	
BTS *r/m16,imm8*	0F BA /5 ib	4/9	
BTS *r/m32,imm8*	0F BA /5 ib	4/9	
CALL *rel16*	E8 cw	1	
CALL *r/m16*	FF /2	2	
CALL *ptr16:16*	9A cd	4	
CALL *ptr16:16*	9A cd	pm=22	stejná úroveň
CALL *ptr16:16*	9A cd	pm=44	vyšší úroveň, bez par.
CALL *ptr16:16*	9A cd	pm=45+2x	vyšší úroveň, x par.
CALL *ptr16:16*	9A cd	pm=21+ts	přepnutí procesu
CALL *m16:16*	FF /3	5	
CALL *m16:16*	FF /3	pm=22	stejná úroveň
CALL *m16:16*	FF /3	pm=44	vyšší úroveň, bez par.
CALL *m16:16*	FF /3	pm=45+2x	vyšší úroveň, x par.
CALL *m16:16*	FF /3	pm=21+ts	přepnutí procesu
CALL *rel32*	E8 cd	1	
CALL *r/m32*	FF /2	2	
CALL *ptr16:32*	9A cp	4	

CALL *ptr16:32*	9A cp	pm=22	stejná úroveň
CALL *ptr16:32*	9A cp	pm=44	vyšší úroveň, bez par.
CALL *ptr16:32*	9A cp	pm=45+2x	vyšší úroveň, x par.
CALL *ptr16:32*	9A cp	pm=21+ts	přepnutí procesu
CALL *m16:32*	FF /3	5	
CALL *m16:32*	FF /3	pm=22	stejná úroveň
CALL *m16:32*	FF /3	pm=44	vyšší úroveň, bez par.
CALL *m16:32*	FF /3	pm=45+2x	vyšší úroveň, x par.
CALL *m16:32*	FF /3	pm=21+ts	přepnutí procesu
CBW	98	3	
CDQ	99	2	
CLC	F8	2	
CLD	FC	2	
CLI	FA	7	
CLTS	0F 06	10	
CMC	F5	2	
CMP AL,*imm8*	3C ib	1	
CMP AX,*imm16*	3D iw	1	
CMP EAX,*imm32*	3D id	1	
CMP *r/m8,imm8*	80 /7 ib	1/2	
CMP *r/m16,imm16*	81 /7 iw	1/2	
CMP *r/m32,imm32*	81 /7 id	1/2	
CMP *r/m16,imm8*	83 /7 ib	1/2	
CMP *r/m32,imm8*	83 /7 ib	1/2	
CMP *r/m8,r8*	38 /r	1/2	
CMP *r/m16,r16*	39 /r	1/2	
CMP *r/m32,r32*	39 /r	1/2	
CMP *r8,r/m8*	3A /r	1/2	
CMP *r16,r/m16*	3B /r	1/2	
CMP *r32,r/m32*	3B /r	1/2	
CMPS *m8,m8*	A6	5	
CMPS *m16,m16*	A7	5	
CMPS *m32,m32*	A7	5	
CMPSB	A6	5	
CMPSW	A7	5	
CMPSD	A7	5	
CMPXCHG *r/m8,r8*	0F B0 /r	6	
CMPXCHG *r/m16,r16*	0F B1 /r	6	
CMPXCHG *r/m32,r32*	0F B1 /r	6	
CMPXCHG8 *r/m64*	0F C7 m64	10	
CPUID	0F A2	14	
CWD	99	2	
CWDE	98	3	

DAA	27	3
DAS	2F	3
DEC *r/m8*	FE /1	1/3
DEC *r/m16*	FF /1	1/3
DEC *r/m32*	FF /1	1/3
DEC *r16*	48+rw	1
DEC *r32*	48+rw	1
DIV AL,*r/m8*	F6 /6	17
DIV AX,*r/m16*	F7 /6	25
DIV EAX,*r/m32*	F7 /6	41
ENTER *imm16*,0	C8 iw 00	11
ENTER *imm16*,1	C8 iw 01	15
ENTER *imm16,imm8*	C8 iw ib	15+2imm8
F2XM1	D9 F0	13-57
FABS	D9 E1	1
FADD *m32real*	D8 /0	3/1
FADD *m64real*	DC /0	3/1
FADD ST,ST(i)	D8 C0+i	3/1
FADD ST(i),ST	DC C0+i	3/1
FADDP ST(i),ST	DE C0+i	3/1
FADDP	DE C1	3/1
FIADD *m32int*	DA /0	7/4
FIADD *m16int*	DE /0	7/4
FBLD *m80dec*	DF /4	48-58
FBSTP *m80dec*	DF /6	148-154
FCHS	D9 E0	1
FCLEX	9B DB E2	9+min. 1 pro FWAIT
FNCLEX	DB E2	9
FCOM *m32real*	D8 /2	4/1
FCOM *m64real*	DC /2	4/1
FCOM ST(i)	D8 D0+i	4/1
FCOM	D8 D1	4/1
FCOMP *m32real*	D8 /3	4/1
FCOMP *m64real*	DC /3	4/1
FCOMP ST(i)	D8 D8+i	4/1
FCOMP	D8 D9	4/1
FCOMPP	DE D9	4/1
FCOS	D9 FF	18-124
FDECSTP	D9 F6	1
FDIV *m32real*	D8 /6	39
FDIV *m64real*	DC /6	39
FDIV ST,ST(i)	D8 F0+i	39
FDIV ST(i),ST	DC F8+i	39

FDIVP ST(i),ST	DE F8+i	39
FDIVP	DE F9	39
FIDIV *m32int*	DA /6	42
FIDIV *m16int*	DE /6	42
FDIVR *m32real*	D8 /7	39
FDIVR *m64real*	DC /7	39
FDIVR ST,ST(i)	D8 F8+i	39
FDIVR ST(i),ST	DC F0+i	39
FDIVRP ST(i),ST	DE F0+i	39
FDIVRP	DE F1	39
FIDIVR *m32int*	DA /7	42
FIDIVR *m16int*	DE /7	42
FFREE ST(i)	DD C0+i	1
FICOM *m16int*	DE /2	8/4
FICOM *m32int*	DA /2	8/4
FICOMP *m16int*	DE /3	8/4
FICOMP *m32int*	DA /3	8/4
FILD *m16int*	DF /0	3/1
FILD *m32int*	DB /0	3/1
FILD *m64int*	DF /5	3/1
FINCSTP	D9 F7	1
FINIT	DB E3	16
FNINIT	DB E3	12
FIST *m16int*	DF /2	6
FIST *m32int*	DB /2	6
FISTP *m16int*	DF /3	6
FISTP *m32int*	DB /3	6
FISTP *m64int*	DF /7	6
FLD *m32real*	D9 /0	1
FLD *m64real*	DD /0	1
FLD *m80real*	DB /5	3
FLD ST(i)	D9 C0+i	1
FLD1	D9 E8	2/2
FLDL2T	D9 E9	5/3
FLDL2E	D9 EA	5/3
FLDPI	D9 EB	5/3
FLDLG2	D9 EC	5/3
FLDLN2	D9 E9	5/3
FLDZ	D9 EE	2/2
FLDCW *m2byte*	D9 /5	7
FLDENV *m14/28byte*	D9 /4	32-37
FMUL *m32real*	D8 /1	3/1
FMUL *m64real*	DC /1	3/1

FMUL ST,ST(i)	D8 C8+i	3/1
FMUL ST(i),ST	DC C8+i	3/1
FMULP ST(i),ST	DE C8+i	3/1
FMULP	DE C9	3/1
FIMUL *m32int*	DA /1	7/4
FIMUL *m16int*	DE /1	7/4
FNOP	D9 D0	1
FPATAN	D9 F3	17-173
FPREM	D9 F8	16-64
FPREM1	D9 F5	20-70
FPTAN	D9 F2	17-173
FRNDINT	D9 FC	9-20
FRSTOR *m94/108byte*	DD /4	70-95
FSAVE *m94/108byte*	9B DD /6	124-151 +min. 3 pro FWAIT
FNSAVE *m94/108byte*	DD /6	124-151
FSCALE	F9 FD	20-31
FSIN	D9 FE	16-126
FSINCOS	D9 FB	17-137
FSQRT	D9-FA	70
FST *m32real*	D9 /2	2
FST *m64real*	DD /2	2
FST ST(i)	DD D0+i	1
FSTP *m32real*	D9 /3	2
FSTP *m64real*	DD /3	2
FSTP *m80real*	DB /7	3
FSTP ST(i)	DD D8+i	1
FSTCW *m2byte*	9B D9 /7	2 +min. 1 pro FWAIT
FNSTCW *m2byte*	D9 /7	2
FSTENV *m14/28byte*	9B D9 /6	48-50 +min. 3 pro FWAIT
FNSTENV *m14/28byte*	D9 /6	48-50
FSTSW *m2byte*	9B DD /7	2 +min. 3 pro FWAIT
FSTSW AX	9B DF E0	2 +min. 3 pro FWAIT
FNSTSW *m2byte*	DD /7	2
FNSTSW AX	DF E0	2
FSUB *m32real*	D8 /4	3/1
FSUB *m64real*	DC /4	3/1
FSUB ST,ST(i)	D8 E0+i	3/1
FSUB ST(i),ST	DC E8+i	3/1
FSUBP ST(i),ST	DE E8+i	3/1
FSUBP	DE E9	3/1
FISUB *m32int*	DA /4	7/4
FISUB *m16int*	DE /4	7/4
FSUBR *m32real*	D8 /5	3/1

FSUBR *m64real*	DC /5	3/1
FSUBR ST,ST(i)	D8 E8+i	3/1
FSUBR ST(i),ST	DC E0+i	3/1
FSUBRP ST(i),ST	DE E0+i	3/1
FSUBRP	DE E1	3/1
FISUBR *m32int*	DA /5	7/4
FISUBR *m16int*	DE /5	7/4
FTST	D9 E4	4/1
FUCOM ST(i)	DD E0+i	4/1
FUCOM	DD E1	4/1
FUCOMP ST(i)	DD E8+i	4/1
FUCOMP	DD E9	4/1
FUCOMPP	DA E9	4/1
FWAIT	9B	(1-3)
FXAM	D9 E5	21
FXCH ST(i)	D9 C8+i	1
FXCH	D9 C9	1
FXTRACT	D9 F4	13
FYL2X	D9 F1	22-111
FYL2XP1	D9 F9	22-103
HLT	F4	inf
IDIV *r/m8*	F6 /7	22
IDIV AX,*r/m16*	F7 /7	30
IDIV EAX,*r/m32*	F7 /7	46
IMUL *r/m8*	F6 /5	11
IMUL *r/m16*	F7 /5	11
IMUL *r/m32*	F7 /5	10
IMUL *r8,r/m8*	0F AF /r	10
IMUL *r16,r/m16*	0F AF /r	10
IMUL *r32,r/m32*	0F AF /r	10
IMUL *r16,r/m16,imm8*	6B /r ib	10
IMUL *r32,r/m32,imm8*	6B /r ib	10
IMUL *r16,imm8*	6B /r ib	10
IMUL *r32,imm8*	6B /r ib	10
IMUL *r16,r/m16,imm16*	69 /r iw	10
IMUL *r32,r/m32,imm32*	69 /r id	10
IMUL *r16,imm16*	69 /r iw	10
IMUL *r32,imm32*	69 /r id	10
IN AL,*imm8*	E4 ib	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19
IN AX,*imm8*	E5 ib	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19
IN EAX,*imm8*	E5 ib	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19
IN AL,DX	EC	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19
IN AX,DX	ED	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19

IN EAX,DX	ED	7,pm=4(CPL≤IOPL),21(CPL≥IOPL),vm=19	
INC *r/m8*	FE /0	1/3	
INC *r/m16*	FF /0	1/3	
INC *r/m32*	FF /0	1/3	
INC *r16*	40+rw	1	
INC *r32*	40+rw	1	
INS *m8*,DX	6C	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INS *m16*,DX	6D	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INS *m32*,DX	6D	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INSB	6C	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INSW	6D	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INSD	6D	9,pm=6(CPL≤IOPL),24(CPL≥IOPL),vm=22	
INT 3	CC	13	
INT 3	CC	27	chr. režim, stejná úroveň
INT 3	CC	44	chr. režim, vyšší úroveň
INT 3	CC	56	z V86 na úroveň 0
INT 3	CC	19+ts	s přep. procesu
INT *imm8*	CD ib	16	
INT *imm8*	CD ib	31	chr. režim, stejná úroveň
INT *imm8*	CD ib	48	chr. režim, vyšší úroveň
INT *imm8*	CD ib	82	z V86 na úroveň 0
INT *imm8*	CD ib	23+ts	s přep. procesu
INTO	CE	13	
INTO	CE	27	chr. režim, stejná úroveň
INTO	CE	44	chr. režim, vyšší úroveň
INTO	CE	56	z V86 na úroveň 0
INTO	CE	19+ts	s přep. procesu
INVD	0F 08	15	
INVLPG *m*	0F 01 /7	25	
IRET	CF	8	z reál. nebo V86
IRET	CF	10	vzdál. návrat
IRET	CF	27	na nižší úroveň
IRET	CF	ts+10	s přepn. procesu (NT=1)
IRETD	CF	10	vzdál. návrat
IRETD	CF	27	na nižší úroveň
IRETD	CF	ts+10	s přepn. procesu (NT=1)
JA *rel8*	77 cb	1	
JAE *rel8*	73 cb	1	
JB *rel8*	72 cb	1	
JBE *rel8*	76 cb	1	
JC *rel8*	72 cb	1	
JCXZ *rel8*	E3 cb	6,5	
JECXZ *rel8*	E3 cb	6,5	

JE *rel8*	74 cb	1
JG *rel8*	7F cb	1
JGE *rel8*	7D cb	1
JL *rel8*	7C cb	1
JLE *rel8*	7E cb	1
JNA *rel8*	76 cb	1
JNAE *rel8*	72 cb	1
JNB *rel8*	73 cb	1
JNBE *rel8*	77 cb	1
JNC *rel8*	73 cb	1
JNE *rel8*	75 cb	1
JNG *rel8*	7E cb	1
JNGE *rel8*	7C cb	1
JNL *rel8*	7D cb	1
JNLE *rel8*	7F cb	1
JNO *rel8*	71 cb	1
JNP *rel8*	7B cb	1
JNS *rel8*	79 cb	1
JNZ *rel8*	75 cb	1
JO *rel8*	70 cb	1
JP *rel8*	7A cb	1
JPE *rel8*	7A cb	1
JPO *rel8*	7B cb	1
JS *rel8*	78 cb	1
JZ *rel8*	74 cb	1
JA *rel16/32*	0F 87 cw/cd	1
JAE *rel16/32*	0F 83 cw/cd	1
JB *rel16/32*	0F 82 cw/cd	1
JBE *rel16/32*	0F 86 cw/cd	1
JC *rel16/32*	0F 82 cw/cd	1
JE *rel16/32*	0F 84 cw/cd	1
JG *rel16/32*	0F 8F cw/cd	1
JGE *rel16/32*	0F 8D cw/cd	1
JL *rel16/32*	0F 8C cw/cd	1
JLE *rel16/32*	0F 8E cw/cd	1
JNA *rel16/32*	0F 86 cw/cd	1
JNAE *rel16/32*	0F 82 cw/cd	1
JNB *rel16/32*	0F 83 cw/cd	1
JNBE *rel16/32*	0F 87 cw/cd	1
JNC *rel16/32*	0F 83 cw/cd	1
JNE *rel16/32*	0F 85 cw/cd	1
JNG *rel16/32*	0F 8E cw/cd	1
JNGE *rel16/32*	0F 8C cw/cd	1

JNL *rel16/32*	0F 8D cw/cd	1	
JNLE *rel16/32*	0F 8F cw/cd	1	
JNO *rel16/32*	0F 81 cw/cd	1	
JNP *rel16/32*	0F 8B cw/cd	1	
JNS *rel16/32*	0F 89 cw/cd	1	
JNZ *rel16/32*	0F 85 cw/cd	1	
JO *rel16/32*	0F 80 cw/cd	1	
JP *rel16/32*	0F 8A cw/cd	1	
JPE *rel16/32*	0F 8A cw/cd	1	
JPO *rel16/32*	0F 8B cw/cd	1	
JS *rel16/32*	0F 88 cw/cd	1	
JZ *rel16/32*	0F 84 cw/cd	1	
JMP *rel8*	EB cb	1	
JMP *rel16*	E9 cw	1	
JMP *r/m16*	FF /4	2	
JMP *ptr16:16*	EA cd	3	
JMP *ptr16:16*	EA cd	pm=18	stejná úroveň
JMP *ptr16:16*	EA cd	pm=19+ts	TSS
JMP *ptr16:16*	EA cd	pm=20+ts	brána TSS
JMP *m16:16*	FF /5	4	
JMP *m16:16*	FF /5	pm=18	stejná úroveň
JMP *m16:16*	FF /5	pm=19+ts	TSS
JMP *m16:16*	FF /5	pm=20+ts	brána TSS
JMP *rel32*	E9 cd	1	
JMP *r/m32*	FF /4	2	
JMP *ptr16:32*	EA cp	3	
JMP *ptr16:32*	EA cp	pm=18	stejná úroveň
JMP *ptr16:32*	EA cp	pm=19+ts	TSS
JMP *ptr16:32*	EA cp	pm=20+ts	brána TSS
JMP *m16:32*	FF /5	5	
JMP *m16:32*	FF /5	pm=18	stejná úroveň
JMP *m16:32*	FF /5	pm=19+ts	TSS
JMP *m16:32*	FF /5	pm=20+ts	brána TSS
LAHF	9F	2	
LAR *r16,r/m16*	0F 02 /r	8	
LAR *r32,r/m32*	0F 02 /r	8	
LDS *r16,m16:16*	C5 /r	4	
LDS *r32,m16:32*	C5 /r	4	
LSS *r16,m16:16*	0F B2 /r	4/pm=8	
LSS *r32,m16:32*	0F B2 /r	4/pm=8	
LES *r16,m16:16*	C4 /r	4	
LES *r32,m16:32*	C4 /r	4	
LFS *r16,m16:16*	0F B4 /r	4	

LFS *r32,m16:32*	0F B4 /r	4
LGS *r16,m16:16*	0F B5 /r	4
LGS *r32,m16:32*	0F B5 /r	4
LEA *r16,m*	8D /r	1
LEA *r32,m*	8D /r	1
LEAVE	C9	3
LGDT *m16&32*	0F 01 /2	6
LIDT *m16&32*	0F 01 /3	6
LLDT *r/m16*	0F 00 /2	9
LMSW *r/m16*	0F 01 /6	8
LOCK	F0	1
LODS *m8*	AC	2
LODS *m16*	AD	2
LODS *m32*	AD	2
LODSB	AC	2
LODSW	AD	2
LODSD	AD	2
LOOP *rel8*	E2 cb	5/6
LOOPE *rel8*	E1 cb	7/8
LOOPZ *rel8*	E1 cb	7/8
LOOPNE *rel8*	E0 cb	7/8
LOOPNZ *rel8*	E0 cb	7/8
LSL *r16,r/m16*	0F 03 /r	8
LSL *r32,r/m32*	0F 03 /r	8
LTR *r/m16*	0F 00 /3	10
MOV *r/m8,r8*	88 /r	1
MOV *r/m16,r16*	89 /r	1
MOV *r/m32,r32*	89 /r	1
MOV *r8,r/m8*	8A /r	1
MOV *r16,r/m16*	8B /r	1
MOV *r32,r/m32*	8B /r	1
MOV *r/m16,Sreg*	8C /r	1
MOV *Sreg,r/m16*	8E /r	2/3
MOV AL,*moffs8*	A0	1
MOV AX,*moffs16*	A1	1
MOV EAX,*moffs32*	A1	1
MOV *moffs8*,AL	A2	1
MOV *moffs16*,AX	A3	1
MOV *moffs32*,EAX	A3	1
MOV *reg8,imm8*	B0+rb	1
MOV *reg16,imm16*	B8+rw	1
MOV *reg32,imm32*	B8+rd	1
MOV *r/m8,imm8*	C6 /0	1

MOV *r/m16,imm16*	C7 /0	1
MOV *r/m32,imm32*	C7 /0	1
MOV CR0,*r32*	0F 22 /r	22
MOV CR2,*r32*	0F 22 /r	12
MOV CR3,*r32*	0F 22 /r	21/46
MOV CR4,*r32*	0F 22 /r	14
MOV *r32*,CR0-4	0F 20 /r	4
MOV *r32*,DR0-3	0F 21 /r	11
MOV *r32*,DR4-5	0F 21 /r	12
MOV *r32*,DR6-7	0F 21 /r	11
MOV DR0-3,*r32*	0F 23 /r	11
MOV DR4-5,*r32*	0F 23 /r	12
MOV DR6-7,*r32*	0F 23 /r	11
MOVS *m8,m8*	A4	4
MOVS *m16,m16*	A5	4
MOVS *m32,m32*	A5	4
MOVSB	A4	4
MOVSW	A5	4
MOVSD	A5	4
MOVSX *r16,r/m8*	0F BE /r	3
MOVSX *r32,r/m8*	0F BE /r	3
MOVSX *r32,r/m16*	0F BF /r	3
MOVZX *r16,r/m8*	0F B6 /r	3
MOVZX *r32,r/m8*	0F B6 /r	3
MOVZX *r32,r/m16*	0F B7 /r	3
MUL AL,*r/m8*	F6 /4	11
MUL AX,*r/m16*	F7 /4	11
MUL EAX,*r/m32*	F7 /4	10
NEG *r/m8*	F6 /3	1/3
NEG *r/m16*	F7 /3	1/3
NEG *r/m32*	F7 /3	1/3
NOP	90	1
NOT *r/m8*	F6 /2	1/3
NOT *r/m16*	F7 /2	1/3
NOT *r/m32*	F7 /2	1/3
OR AL,*imm8*	0C ib	1
OR AX,*imm16*	0D iw	1
OR EAX,*imm32*	0D id	1
OR *r/m8,imm8*	80 /1 ib	1/3
OR *r/m16,imm16*	81 /1 iw	1/3
OR *r/m32,imm32*	81 /1 id	1/3
OR *r/m16,imm8*	83 /1 ib	1/3
OR *r/m32,imm8*	83 /1 ib	1/3

OR *r/m8,r8*	08 /r	1/3
OR *r/m16,r16*	09 /r	1/3
OR *r/m32,r32*	09 /r	1/3
OR *r8,r/m8*	0A /r	1/2
OR *r16,r/m16*	0B /r	1/2
OR *r32,r/m32*	0B /r	1/2
OUT *imm8,*AL	E6 ib	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUT *imm8,*AX	E7 ib	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUT *imm8,*EAX	E7 ib	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUT DX,AL	EE	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUT DX,AX	EF	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUT DX,EAX	EF	12,pm=9(CPL≤IOPL),26(CPL≥IOPL),vm=24
OUTS DX,*r/m8*	6E	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
OUTS DX,*r/m16*	6F	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
OUTS DX,*r/m32*	6F	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
OUTSB	6E	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
OUTSW	6F	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
OUTSD	6F	13,pm=10(CPL≤IOPL),27(CPL≥IOPL),vm=25
POP *m16*	8F /0	3
POP *m32*	8F /0	3
POP *r16*	58+rw	1
POP *r32*	58+rd	1
POP DS	1F	3
POP ES	07	3
POP SS	17	3
POP FS	0F A1	3
POP GS	0F A9	3
POPA	61	5
POPAD	61	5
POPF	9D	pm=4, reál/V86=6
POPFD	9D	pm=4, reál/V86=6
PUSH *r/m16*	FF /6	1/2
PUSH *r/m32*	FF /6	1/2
PUSH *r16*	50+rw	1
PUSH *r32*	50+rd	1
PUSH *imm8*	6A	1
PUSH *imm16*	68	1
PUSH *imm32*	68	1
PUSH CS	0E	3
PUSH DS	1E	3
PUSH ES	06	3
PUSH SS	16	3
PUSH FS	0F A0	3

PUSH GS	0F A8	3
PUSHA	60	5
PUSHAD	60	5
PUSHF	9C	pm=3, reál/V86=4
PUSHFD	9C	pm=3, reál/V86=4
RCL *r/m8*,1	D0 /2	1/3
RCL *r/m8*,CL	D2 /2	7-24/9-26
RCL *r/m8,imm8*	C0 /2 ib	8-25/10-27
RCL *r/m16*,1	D1 /2	1/3
RCL *r/m16*,CL	D3 /2	7-24/9-26
RCL *r/m16,imm8*	C1 /2 ib	8-25/10-27
RCL *r/m32*,1	D1 /2	1/3
RCL *r/m32*,CL	D3 /2	7-24/9-26
RCL *r/m32,imm8*	C1 /2 ib	8-25/10-27
RCR *r/m8*,1	D0 /3	1/3
RCR *r/m8*,CL	D2 /3	7-24/9-26
RCR *r/m8,imm8*	C0 /3 ib	8-25/10-27
RCR *r/m16*,1	D1 /3	1/3
RCR *r/m16*,CL	D3 /3	7-24/9-26
RCR *r/m16,imm8*	C1 /3 ib	8-25/10-27
RCR *r/m32*,1	D1 /3	1/3
RCR *r/m32*,CL	D3 /3	7-24/9-26
RCR *r/m32,imm8*	C1 /3 ib	8-25/10-27
ROL *r/m8*,1	D0 /0	1/3
ROL *r/m8*,CL	D2 /0	4
ROL *r/m8,imm8*	C0 /0 ib	1/3
ROL *r/m16*,1	D1 /0	4
ROL *r/m16*,CL	D3 /0	1/3
ROL *r/m16,imm8*	C1 /0 ib	4
ROL *r/m32*,1	D1 /0	1/3
ROL *r/m32*,CL	D3 /0	4
ROL *r/m32,imm8*	C1 /0 ib	1/3
ROR *r/m8*,1	D0 /1	1/3
ROR *r/m8*,CL	D2 /1	4
ROR *r/m8,imm8*	C0 /1 ib	1/3
ROR *r/m16*,1	D1 /1	4
ROR *r/m16*,CL	D3 /1	1/3
ROR *r/m16,imm8*	C1 /1 ib	4
ROR *r/m32*,1	D1 /1	1/3
ROR *r/m32*,CL	D3 /1	4
ROR *r/m32,imm8*	C1 /1 ib	1/3
RDMSR	0F 32	20-24
REP INS *m8*,DX	F3 6C	

REP INS *m16*,DX	F3 6D		
REP INS *m32*,DX	F3 6D		
REP MOVS *m8,m8*	F3 A4		
REP MOVS *m16,m16*	F3 A5		
REP MOVS *m32,m32*	F3 A5		
REP OUTS DX,*r/m8*	F3 6E		
REP OUTS DX,*r/m16*	F3 6F		
REP OUTS DX,*r/m32*	F3 6F		
REP LODS *m8*	F3 AC		
REP LODS *m16*	F3 AD		
REP LODS *m32*	F3 AD		
REP STOS *m8*	F3 AA		
REP STOS *m16*	F3 AB		
REP STOS *m32*	F3 AB		
REPE CMPS *m8,m8*	F3 A6		
REPE CMPS *m16,m16*	F3 A7		
REPE CMPS *m32,m32*	F3 A7		
REPE SCAS *m8*	F3 AE		
REPE SCAS *m16*	F3 AF		
REPE SCAS *m32*	F3 AF		
REPNE CMPS *m8,m8*	F2 A6		
REPNE CMPS *m16,m16*	F2 A7		
REPNE CMPS *m32,m32*	F2 A7		
REPNE SCAS *m8*	F2 AE		
REPNE SCAS *m16*	F2 AF		
REPNE SCAS *m32*	F2 AF		
RET	C3	2	blízký
RET	CB	4	vzdálený na stejnou úroveň
RET	CB	23	vzdálený na nižší úroveň
RET *imm16*	C2 iw	3	blízký
RET *imm16*	CA iw	4	vzdálený na stejnou úroveň
RET *imm16*	CA iw	23	vzdálený na nižší úroveň
RSM	0F AA	83	
SAHF	9E	2	
SAL *r/m8*,1	D0 /4	1/3	
SAL *r/m8*,CL	D2 /4	4	
SAL *r/m8,imm8*	C0 /4 ib	1/3	
SAL *r/m16*,1	D1 /4	4	
SAL *r/m16*,CL	D3 /4	1/3	
SAL *r/m16,imm8*	C1 /4 ib	4	
SAL *r/m32*,1	D1 /4	1/3	
SAL *r/m32*,CL	D3 /4	4	
SAL *r/m32,imm8*	C1 /4 ib	1/3	

SAR *r/m8*,1	D0 /7	1/3
SAR *r/m8*,CL	D2 /7	4
SAR *r/m8,imm8*	C0 /7 ib	1/3
SAR *r/m16*,1	D1 /7	4
SAR *r/m16*,CL	D3 /7	1/3
SAR *r/m16,imm8*	C1 /7 ib	4
SAR *r/m32*,1	D1 /7	1/3
SAR *r/m32*,CL	D3 /7	4
SAR *r/m32,imm8*	C1 /7 ib	1/3
SHL *r/m8*,1	D0 /4	1/3
SHL *r/m8*,CL	D2 /4	4
SHL *r/m8,imm8*	C0 /4 ib	1/3
SHL *r/m16*,1	D1 /4	4
SHL *r/m16*,CL	D3 /4	1/3
SHL *r/m16,imm8*	C1 /4 ib	4
SHL *r/m32*,1	D1 /4	1/3
SHL *r/m32*,CL	D3 /4	4
SHL *r/m32,imm8*	C1 /4 ib	1/3
SHR *r/m8*,1	D0 /5	1/3
SHR *r/m8*,CL	D2 /5	4
SHR *r/m8,imm8*	C0 /5 ib	1/3
SHR *r/m16*,1	D1 /5	4
SHR *r/m16*,CL	D3 /5	1/3
SHR *r/m16,imm8*	C1 /5 ib	4
SHR *r/m32*,1	D1 /5	1/3
SHR *r/m32*,CL	D3 /5	4
SHR *r/m32,imm8*	C1 /5 ib	1/3
SBB AL,*imm8*	1C ib	1
SBB AX,*imm16*	1D iw	1
SBB EAX,*imm32*	1D id	1
SBB *r/m8,imm8*	80 /3 ib	1/3
SBB *r/m16,imm16*	81 /3 iw	1/3
SBB *r/m32,imm32*	81 /3 id	1/3
SBB *r/m16,imm8*	83 /3 ib	1/3
SBB *r/m32,imm8*	83 /3 ib	1/3
SBB *r/m8,r8*	18 /r	1/3
SBB *r/m16,r16*	19 /r	1/3
SBB *r/m32,r32*	19 /r	1/3
SBB *r8,r/m8*	1A /r	1/2
SBB *r16,r/m16*	1B /r	1/2
SBB *r32,r/m32*	1B /r	1/2
SCAS *m8*	AE	4
SCAS *m16*	AF	4

SCAS *m32*	AF	4
SCASB	AE	4
SCASW	AF	4
SCASD	AF	4
SETA *r/m8*	0F 97	1/2
SETAE *r/m8*	0F 93	1/2
SETB *r/m8*	0F 92	1/2
SETBE *r/m8*	0F 96	1/2
SETC *r/m8*	0F 92	1/2
SETE *r/m8*	0F 94	1/2
SETG *r/m8*	0F 9F	1/2
SETGE *r/m8*	0F 9D	1/2
SETL *r/m8*	0F 9C	1/2
SETLE *r/m8*	0F 9E	1/2
SETNA *r/m8*	0F 96	1/2
SETNAE *r/m8*	0F 92	1/2
SETNB *r/m8*	0F 93	1/2
SETNBE *r/m8*	0F 97	1/2
SETNC *r/m8*	0F 93	1/2
SETNE *r/m8*	0F 95	1/2
SETNG *r/m8*	0F 9E	1/2
SETNGE *r/m8*	0F 9C	1/2
SETNL *r/m8*	0F 9D	1/2
SETNLE *r/m8*	0F 9F	1/2
SETNO *r/m8*	0F 91	1/2
SETNP *r/m8*	0F 9B	1/2
SETNS *r/m8*	0F 99	1/2
SETNZ *r/m8*	0F 95	1/2
SETO *r/m8*	0F 90	1/2
SETP *r/m8*	0F 9A	1/2
SETPE *r/m8*	0F 9A	1/2
SETPO *r/m8*	0F 9B	1/2
SETS *r/m8*	0F 98	1/2
SETZ *r/m8*	0F 94	1/2
SGDT *m*	0F 01 /0	4
SIDT *m*	0F 01 /1	4
SHLD *r/m16,r16,imm8*	0F A4	4
SHLD *r/m32,r32,imm8*	0F A4	4
SHLD *r/m16,r16,*CL	0F A5	4/5
SHLD *r/m32,r32,*CL	0F A5	4/5
SHRD *r/m16,r16,imm8*	0F AC	4
SHRD *r/m32,r32,imm8*	0F AC	4
SHRD *r/m16,r16,*CL	0F AD	4/5

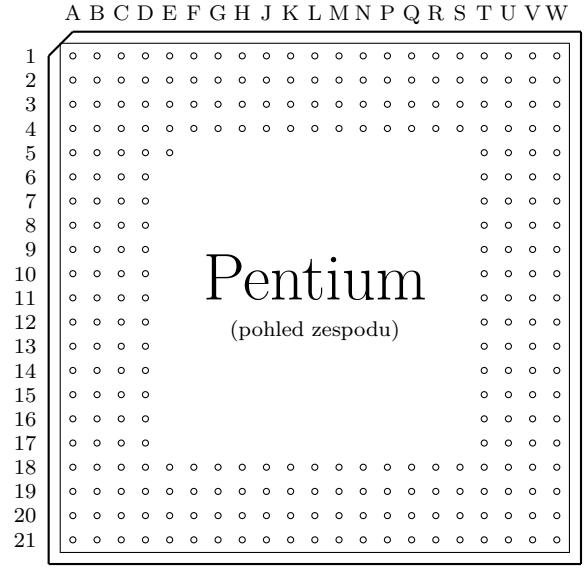
SHRD *r/m32,r32*,CL	0F AD	4/5
SLDT *r/m16*	0F 00 /0	2
SMSW *r/m16*	0F 01 /4	4
STC	F9	2
STD	FD	2
STI	FB	7
STOS *m8*	AA	3
STOS *m16*	AB	3
STOS *m32*	AB	3
STOSB	AA	3
STOSW	AB	3
STOSD	AB	3
STR *r/m16*	0F 00 /1	2
SUB AL,*imm8*	2C ib	1
SUB AX,*imm16*	2D iw	1
SUB EAX,*imm32*	2D id	1
SUB *r/m8,imm8*	80 /5 ib	1/3
SUB *r/m16,imm16*	81 /5 iw	1/3
SUB *r/m32,imm32*	81 /5 id	1/3
SUB *r/m16,imm8*	83 /5 ib	1/3
SUB *r/m32,imm8*	83 /5 ib	1/3
SUB *r/m8,r8*	28 /r	1/3
SUB *r/m16,r16*	29 /r	1/3
SUB *r/m32,r32*	29 /r	1/3
SUB *r8,r/m8*	2A /r	1/2
SUB *r16,r/m16*	2B /r	1/2
SUB *r32,r/m32*	2B /r	1/2
TEST AL,*imm8*	A8 ib	1
TEST AX,*imm16*	AD iw	1
TEST EAX,*imm32*	AD id	1
TEST *r/m8,imm8*	F6 /0 ib	1/2
TEST *r/m16,imm16*	F7 /0 iw	1/2
TEST *r/m32,imm32*	F7 /0 id	1/2
TEST *r/m8,r8*	84 /r	1/2
TEST *r/m16,r16*	85 /r	1/2
TEST *r/m32,r32*	85 /r	1/2
VERR *r/m16*	0F 00 /4	7
VERW *r/m16*	0F 00 /5	7
WAIT	9B	1
WBINVD	0F 09	2000+
WRMSR	0F 30	30-45
XADD *r/m8,r8*	0F C0 /r	3/4
XADD *r/m16,r16*	0F C1 /r	3/4

XADD *r/m32,r32*	0F C1 /r	3/4
XCHG AX,*r16*	90+rw	2
XCHG *r16*,AX	90+rw	2
XCHG EAX,*r32*	90+rd	2
XCHG *r16*,EAX	90+rd	2
XCHG *r/m8,r8*	86 /r	3
XCHG *r8,r/m8*	86 /r	3
XCHG *r/m16,r16*	87 /r	3
XCHG *r16,r/m16*	87 /r	3
XCHG *r/m32,r32*	87 /r	3
XCHG *r32,r/m32*	87 /r	3
XLAT *m8*	D7	4
XLATB	D7	4
XOR AL,*imm8*	34 ib	1
XOR AX,*imm16*	35 iw	1
XOR EAX,*imm32*	35 id	1
XOR *r/m8,imm8*	80 /6 ib	1/3
XOR *r/m16,imm16*	81 /6 iw	1/3
XOR *r/m32,imm32*	81 /6 id	1/3
XOR *r/m16,imm8*	83 /6 ib	1/3
XOR *r/m32,imm8*	83 /6 ib	1/3
XOR *r/m8,r8*	30 /r	1/3
XOR *r/m16,r16*	31 /r	1/3
XOR *r/m32,r32*	31 /r	1/3
XOR *r8,r/m8*	32 /r	1/2
XOR *r16,r/m16*	33 /r	1/2
XOR *r32,r/m32*	33 /r	1/2

B. Popisy signálů

B.1 Popis signálů procesoru Pentium

Procesor je integrován do čtvercového keramického integrovaného obvodu, který má vývody na spodním povrchu (PGA – Pin Grid Array). Obvod má 273 vývodů. Je vyroben technologií BiCMOS. K procesoru jsou též zkonstruovány další podpůrné obvody: řadič vyrovnávací paměti 82496 Cache Controller a vlastní externí vyrovnávací paměť 82491 Cache SRAM. V dalším popisu jsou signály označené „I“ (Input) vstupní, „O“ (Output) výstupní a „I/O“ jsou obousměrné. Zapojení procesoru je patrné z následujícího obrázku a tabulek. Nepopsané vývody jsou nezapojeny.



A_{31}	I/O	V19
A_{30}	I/O	W05
A_{29}	I/O	V20
A_{28}	I/O	V06
A_{27}	I/O	V21
A_{26}	I/O	T09
A_{25}	I/O	U19
A_{24}	I/O	U08
A_{23}	I/O	U20
A_{22}	I/O	U09
A_{21}	I/O	U21
A_{20}	I/O	U10
A_{19}	I/O	T10
A_{18}	I/O	U11
A_{17}	I/O	T11
A_{16}	I/O	U12
A_{15}	I/O	T12
A_{14}	I/O	U13
A_{13}	I/O	T13
A_{12}	I/O	U14
A_{11}	I/O	T14
A_{10}	I/O	U15
A_{9}	I/O	T15
A_{8}	I/O	U16
A_{7}	I/O	T16
A_{6}	I/O	U17
A_{5}	I/O	U18
A_{4}	I/O	W19
A_{3}	I/O	T17
D_{63}	I/O	H18
D_{62}	I/O	J19
D_{61}	I/O	K19
D_{60}	I/O	L19
D_{59}	I/O	H19
D_{58}	I/O	J18
D_{57}	I/O	F20
D_{56}	I/O	H21
D_{55}	I/O	G19
D_{54}	I/O	E20
D_{53}	I/O	G18
D_{52}	I/O	C21
D_{51}	I/O	F18
D_{50}	I/O	C20
D_{49}	I/O	F19
D_{48}	I/O	B21
D_{47}	I/O	B20
D_{46}	I/O	E18
D_{45}	I/O	A21
D_{44}	I/O	D21
D_{43}	I/O	A20
D_{42}	I/O	D20
D_{41}	I/O	B19
D_{40}	I/O	D14
D_{39}	I/O	D15
D_{38}	I/O	D19
D_{37}	I/O	D16
D_{36}	I/O	C18
D_{35}	I/O	D17
D_{34}	I/O	C19
D_{33}	I/O	C17
D_{32}	I/O	D10
D_{31}	I/O	C10
D_{30}	I/O	D12
D_{29}	I/O	C09
D_{28}	I/O	D11
D_{27}	I/O	D09
D_{26}	I/O	C11
D_{25}	I/O	C08
D_{24}	I/O	B10
D_{23}	I/O	A10
D_{22}	I/O	C07
D_{21}	I/O	C16
D_{20}	I/O	D07
D_{19}	I/O	C15
D_{18}	I/O	C06
D_{17}	I/O	B09
D_{16}	I/O	D06
D_{15}	I/O	D05
D_{14}	I/O	D13
D_{13}	I/O	D04
D_{12}	I/O	C14
D_{11}	I/O	E05
D_{10}	I/O	C13
D_{9}	I/O	C12
D_{8}	I/O	F04
D_{7}	I/O	G04
D_{6}	I/O	B04
D_{5}	I/O	G03
D_{4}	I/O	C04
D_{3}	I/O	F03
D_{2}	I/O	E04
D_{1}	I/O	E03
D_{0}	I/O	D03
$\overline{\text{A20M}}$	I	U05
$\overline{\text{ADS}}$	O	P04
AHOLD	I	L02
AP	I/O	P03
$\overline{\text{APCHK}}$	O	W03
BE_0	O	U04
BE_1	O	Q04
BE_2	O	U06
BE_3	O	V01
BE_4	O	T06
BE_5	O	S04
BE_6	O	U07
BE_7	O	W01
$\overline{\text{BOFF}}$	I	K04
BP2	O	B02
BP3	O	B03
$\overline{\text{BRDY}}$	I	L04
BREQ	O	V02
BT_0	O	T08
BT_1	O	W21
BT_2	O	T07
BT_3	O	W20
$\overline{\text{BUSCHK}}$	I	T03
$\overline{\text{CACHE}}$	O	J04
CLK	I	K18
$D/\overline{C}$	O	V04
DP_0	I/O	H04
DP_1	I/O	C05
DP_2	I/O	A09
DP_3	I/O	D08
DP_4	I/O	D18
DP_5	I/O	A19
DP_6	I/O	E19

DP_7	I/O	E21	INTR	I	N18	PRDY	O	U03
$\overline{EADS}$	I	M03	INV	I	A01	PWT	O	S03
$\overline{EWBE}$	I	A03	IU	O	J02	RESET	I	L18
$\overline{FERR}$	O	H03	IV	O	B01	R/$\overline{S}$	I	R18
$\overline{FLUSH}$	I	U02	$\overline{KEN}$	I	J03	SCYC	O	R04
$\overline{FRCMC}$	I	M19	$\overline{LOCK}$	O	V03	$\overline{SMI}$	I	P18
$\overline{HIT}$	O	W02	M/$\overline{IO}$	O	A02	$\overline{SMIACT}$	O	T05
$\overline{HITM}$	O	M04	$\overline{NA}$	I	K03	TCK	I	T04
HLDA	O	Q03	NMI	I	N19	TDI	I	T21
HOLD	I	V05	PCD	O	W04	TDO	O	S21
IBT	O	T19	$\overline{PCHK}$	O	R03	TMS	I	P19
$\overline{IERR}$	O	C02	$\overline{PEN}$	I	M18	$\overline{TRST}$	I	S18
$\overline{IGNNE}$	I	S20	PM0/BP0	O	D02	WB/$\overline{WT}$	I	M02
INIT	I	T20	PM1/BP1	O	C03	W/$\overline{R}$	O	N03

U_{cc}: A04 A05 A06 A07 A08 A11 A12 A13 A14 A15 A16 A17 A18 C01 D01 E01 F01 F21 G01 G21 H01 J21 K21 L21 M21 N01 N21 P01 P21 Q01 Q18 Q21 R01 R21 S01 T01 U01 W06 W07 W08 W09 W10 W11 W12 W13 W14 W15 W16 W17 W18

GND: B05 B06 B07 B08 B11 B12 B13 B14 B15 B16 B17 B18 E02 F02 G02 G20 H02 H20 J01 J20 K01 K02 K20 L01 L20 M01 M20 N02 N20 P02 P20 Q02 Q20 R02 R20 S02 T02 V07 V08 V09 V10 V11 V12 V13 V14 V15 V16 V17 V18

Jednotlivé signály mají tento význam:

A_3 – A_{31} 32bitová adresová sběrnice (adresa paměti a V/V bran) adresující 32bitová dvojslova.

D_0 – D_{63} 64bitová obousměrná datová sběrnice.

$\overline{A20M}$ Maskování adresy podle pravidel 8086.

$\overline{ADS}$, **$\overline{BOFF}$**, **$\overline{BRDY}$**, **BREQ**, **$\overline{BUSCHK}$**, **D/$\overline{C}$**, **W/$\overline{R}$** (Address Status, Backoff, Burst Ready, Bus Check, Data/Code, Write/Read) Signály pro řízení sběrnice.

AHOLD, **$\overline{CACHE}$**, **WB/$\overline{WT}$** (Address Hold, Writeback/Writethrough) Signály pro řízení vnitřní vyrovnávací paměti.

AP, **$\overline{APCHK}$** (Address Parity, Address Parity Check) Signály pro kontrolu parity na adresové sběrnici.

$\overline{BE_0}$ – $\overline{BE_7}$ (Byte Enable) Bližší určení přenášených slabik v rámci 64 bitů datové sběrnice.

BT_0 – BT_3 (Branch Trace) Signály přenášené bity 0-2 cílové lineární adresy skoku a implicitní velikost operandu (BT_3) skokové instrukce.

CLK (Clock) Hodinový signál.

DP_0 – DP_7 (Data Parity) Paritní bit pro každou slabiku přenášenou po datové sběrnici.

$\overline{EADS}$, **$\overline{EWBE}$**, **$\overline{HIT}$**, **$\overline{HITM}$**, **INV** (Valid External Address, External Write Buffer Empty, Hit, Hit to a Modified Line, Invalidation) Signály pro řízení vnější vyrovnávací paměti.

$\overline{FERR}$ (Floating Point Error) Signál oznamuje vznik odmaskované numerické výjimky v FPU (podobně jako $\overline{ERROR}$ u předchůdců Intel486). Tento signál slouží pro kompatibilitu s předchůdci (viz str. 101).

$\overline{FLUSH}$ (Cache Flush) Pokyn k vyprázdnění vnitřní vyrovnávací paměti.

$\overline{FRCMC}$ (Functional Redundancy Checking Master/Checker) Signál je určen pro kontrolu činnosti procesoru jeho zdvojením (znásobením). Signálem se procesoru sděluje, zdali má procesor pracovat jako řídicí (Master) a řídit činnost sběrnic, nebo jako podřízený (Checker) a pouze kontrolovat svoje výsledky s hodnotami na sběrnicích. Pokud v tomto případě zjistí rozdíl, aktivuje signál $\overline{IERR}$.

GND (Ground) Společný vodič 0 V.

HLDA (Bus Hold Acknowledge) Potvrzení odpojení procesoru od sběrnice.

HOLD (Bus Hold Request) Žádost o přidělení lokální sběrnice od jiného zařízení.

$\overline{IERR}$ (Internal Error) Signál hlásí vnitřní chybu procesoru (chyba parity nebo rozdíl při zpracování téže informace více jednotkami).

$\overline{IGNNE}$ (Ignore Numeric Error Input) Ignorování chyb hlášených koprocesorem (viz str. 101).

INIT (Initialization) Signál uvádí procesor do počátečního stavu podobně jako signál RESET s tím rozdílem, že nenuluje obsah interních vyrovnávacích pamětí a registrů FPU (viz str. 105).

INTR (Maskable Interrupt) Signál žádosti o maskovatelné přerušení.

IBT (Instruction Branch) Indikace provádění skokové operace (viz též str. 155).

IU, **IV** (U-Pipe Instruction, V-Pipe Instruction) Signály oznamují stav zřetězeného zpracování instrukcí (viz též str. 155).

$\overline{KEN}$ (Cache Enable) Povoluje nebo zakazuje použití vnitřní vyrovnávací paměti.

$\overline{LOCK}$ (Bus Lock) Uzamčení sběrnice pro procesor, který nastavil $\overline{LOCK}$=L instrukčním prefixem LOCK.

M/$\overline{IO}$ (Memory/Input-Output) Rozlišuje, zda adresa patří paměti nebo V/V branám.

$\overline{\text{NA}}$ (Next Address) Slouží k zahájení výběru obsahu další adresy při proudovém zpracování.

NMI (Non-Maskable Interrupt) Signál nemaskovatelného přerušení.

PCD, **PWT** Signály přenášející hodnoty bitů PWT a PCD.

$\overline{\text{PCHK}}$ (Parity Check) Chyba parity na datové sběrnici.

$\overline{\text{PEN}}$ (Parity Enable) Signál určuje režim kontroly parity při čtení dat.

PM0/BP0, **PM1/BP1**, **BP2**, **BP3** (Performance Monitoring, Breakpoints Pins) Signály odpovídají ladicím registrům DR0-DR3. Procesor jimi indikuje rozpoznání ladicích bodů. Signály PM jsou určeny pro sledování výkonu procesoru.

PRDY Signál oznamuje, že procesor normálně ukončil činnost po přijetí signálu R/$\overline{\text{S}}$.

R/$\overline{\text{S}}$ Signál zastavuje činnost procesoru. Je určen pro ladění.

RESET Signál okamžitě ukončuje aktivitu procesoru a uvede jej do počátečního stavu.

SCYC (Split Cycle) Signálem se spojují po sobě jdoucí cykly LOCK.

$\overline{\text{SMI}}$ (System Management Interrupt) Signálem se zapíná režim správy systému System Management Mode. Viz též str. 131.

$\overline{\text{SMIACT}}$ (System Management Interrupt Active) Procesor oznamuje, že je v režimu správy systému System Management Mode.

TCK (Testability Clock) Kontrolní časování podle IEEE 1149.1.

TDI (Test Data Input) Signál je sériový vstup pro testovací logiku.

TDO (Test Data Output) Sériový výstup testovací logiky.

$\overline{\text{TRST}}$ (Test Reset) Signál uvádí testovací logiku do počátečního stavu.

U_{cc} Napájecí napětí +5 V.

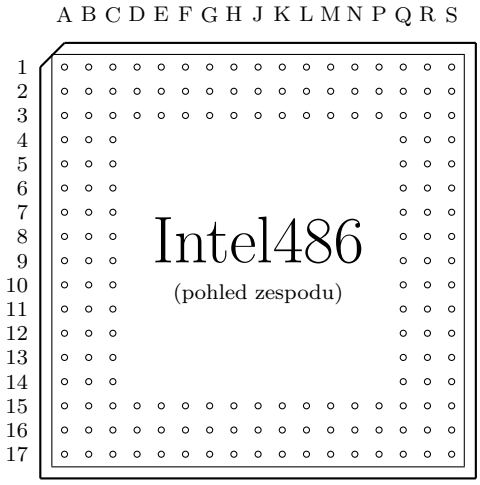
B.2 Popis signálů procesoru Intel Intel486

Procesor je integrován do PGA se 168 vývody (viz obr. B.1).

B.3 Popis signálů procesoru Intel Intel386

Procesor je integrován do PGA se 132 vývody.

$\overline{\text{BS16}}$ Volba 16bitového přenosu dat.



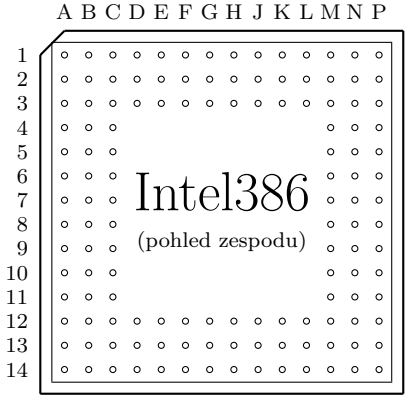
A_{31}	Q1	A_{15}	R7	D_{31}	B8	D_{15}	F3
A_{30}	P3	A_{14}	S5	D_{30}	C9	D_{14}	K3
A_{29}	P2	A_{13}	Q10	D_{29}	A8	D_{13}	D2
A_{28}	R1	A_{12}	S7	D_{28}	C8	D_{12}	G3
A_{27}	S1	A_{11}	R12	D_{27}	C6	D_{11}	C1
A_{26}	S2	A_{10}	S13	D_{26}	C7	D_{10}	E3
A_{25}	R2	A_9	Q11	D_{25}	B6	D_9	D1
A_{24}	Q6	A_8	R13	D_{24}	A6	D_8	F2
A_{23}	S3	A_7	Q13	D_{23}	A4	D_7	L3
A_{22}	Q7	A_6	S15	D_{22}	A2	D_6	L2
A_{21}	Q5	A_5	Q12	D_{21}	B2	D_5	J2
A_{20}	Q8	A_4	S16	D_{20}	A1	D_4	M3
A_{19}	Q4	A_3	R15	D_{19}	B1	D_3	H2
A_{18}	R5	A_2	Q14	D_{18}	C2	D_2	N1
A_{17}	Q3			D_{17}	D3	D_1	N2
A_{16}	Q9			D_{16}	J3	D_0	P1

$\overline{\text{A20M}}$	D15	$\overline{\text{BOFF}}$	D17	DP_1	F1	HOLD	E15	PCD	J17
$\overline{\text{ADS}}$	S17	$\overline{\text{BRDY}}$	H15	DP_2	H3	$\overline{\text{IGNNE}}$	A15	$\overline{\text{PCHK}}$	Q17
AHOLD	A17	$\overline{\text{BREQ}}$	Q15	DP_3	A5	INTR	A16	PWT	L15
$\overline{\text{BE}_0}$	K15	$\overline{\text{BS8}}$	D16	$\overline{\text{EADS}}$	B17	$\overline{\text{KEN}}$	F15	$\overline{\text{PLOCK}}$	Q16
$\overline{\text{BE}_1}$	J16	$\overline{\text{BS16}}$	C17	$\overline{\text{FERR}}$	C14	$\overline{\text{LOCK}}$	N15	$\overline{\text{RDY}}$	F16
$\overline{\text{BE}_2}$	J15	CLK	C3	$\overline{\text{FLUSH}}$	C15	$\overline{\text{M/IO}}$	N16	RESET	C16
$\overline{\text{BE}_3}$	F17	D/$\overline{\text{C}}$	M15	HLDA	P15	NMI	B15	$\overline{\text{W/R}}$	N17
$\overline{\text{BLAST}}$	R16	DP_0	N3						

GND: A7 A9 A11 B3 B4 B5 E1 E17 G1 G17 H1 H17 K1 K17 L1 L17 M1 M17 P17 Q2 R4 S6 S8 S9 S10 S11 S12 S14

U_{cc}: B7 B9 B11 C4 C5 E2 E16 G2 G16 H16 J1 K2 K16 L16 M2 M16 P16 R3 R6 R8 R9 R10 R11 R14

Obr. B.1 Zapojení procesoru Intel486



A_{31}	N2	A_{15}	F1	D_{31}	M5	D_{15}	M11
A_{30}	P1	A_{14}	E1	D_{30}	P3	D_{14}	P12
A_{29}	M2	A_{13}	E2	D_{29}	P4	D_{13}	P13
A_{28}	L3	A_{12}	E3	D_{28}	M6	D_{12}	N12
A_{27}	N1	A_{11}	D1	D_{27}	N5	D_{11}	N13
A_{26}	M1	A_{10}	D2	D_{26}	P5	D_{10}	M12
A_{25}	K3	A_9	D3	D_{25}	N6	D_9	N14
A_{24}	L2	A_8	C1	D_{24}	P7	D_8	L13
A_{23}	L1	A_7	C2	D_{23}	N8	D_7	K12
A_{22}	K2	A_6	C3	D_{22}	P9	D_6	L14
A_{21}	K1	A_5	B2	D_{21}	N9	D_5	K13
A_{20}	J1	A_4	B3	D_{20}	M9	D_4	K14
A_{19}	H3	A_3	A3	D_{19}	P10	D_3	J14
A_{18}	H2	A_2	C4	D_{18}	P11	D_2	H14
A_{17}	H1			D_{17}	N10	D_1	H13
A_{16}	G1			D_{16}	N11	D_0	H12

$\overline{\text{ADS}}$	E14	$\overline{\text{BS16}}$	C14	$\overline{\text{LOCK}}$	C10	W/$\overline{\text{R}}$	B10	INTR	B7
$\overline{\text{BE}_3}$	A13	$\overline{\text{BUSY}}$	B9	M/$\overline{\text{IO}}$	A12	CLK2	F12	NMI	B8
$\overline{\text{BE}_2}$	B13	D/$\overline{\text{C}}$	A11	$\overline{\text{NA}}$	D13	HOLD	D14	PEREQ	C8
$\overline{\text{BE}_1}$	C13	$\overline{\text{ERROR}}$	A8	$\overline{\text{READY}}$	G13	HLDA	M14	RESET	C9
$\overline{\text{BE}_0}$	E12								

GND: A2 A6 A9 B1 B5 B11 B14 C11 F2 F3 F14 J2 J3 J12 J13 M4 M8 M10 N3 P6 P14
U_{cc}: A1 A5 A7 A10 A14 C5 C12 D12 G2 G3 G12 G14 L12 M3 M7 M13 N4 N7 P2 P8

Obr. B.2 Zapojení procesoru Intel386

B.4 Popis signálů procesoru Intel286

Procesor Intel286 je umístěn ve čtvercovém integrovaném obvodu s 68 vývody. Vývody, které na obr. B.3 nemají přidělen název, jsou nezapojeny.

CAP Mezi tento vývod a vývod GND musí být zapojen kondenzátor kapacity 0,047 μF $\pm$20 % 12 V vyhlazující nežádoucí napěťové zákmity.

$\overline{\mathbf{S_0}}$,$\overline{\mathbf{S_1}}$ společně se signály COD/$\overline{\text{INTA}}$ a M/$\overline{\text{IO}}$ oznamují stav vnitřní fronty procesoru.

PEREQ Signálem koprocesor žádá procesor o vyslání operandu.

$\overline{\text{PEACK}}$ Signálem procesor oznamuje koprocesoru, že vysílá operand.

$\overline{\text{BUSY}}$ Aktivní úroveň signálu oznamuje, že koprocesor provádí výpočet. Signál je testován instrukcí WAIT.

$\overline{\text{ERROR}}$ Signálem koprocesor oznamuje chybový stav.

B.5 Popis signálů procesoru Intel 8086

Některé z vývodů integrovaného obvodu procesoru 8086 a 8088 mají dva různé významy podle toho, v jakém režimu procesor pracuje. Režim je definován signálem MN/$\overline{\text{MX}}$. Vysoká úroveň tohoto signálu (H) zapíná **minimální režim** a nízká úroveň (L) **maximální režim**. V maximálním režimu musí být procesor doplněn o další obvody.
Význam následujících signálů je společný pro oba režimy:

$\mathbf{AD_0}$ – $\mathbf{AD_{15}}$ 16bitová datová a současně část 20bitové adresové sběrnice v procesoru 8086.

$\mathbf{AD_0}$ – $\mathbf{AD_7}$ 8bitová datová a současně část 20bitové adresové sběrnice v procesoru 8088.

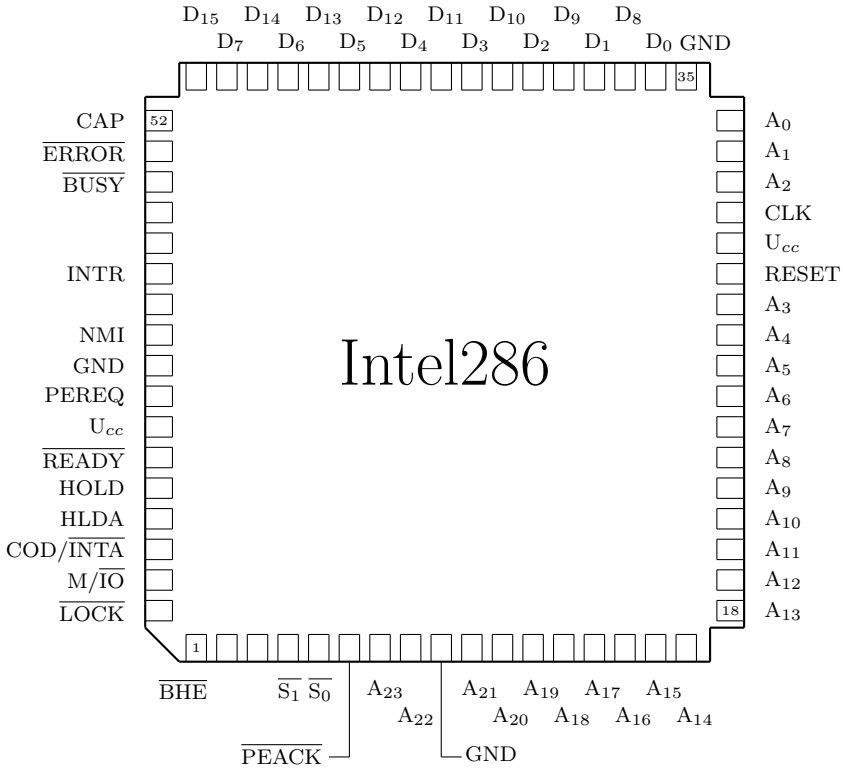
$\mathbf{A_8}$ – $\mathbf{A_{15}}$ Druhá část 20bitové adresové sběrnice v procesoru 8088.

$\mathbf{A_{16}/S_3}$ – $\mathbf{A_{19}/S_6}$ Nejvyšší bity 20bitové adresové sběrnice a část stavové informace.

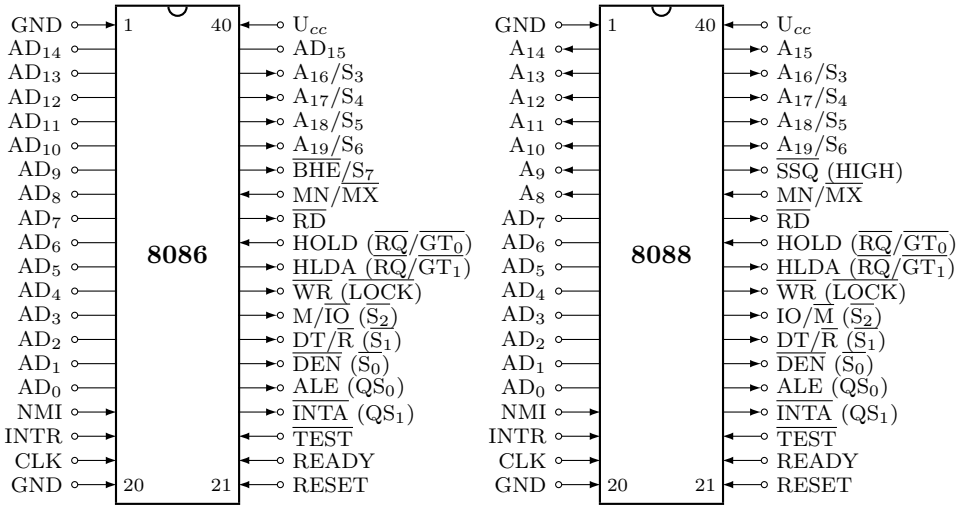
$\overline{\text{BHE}}$ Je-li $\overline{\text{BHE}}$=L a A_0=L, přenáší se 16bitové slovo, je-li $\overline{\text{BHE}}$=L a A_0=H, přenáší se slabika z liché adresy (vyšší polovina slova) a je-li $\overline{\text{BHE}}$=H a A_0=L, přenáší se slabika ze sudé adresy.

$\overline{\text{SSQ}}$ Stavová informace v minimálním režimu (v max. vždy v úrovni H) v procesoru 8088.

$\overline{\text{RD}}$ Čtení z paměti nebo V/V zařízení.



Obr. B.3 Zapojení procesoru Intel286



Obr. B.4 Zapojení procesorů Intel 8086 a 8088

READY Signál oznamuje procesoru, že data na sběrnici jsou platná.

$\overline{\text{TEST}}$ Signál testovatelný instrukcí WAIT. Při $\overline{\text{TEST}}$=L program pokračuje další instrukcí.

MN/$\overline{\text{MX}}$ Volba režimu procesoru maximální/minimální.

Další vývody mají odlišný význam pro jednotlivé režimy. V maximálním režimu platí:

$\overline{\text{S}_0}$ – $\overline{\text{S}_2}$ Stavová informace.

$\overline{\text{RQ}}/\overline{\text{GT}_0}$, $\overline{\text{RQ}}/\overline{\text{GT}_1}$ Obousměrné vývody používané pro řízení spolupráce procesorů na lokální sběrnici.

QS_0, QS_1 Signály oznamují stav vnitřní fronty procesoru.

V minimálním režimu platí:

IO/$\overline{\text{M}}$ Rozlišuje, zda adresa patří paměti nebo V/V v procesoru 8088.

$\overline{\text{WR}}$ Signál oznamující, že na datové sběrnici jsou platná výstupní data.

$\overline{\text{INTA}}$ Signál přijetí žádosti o přerušení.

ALE Sděluje, že na sběrnici je adresa (přepisuje adresu do adresového registru).

DT/$\overline{\text{R}}$ Informuje o směru přenosu po sběrnici: vysílání/čtení.

$\overline{\text{DEN}}$ Sděluje, že sběrnice je procesorem využívána (řídí zesilovače datové sběrnice).

Index instrukcí

Index

Literatura

[1] Brandejs, M.: Mikroprocesory INTEL 8086 – 80486, Grada a. s., Praha, 1991.

[2] Pentium Processor User's Manual, díl 1 až 3, Intel Corp., U. S. A., 1993.

Michal Brandejs

Mikroprocesory Intel – Pentium

Původní vydání Grada, Praha 1994
Další elektronické vydání Michal Brandejs, Brno 1. 9. 2010
http://www.fi.muni.cz/usr/brandejs/Brandejs_Mikroprocesory_Intel_Pentium_2010.pdf