Michal Brandejs

Mikroprocesory Intel 8086 – 80486

Michal Brandejs

Mikroprocesory Intel 8086 – 80486

Tento text byl vydán v nakladatelství Grada v roce 1991. Po vypršení platnosti nakladatelské smlouvy byl text autorem jako další vydání elektronicky zveřejněn dne 1. 9. 2010. Text lze šířit výhradně bezplatně a s uvedením autora a této copyrightové doložky. Text lze libovolně citovat, pokud je uveden odkaz na zdroj následovně:
Brandejs, M. Mikroprocesory Intel 8086 – 80486 [online]. Brno : Fakulta informatiky, Masarykova univerzita, 2010. Dostupný z WWW:
http://www.fi.muni.cz/usr/brandejs/Brandejs_Mikroprocesory_Intel_8086_80486_2010.pdf

Obsah

Seznam obrázku

Publikace, která se vám dostává do rukou, je o mikroprocesorech firmy Intel. Popisuje architekturu a programování nejznámějších mikroprocesorů z řady označované iAPX 86. Mezi nejznámější autor zařadil tyto:

8086/8088,
80286,
80386/386SX,
80486/486SX.

Publikace stejně detailně popisuje jak procesor 8086, tak i 80486. Protože každý novější typ této řady je rozšířením předchozího, lze např. 80286 popsat tak, že popíšeme 8086 a do popisu 80286 zařadíme jenom to, co je v 80286 nové. Stejně naložíme s popisem 80386 a 80486. Díky koncepci firmy Intel jsme tedy při studiu 80486, i když jsme museli přijmout informace i o všech předchozích členech řady, nečetli jediný zbytečný odstavec.

Dalším faktem, který hovoří pro popis celé řady, je existence nového režimu 80386 a 80486, pomocí kterého lze v těchto procesorech simulovat několik virtuálních procesorů 8086. Proto musíme vědět, co si můžeme v 8086 dovolit.

Čtení tohoto textu by měly ulehčit odkazy na stránky, na kterých byla právě diskutovaná problematika již dříve detailně popsána. Rovněž tak index napomáhá ke snadnému vyhledávání míst, kde je termín vysvětlen (v indexu je taková stránka označena symbolem ^{def}) nebo použit.

Předkládaný text obsahuje detailní popis činnosti instrukcí a neobsahuje popis konkrétního programovacího jazyka (asembleru), i když se vždy nelze nedotknout konkrétních programátorských problémů. Závěrečné příklady jsou sice vytvořeny pod operačním systémem MS-DOS, ale jejich myšlenky a obsahy kapitol věnovaných jednotlivým procesorům platí pro většinu operačních systémů.

Na závěr si dovoluji poděkovat Mgr. Zbyňku Sýkorovi za pomoc při sestavování popisu instrukcí a Davidu Tomanovi za pečlivé čtení rukopisu.

Autor

1 Řada procesorů Intel

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 iAPX 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 shodný s 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 matematickým koprocesorem 80287 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 Intel **80386** 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 80386 spolupracuje buď s koprocesorem 80387, nebo 80287. Procesor 80386SX je 32bitový procesor pro 16bitové vnější prostředí.

Rok 1989 přinesl dva rozdílné typy procesorů Intel, lišící se od předchozích pokročilejší výrobní technologií. Procesor Intel **80486** je programově slučitelný s procesorem 80386. Liší se pouze výkonem (je 3 až 5krát rychlejší) a na čipu má integrován rovněž matematický koprocesor a rychlou vyrovnávací paměť (Cache).

Druhý typ, který přinesl rok 1989, je 64bitový RISC procesor Intel **80860**. Tento procesor není, vzhledem k architektuře RISC, kompatibilní s řadou procesorů iAPX 86, a proto se o něm v tomto textu nezmiňujeme. Počítá se s ním pro super-výkonné stanice s obrovským výpočetním potenciálem v pohyblivé řádové čárce (17 MFLOPS na čipu) a pro grafické transformace v reálném čase.

2 Intel 8086

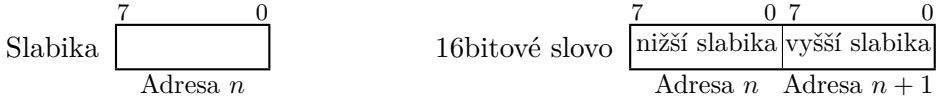
Procesor 8086 je prvním 16bitovým procesorem firmy Intel. S tímto procesorem je programově plně slučitelný procesor 8088 (procesory 80186 a 80188 se v praxi neujaly). Rovněž procesory 80286, 80386 a 80486 se v reálném režimu programují shodně. Tato kapitola tedy popisuje procesory 8086, 8088 a procesory 80286, 80386 a 80486 v reálném režimu, i když se v následujícím textu budeme odvolávat jenom na 8086.

Procesor 8086 pracuje s daty šířky 16 nebo 8 bitů a vytváří 20bitovou adresu pro adresaci 1 MB fyzické paměti. Virtuální paměť nepodporuje. Procesor je v pouzdře typu dual-inline se 40 vývody s jediným napájením +5 V (podrobnosti viz str. 225).

2.1 Typy dat

Paměť procesorů Intel je rozdělena do adresovatelných jednotek velikosti **slabiky** (= 8 bitů = 1 B). Bity číslujeme 0 až 7 s tím, že bit číslo 0 je bit nejnižšího řádu.

Do paměti dále ukládáme **16bitová slova** tak, že na **nižší adrese je slabika nižšího řádu**. Např. číslo 1234_{16} bude v paměti, vypsané se vzrůstající hodnotou adresy, mít tvar 3412_{16}. Adresa slova je adresa slabiky nižšího řádu.



Obr. 2.1. Formát slabiky a 16bitového slova v paměti

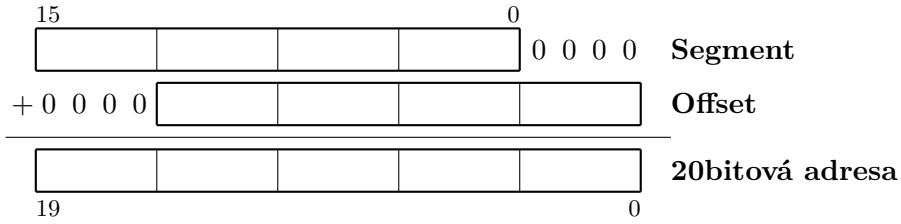
Ve slabice může být uloženo buď číslo bez znaménka v intervalu 0 až 255_{10},

nebo číslo se znaménkem ve **dvojkovém doplňkovém kódu** v intervalu −128 až 127. Znaménko je uloženo v bitu nejvyššího řádu a pro záporné číslo je znaménkový bit jedničkový. Záporné číslo se z kladného vytvoří tak, že se provede inverze všech bitů a přičte se jednička. Do 16bitového slova můžeme uložit číslo bez znaménka v intervalu 0 až 65 535 a se znaménkem v intervalu −32 768 až 32 767.

Dále procesor podporuje práci se řetězci slabik (viz str. 64) a dvojkově kódovanými desítkovými čísly (BCD kód – viz str. 69).

2.2 Adresace paměti procesoru 8086

Jelikož procesor zpracovává 16bitové hodnoty a adresa je 20bitová (20bitová adresa odpovídá 1 MB paměti), je zde technicky implementován mechanismus výpočtu fyzické adresy ze dvou složek. První složka, tvořící nižší bity fyzické adresy, se nazývá **offset** a druhá složka, tvořící vyšší bity, se nazývá **segment**. Fyzická adresa se vytvoří sečtením těchto dvou částí vzájemně posunutých o 4 bity (viz obr. 2.2).



Obr. 2.2. Vytváření 20bitové fyzické adresy v procesoru 8086

Adresu zapisujeme ve tvaru *segment : offset*. Zápis `01A5:0012` představuje tedy dvacetibitovou adresu `01A62`. Výpočet fyzické adresy se děje bez účasti programátora. 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 řešit programy větší než

64 KB, musí se za běhu procesu měnit hodnota segmentu.

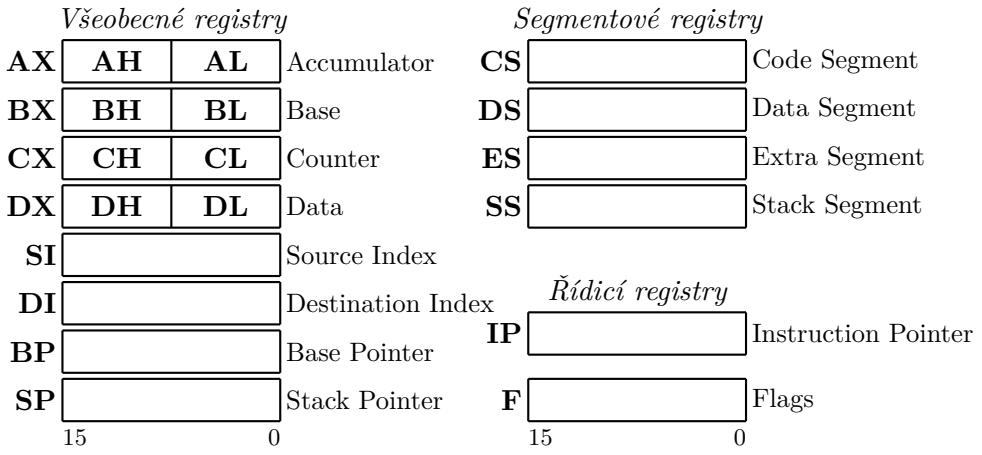
Procesor 8086 pro uložení segmentu poskytuje čtyři registry (viz dále). 16bitový registr **CS** (Code Segment) je určen pro výpočet adresy instrukce. Má-li instrukce skoku nebo volání podprogramu ve svém těle uložen pouze offset cílového místa, potom se segment při výpočtu fyzické adresy instrukce vezme právě z registru CS. Pro výpočet adresy dat, např. v instrukci naplnění registru hodnotou z paměti, se používá registr **DS** (Data Segment). Registr **SS** (Stack Segment) se použije při přístupu k zásobníku a v registru **ES** (Extra Segment) je adresa pomocného datového segmentu.

2.3 Registry procesoru 8086

Procesor 8086 je rozdělen na dvě jednotky: sběrnicovou a prováděcí. **Sběrnicová jednotka** (Bus Unit) vybírá instrukční kód z paměti a přenáší data po sběrnicích. **Prováděcí jednotka** (Execution Unit) provádí instrukce dodané sběrnicovou jednotkou. Obě jednotky jsou konstruovány tak, aby mohly pracovat nezávisle, a tím šetřit čas. Sběrnicová jednotka ukládá vybrané instrukční kódy do instrukční fronty a prováděcí jednotka je odtud vybírá. Sběrnicová jednotka vybírá instrukční kódy „do zásoby“ (je to zárodek proudového zpracování instrukcí, tzv. pipeline). Brzdou ovšem jsou instrukce měnící pořadí vykonávání instrukcí (instrukce skoků).

Procesor 8086 obsahuje 14 programátorovi přístupných 16bitových registrů (viz obr. 2.3). Registry AX, BX, CX a DX jsou registry pro všeobecné použití. Mohou se používat též jako 8bitové s označením (např. pro AX) AH (High) a AL (Low). **AX** (Accumulator) je všeobecný střádač. **BX** (Base) se používá jako bázový registr. Jeho obsah může být použit jako offsetová část adresy. **CX** (Counter) je čítač např. v instrukcích cyklů. **DX** (Data) je všeobecný datový registr. DX má speciální význam ve V/V instrukcích a při celočíselném násobení a dělení. Všechny tyto registry se používají pro uložení operandů a výsledků aritmetických a logických operací bez omezení. U ostatních instrukcí mají význam definovaný spolu s instrukcí.

Registry SP, BP, SI a DI se zpravidla používají pro uložení offsetu. Registr **SP** (Stack Pointer) obsahuje offset adresy vrcholu zásobníku (viz obr. 2.5 na straně 21). Kompletní adresa vrcholu zásobníku je tedy SS:SP. Registr **BP** (Base Pointer) je určen převážně pro uložení offsetové části adresy pro



Obr. 2.3. Registrová struktura procesoru 8086

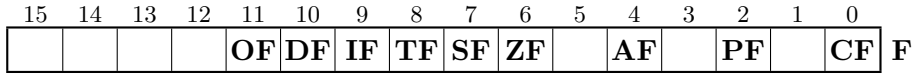
práci se zásobníkem. Nejčastěji se používá pro adresaci operandů předávaných do podprogramu prostřednictvím zásobníku. SI a DI jsou tzv. indexové registry. **SI** (Source Index) je indexový registr pro uložení offsetové části adresy zdrojového operandu a **DI** (Destination Index) cílového operandu. Toto přesně definované určení indexových registrů je nutné dodržet v instrukcích pracujících s řetězci. Jinak registry BP, SI a DI patří do skupiny všeobecných registrů. Postavení registru SP je výsadní (viz instrukce PUSH a POP).

Registr **IP** (Instruction Pointer) obsahuje vždy offsetovou část právě prováděné instrukce. Kompletní adresa právě prováděné instrukce je v registrech CS:IP.

Skupina segmentových registrů obsahuje čtyři registry pro uložení segmentové části adresy. Registr **CS** (Code Segment) ve spojení s IP je určen pro adresaci kódu (instrukcí), registr **DS** (Data Segment) ve spojení s registry BX a SI adresuje data, registr **ES** (Extra Segment) ve spojení s DI adresuje rovněž data a registr **SS** (Stack Segment) adresuje zásobník ve spojení s registry SP a BP. Logické slučování segmentových a offsetových registrů bude popsáno na obr. 2.8 na str. 28.

Příznakový registr **F** (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) jsou *příznaky nastavované programátorem*, který jimi řídí chod procesoru. Jedná se o zablokování přerušovacího systému počítače, zapnutí krokovacího režimu procesoru a změnu směru zpracovávání řetězcových operací. Z 16bitového registru F je využito 9 bitů. Rozložení funkčních bitů je patrné z obr. 2.4 a jednotlivé příznaky mají následující význam:



Obr. 2.4. Příznakový registr procesoru 8086

CF (Carry Flag) se nastaví na jedničku, jestliže při právě provedené aritmetické operaci došlo k přenosu z nejvyššího bitu, a to jak při práci s 8, tak 16bitovým 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. 69).

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 (výsledek spadá mimo rozsah zobrazení). Procesor oznámí přeplnění, pokud se přenos do znaménkového bitu nerovnal přenosu ze znaménkového bitu.

TF (Trap Flag) uvádí procesor do krokovacího režimu, ve kterém je po provedení první instrukce generováno přerušení (INT 1). Příznak nastavují různé ladicí systémy, které tímto režimem po jednotlivých instrukcích krokují laděný program. Ladicí systém se aktivuje generovaným přerušením 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[1]). Nastavení příznaku na jedničku (instrukcí STI) přerušení povoluje. Maskovat lze přerušení od vnějších zařízení (klávesnice, tiskárna atd.), nikoli programově simulovaná přerušení (instrukce INT) a NMI (přerušení ze skupiny nemaskovatelných přerušení).

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

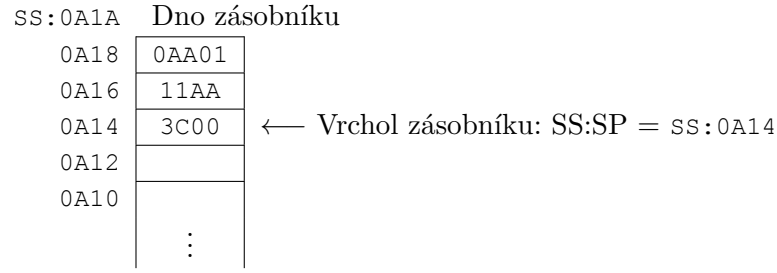
Programování v asembleru vyžaduje detailní znalost vlivu instrukcí na jednotlivé příznaky, ty se buď nastavují, nebo nechávají nedotčeny. Proto při studiu instrukcí je nutné těmto informacím věnovat pozornost.

2.4 Zásobník

Řada instrukcí procesoru 8086 pracuje se zásobníkem. Zásobník procesor implementuje jako strukturu LIFO (Last In - First Out) kdekoli v operační paměti. Všechny odkazy na zásobník jsou adresovány přes registr SS. Práci se zásobníkem si vysvětlíme na příkladu, v němž je dno zásobníku na adrese

[1] Všechny důležité signály procesoru jsou stručně popsány v příloze.

SS:0A1A. Zásobník byl do současného stavu naplněn posloupností instrukcí, které zapsaly hodnoty: 0AA01, 11AA, 3C00 (viz obr. 2.5).



Obr. 2.5. Zásobník procesoru 8086

Dvě základní operace pracující se zásobníkem vykonávají instrukce PUSH (vložení do zásobníku) a POP (výběr ze zásobníku). Registrem řídícím výběr a zápis do zásobníku je registr **SP** (Stack Pointer), který obsahuje offsetovou část adresy právě zapsané položky.
Instrukce PUSH provede činnosti v následujícím pořadí:

1. sníží obsah SP o dvě,

2. na adresu SS:SP uloží obsah 16bitového operandu.

Instrukce POP provede tyto akce:

1. operand naplní 16bitovým obsahem adresy SS:SP,

2. zvýší obsah SP o dvě.

Procesor nemá žádný prostředek, kterým by hlídal maximální mez naplnění zásobníku. Je na programátorovi, aby si pro uložení zásobníku vymezil dostatečnou kapacitu paměti.

Zásobníku používá také instrukce volání podprogramu pro ukládání návratové adresy, přerušovací systém a instrukce INT pro uložení registru příznaků a adresy, na níž byl proces přerušen. Přes zásobník rovněž vyšší programovací jazyky předávají podprogramům parametry.

2.5 Přerušení

Mechanismus přerušení (Interrupt) je v procesoru 8086 natolik zobecněn, že jej nevyužívají jen vnější V/V zařízení pro přivolání pozornosti operačního

systému a procesor pro oznamování výjimečných událostí (např. dělení nulou), ale také operační systém a programy v pamětech ROM pro zpřístupnění vlastních služeb programátorovi. Přerušení dělíme do skupin podle toho, čím jsou generovány:

technickými prostředky (vnější)	– nemaskovatelná (signál NMI)
	– maskovatelná (signál INTR)
programově (vnitřní)	– instrukcí INT n
	– chybou při běhu programu

Procesor rozlišuje 256 různých přerušení. Pro každé přerušení je v paměti uloženo 32 bitů adresy (přerušovací vektor) začátku obslužné programové rutiny. Adresy jsou zapsány v **tabulce přerušovacích vektorů** (viz obr. 2.6). Tabulka je uložena na začátku paměti od adresy 0000:0000 a má velikost 1 KB.

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. 2.6. Tabulka přerušovacích vektorů 8086

Přerušení způsobená vnějšími V/V zařízeními lze maskovat vynulováním příznaku **IF** instrukcí CLI. Tento zákaz přerušení se nevztahuje na vnitřní přerušení a NMI. Zakázané přerušení se povoluje instrukcí STI. Pro pochopení přerušovacího systému je nutné se detailně obeznámit s akcemi, které procesor v okamžiku přerušení provede. Předpokládejme, že nastalo přerušení číslo n nebo procesor dekódoval instrukci INT n. Potom se provedou činnosti v tomto pořadí:

1. do zásobníku se uloží registr příznaků (F),
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 $n \times 4$.

Pro přerušovací systém je instrukce nedělitelná, a proto se přerušení uplatní vždy až po dokončení instrukce. Po uplatnění vnějšího přerušení ukazuje IP uložené v zásobníku na instrukci, která nebyla dosud provedena. Ostatní přerušení v procesoru 8086 (instrukce INT n, přerušení generovaná procesorem, tj. INT 0, 1, 3 a 4) ukládají do zásobníku IP ukazující na následující instrukci.

Výše uvedenou posloupností akcí se předá řízení rutině pro obsluhu konkrétního přerušení, přičemž se uschovají nejdůležitější registry pro pozdější obnovení činnosti přerušeného procesu. Vynulováním příznaku IF se zajistí klidný (nepřerušitelný) průběh obsluhy vzniklého přerušení. Vynulováním TF, v případě krokovacího režimu procesoru, zajistíme správné aktivování ladicího systému.

Při programování rutin obsluhujících přerušení musí mít programátor na paměti, že všechny registry, jejichž obsah hodlá změnit, musí předem uschovat (nabízí se opět zásobník). Před ukončením obslužné rutiny je musí opět obnovit. Poněvadž je vyvoláním přerušení další zakázáno, je na programátorovi, zda během provádění přerušovací rutiny přerušení povolí (instrukcí STI) nebo ponechá přerušení zakázané s tím, že stav IF před přerušením nechá obnovit až návratem do přerušeného procesu.

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 (F).

V tabulce přerušovacích vektorů jsou některé vektory rezervovány pro přerušení, které generuje procesor 8086. Je na operačním systému, jak bude tato přerušení obsluhovat. Jde o tato přerušení:

INT n	*Význam*
0	Celočíselné dělení nulou (Divide by Zero)
1	Krokovací režim (Single-Step)
2	Nemaskovatelná přerušení (NMI)
3	Ladicí bod (Breakpoint Trap)
4	Přeplnění (Overflow Trap)

Přerušení INT 0 nastane při dělení nulou v instrukcích DIV a IDIV. Obsah CS:IP uložený do zásobníku ukazuje **za** instrukci, která přerušení způsobila. V procesorech vyšších typů (80286, …) je změna v tom, že obsah CS:IP ukazuje **na** instrukci, která přerušení způsobila. Při přenosu programového vybavení z 8086 na 80286 může proto při obsluze přerušení INT 0 dojít k zacyklení.

Je-li ladicím systémem nastaven příznak TF=1 (Trap Flag), vygeneruje se přerušení INT 1 po provedení první instrukce. Přerušením 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 systém 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.

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

Přerušení INT 2 se vygeneruje po přijetí signálu NMI aktivovaného vnějším zařízením v důsledku např. chyby parity v paměti. Signál NMI (na rozdíl od signálu INTR) nelze maskovat nulovou hodnotou příznaku IF. INT 2 je vnější nemaskovatelné přerušení.

Přerušení INT 3 se používá společně s přerušením INT 1 v ladicích systémech. Přerušení 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. Přerušení uloží do zásobníku obsah CS:IP ukazující na slabiku bezprostředně za touto instrukcí.

Ladicí systém uloží instrukci 0CCh na místo operačního kódu (resp. instrukčního prefixu) té instrukce, před kterou se má provádění programu zastavit. Ladicí systém si musí zapamatovat adresu a původní obsah přepsaného paměťového místa ze dvou důvodů: za prvé, aby po aktivaci přerušení INT 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.

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

Zažádá-li v jednom okamžiku o přerušení více zdrojů, procesor z nich vybere podle těchto priorit: INT 0 a INT, NMI, INTR a na posledním místě přerušení krokovacího režimu.

2.6 Ovládání V/V zařízení

Procesor prostřednictvím adresové a datové sběrnice komunikuje se dvěma typy zařízení. Prvním je vnitřní paměť počítače. S tou komunikuje tak, že na adresovou sběrnici nastaví adresu a po datové přenáší jedním nebo druhým směrem data. Druhým typem je celá skupina vstupních a výstupních zařízení (dále jen V/V zařízení). Procesor opět na adresovou sběrnici nastaví „číslo“ zařízení a po datové sběrnici přenáší data. Je-li na adresové sběrnici adresa paměti nebo „číslo“ V/V zařízení, sděluje signál M/$\overline{\text{IO}}$. Tak vypadá komunikace z vnějšku procesoru.

Uvnitř procesoru instrukcemi pracujícími s pamětí adresujeme paměť a instrukcemi pracujícími se V/V zařízeními adresujeme tato zařízení. Pomyslné adresy V/V zařízení ukazují na tzv. brány. Potom můžeme říci, že ovládání V/V zařízení připojených k procesoru 8086 se děje přes **V/V brány** (Port). Brány jsou stejně jako operační paměť adresovatelné po slabikách (brána má kapacitu 8 bitů), ale protože je datová sběrnice 16bitová, lze pracovat i

s 16bitovými slovy (dvěma sousedícími bránami).

Procesor pro adresování bran poskytuje pouze 16bitovou adresu, tj. může být 64 K 8bitových bran. Data se na specifikovanou bránu zapisují instrukcí OUT a čtou instrukcí IN. Některé z instalovaných bran jsou určeny pouze pro čtení, některé pouze pro zápis a jiné dovolují obě operace. Adresy bran pro daná zařízení a popis jejich použití je třeba hledat v dokumentaci příslušného zařízení.

2.7 Počáteční nastavení procesoru

Procesor je inicializován aktivní úrovní signálu RESET. V tom okamžiku procesor vynuluje příznakový registr, registr IP, segmentové registry a zruší obsah instrukční fronty. Registr CS naplní hodnotou 0FFFFh. Tzn., že první instrukce, kterou bude procesor po inicializaci signálem RESET zpracovávat, je umístěna na adrese 0FFFF:0000h.

2.8 Adresovací techniky

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

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

2.8.1 Registr

Operand je uložen v některém 16bitovém (AX, BX, CX, DX, SI, DI, SP, BP) nebo 8bitovém (AH, AL, BH, BL, CH, CL, DH, DL) registru. Některé instrukce také připouštějí uložení operandu v registrech CS, DS, ES, SS nebo F.

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.

2.8.2 Přímý operand

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

Dalších šest technik zpřístupňuje operandy uložené ve fyzické paměti. Základní operace výpočtu 20bitové fyzické adresy je součástí každé z technik. Její mechanismus byl popsán v odstavci 2.2 na straně 16. Část adresy nazývaná *segment* je uložena v některém ze čtyř segmentových registrů. V instrukci se potom určí, ze kterého z registrů CS, DS, ES nebo SS se segment vybere a použije se spolu s offsetem pro výpočet fyzické adresy.

Typ přístupu do paměti jednoznačně určuje, který ze segmentových registrů má být použit. Přiřazení segmentových registrů typům přístupu do paměti popisuje obr. 2.7.

Při přístupu k	*se použije registr*	*Operace*
instrukcím	**CS** (Code Segment)	Výběr operačního kódu nebo přímého operandu.
zásobníku	**SS** (Stack Segment)	Při všech přístupech k zásobníku nebo ve spojitosti s registrem BP.
datům	**DS** (Data Segment)	Při všech přístupech k datům v paměti vyjma zásobníku a přímých operandů. V řetězcových operacích segmentuje zdrojový operand.
alternativním datům	**ES** (Extra Segment)	V řetězcových operacích pro segmentování cílového operandu.

Obr. 2.7. Určení segmentových registrů

Každý registr, který lze použít pro uložení offsetové části adresy, má přiřazen jeden ze segmentových registrů (viz obr. 2.8). Není-li v instrukci specifikováno jinak, použije se toto tzv. **implicitní přiřazení**.

Šest adresovacích technik, o kterých bude diskutováno dále, se už týká pouze výpočtu offsetové části adresy. Offset se počítá z těchto tří složek:

Registr s offsetem	Implicitně použitý segmentový registr
SP	SS
BP	SS
BX	DS
SI	DS
DI	DS (ES v řetězcových operacích)
BP v kombinaci s SI nebo DI	SS
BX v kombinaci s SI nebo DI	DS

Obr. 2.8. Implicitní přiřazení segmentových registrů

1. přímé adresy (Displacement) uložené v instrukci,
2. báze (obsah registru BX nebo BP),
3. indexu (obsah registru SI nebo DI).

Každá z níže uvedených technik kombinuje tyto složky jiným způsobem.

2.8.3 Přímá adresa

Offsetové části adresy operandu umístěného ve fyzické paměti se přiřadí 16bitová přímá adresa uložená v instrukci. Přímá adresa v instrukci adresující data (MOV) se segmentuje přes DS a v instrukci adresující jinou instrukci (JMP) se segmentuje přes CS.

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 zatím 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`.

2.8.4 Nepřímá adresa

Offsetové části adresy se přiřadí obsah registru SI, DI, SP, BP nebo BX.
Příklad: `MOV AH,[BX]`. Je-li jméno registru BX (nebo SI, DI, SP, BP) uvedeno v hranatých závorkách, znamená to, že se do AH neuloží jeho obsah, ale obsah ležící na adrese uvedené v zadaném registru.

2.8.5 Bázovaná adresa

Offsetová část adresy se získá sečtením přímé adresy umístěné v instrukci a jednoho z bázových registrů (BX nebo BP).
Příklad: `MOV AH,[BP+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.

2.8.6 Indexovaná adresa

Offsetová část adresy se získá sečtením přímé adresy umístěné v instrukci a jednoho z indexových registrů (SI nebo DI).
Příklad: `MOV AH,[Adresa+SI]` a `MOV AH,Adresa[SI]`
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 SI nebo DI je indexem (ve slabikách) vybírajícím danou slabiku pole.

Indexování a bázování se v procesoru 8086 od sebe liší jenom použitím buď indexového, nebo bázového registru. V konečném efektu jsou to techniky shodné.

2.8.7 Kombinovaná adresa: báze+index

Tato adresní technika je kombinací dvou předchozích. Offset operandu je vypočítán součtem hodnoty uložené v jednom z bázových registrů (BX, BP) a hodnoty uložené v jednom z indexových registrů (SI, DI)
Příklad: `MOV AH,[BX][DI]` nebo `MOV AH,[BP+DI]`

S výhodou se dá tato adresovací technika použít v případech, kdy potřebujeme pracovat s dynamickou datovou strukturou. To znamená, že například počáteční adresa pole se bude během výpočtu měnit. Pak bázový registr udržuje počáteční adresu pole a indexový registr ukazuje na jednotlivé prvky tohoto pole.

2.8.8 Kombinovaná adresa: přímá+báze+index

V tomto případě je offset vypočítán jako součet obsahu bázového registru (BX nebo BP), indexového registru (SI nebo DI) a přímé adresy uvedené v instrukci.
Příklad: `MOV AH,Adresa[BX][DI]` nebo `MOV AH,[Adresa+BX+DI]`
Poznamenejme, že instrukce např. `MOV AH,[Adresa+BX+DI+1]` je stále stejného typu, protože překladač hodnotu `(Adresa+1)` vyčíslí a uloží jako přímou adresu.

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.

Všechny výše uvedené adresovací techniky lze shrnout do schématu:

$$\text{Vypočtená adresa} = \textit{přímá adresa} + \textit{báze} + \textit{index}\ .$$

Jednotlivé složky kombinujeme podle tabulky na obr. 2.9.

2.8.9 Změna segmentového registru

Na závěr této části poznamenejme, že v případě potřeby můžeme pro jednu instrukci „zrušit“ implicitní přiřazení segmentového registru offsetovému **explicitním uvedením instrukčního prefixu** před operačním kódem instrukce. Prefixy používané pro explicitní přiřazení segmentových registrů jsou uvedeny v tabulce na obr. 2.10.

Uvedení prefixu před operační kód se v asembleru zajistí zapsáním jména segmentového registru před offsetem odděleným dvojtečkou.

Adresovací technika	Vypočtená adresa =
přímá adresa	*přímá adresa*
nepřímá adresa	*báze*
bázovaná adresa	*přímá adresa + báze*
indexovaná adresa	*přímá adresa + index*
kombinovaná: báze+index	*báze + index*
kombinovaná: přímá+báze+index	*přímá adresa + báze + index*

Obr. 2.9. Adresovací techniky 8086

Segmentový registr	*Instrukční prefix*
DS (Data Segment)	3Eh
CS (Code Segment)	2Eh
SS (Stack Segment)	36h
ES (Extra Segment)	26h

Obr. 2.10. Prefixy měnící přiřazení segmentových registrů

Příklady:

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

2.9 Instrukční repertoár procesoru 8086

Instrukce v této publikaci, jak bývá v příručkách zvykem, nejsou řazeny abecedně. Jsou uspořádány v pořadí vhodném pro studium. Při abecedním vyhledávání lze použít index. Pro lepší pochopení rozdělíme instrukce do těchto skupin:

1. Instrukce MOV
2. Aritmetické instrukce
3. Logické instrukce
4. Rotace a posuvy
5. Větvení programu
6. Zásobník a příznakový registr
7. Přerušovací systém
8. Cykly
9. Ovládání V/V
10. Přesuny dat
11. Řetězcové instrukce
12. Instrukce BCD aritmetiky
13. Řídicí instrukce

Každou instrukci popíšeme v následujícím tvaru:

Zkratka instrukce *Jméno instrukce*

POPIS:

O	D	I	T	S	Z	A	P	C
?				*	?	*	0	1

Základní popis činnosti a operandů instrukce.

SYNTAX: ⌜ ZKRATKA *operand,operand*

Značka „ ⌜“ (levý horní roh listu) sděluje, že se zkratka instrukce nezapisuje od prvního sloupce (v prvním sloupci začíná případné návěští), ale od některého z dalších. První operand se od zkratky instrukce odděluje alespoň jednou mezerou. Operandy mezi sebou jsou odděleny čárkou. Za středníkem může následovat ještě komentář.

Tabulka pod jménem instrukce sděluje, jak byly provedením instrukce dotčeny příznaky v příznakovém registru.

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

Dále zpravidla následuje výčet možných kombinací operandů včetně jednoduchého příkladu a komentáře. Možné operandy jsou popsány pomocí následujících symbolů:

rel8 Relativní 8bitová adresa v intervalu −128 až 127.
rel16 Relativní 16bitová adresa v intervalu 0 až 65 535.
ptr16:16 32 bitů absolutní přímé adresy složené ze segmentu a offsetu. Viz též poznámka u *m16:16.*
r8 8bitový všeobecný registr.
r16 16bitový všeobecný registr.
imm8 8bitová přímá hodnota.
imm16 16bitová přímá hodnota.
r/m8 Buď 8bitový registr nebo offset slabiky v paměti.
r/m16 Buď 16bitový registr nebo offset 16bitového slova v paměti.
m16:16 32 bitů nepřímé adresy umístěné v paměti složené ze segmentu a offsetu (složky adresy jsou v paměti uloženy v obráceném pořadí – např. adresa `1234:5678h` bude mít v paměti, vypsané se vzrůstající adresou, tvar: `78 56 34 12h`).
Sreg Segmentový registr (ES, CS, SS, DS).

V příkladech použití instrukcí se vyskytují následující odkazy na paměť:

`Slab` představuje offset slabiky v paměti,
`Slovo` představuje offset 16bitového slova v paměti,
`DvojSlovo` představuje offset 32bitového slova v paměti.

Tyto odkazy mohou obsahovat libovolnou z šesti adresovacích technik pro práci s pamětí včetně explicitního uvedení segmentového registru.

2.9.1 Instrukce MOV

MOV — *Move Data*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce MOV přenáší obsah zdrojového operandu (tj. operandu stojícího více vpravo) do cílového operandu. Přitom obsah zdrojového operandu zůstává beze změny.

SYNTAX: MOV *cílový_operand,zdrojový_operand*

Podle typu zdrojového a cílového operandu je k dispozici několik variant instrukce MOV:

Instrukce	*Příklad*	*Komentář*
MOV *r/m8,r8*	`MOV AL,BL`	`; AL je naplněn obsahem BL`
	`MOV Slab,CH`	`; Slab je naplněn obsahem CH`
MOV *r/m16,r16*	`MOV BX,CX`	`; BX je naplněn obsahem CX`
	`MOV Slovo,DX`	`; Slovo je naplněn obsahem DX`
MOV *r8,r/m8*	`MOV CL,Slab`	`; CL je naplněn obsahem Slab`
MOV *r16,r/m16*	`MOV CX,Slovo`	`; CX je naplněn obsahem Slovo`
MOV *r/m16,Sreg*	`MOV AX,CS`	`; AX je naplněn obsahem CS`
	`MOV Slovo,SS`	`; Slovo je naplněn obsahem SS`
MOV *Sreg,r/m16*	`MOV DS,AX`	`; DS je naplněn obsahem AX`
	`MOV ES,Slovo`	`; ES je naplněn obsahem Slovo`
MOV *r/m8,imm8*	`MOV DL,32`	`; DL je naplněn konstantou 32`
	`MOV Slab,-128`	`; Slab je naplněn konstantou -128`
MOV *r/m16,imm16*	`MOV DI,32767`	`; DI je naplněn konstantou 32767`
	`MOV Slovo,-1`	`; Slovo je naplněn konstantou -1`

V případě instrukce MOV *Sreg,r/m16* není dovoleno na místě cílového operandu použít registr CS. I když to z výše uvedeného přehledu vyplývá, podotýkáme, že nelze kombinovat operandy různých délek (např. nelze `MOV Slovo,BL`).

Plní-li instrukce MOV registr SS, je automaticky zakázáno přerušení až do dokončení následující instrukce. Předpokládá se, že bude následovat instrukce plnění registru SP.

2.9.2 Aritmetické instrukce

ADD — *Integer Addition*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce ADD celočíselně přičítá obsah zdrojového operandu k obsahu cílového operandu a výsledek této operace umístí do cílového operandu. Obsah zdrojového operandu se nezmění. Pokud dojde při sčítání k přeplnění, nastaví se příznak OF na 1.

SYNTAX: ⌜ ADD *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
ADD *r/m8,imm8*	`ADD AL,80`	`; AL := AL + 80`
	`ADD Slab,-10`	`; Slab := Slab + (-10)`
ADD *r/m16,imm16*	`ADD CX,10000`	`; CX := CX + 10000`
	`ADD Slovo,30000`	`; Slovo := Slovo + 30000`
ADD *r/m16,imm8*	`ADD DI,2`	`; DI := DI + 2`
	`ADD Slovo,57`	`; Slovo := Slovo + 57`
ADD *r/m8,r8*	`ADD CH,CL`	`; CH := CH + CL`
	`ADD Slab,BL`	`; Slab := Slab + BL`
ADD *r/m16,r16*	`ADD AX,BX`	`; AX := AX + BX`
	`ADD Slovo,DX`	`; Slovo := Slovo + DX`
ADD *r8,r/m8*	`ADD CL,Slab`	`; CL := CL + Slab`
ADD *r16,r/m16*	`ADD BP,Slovo`	`; BP := BP + Slovo`

Je-li k cílovému operandu šířky 16 bitů přičítán přímý operand (konstanta) šířky 8 bitů, je rozšířen s respektováním znaménka na 16 bitů tak, že všech 8 horních bitů je naplněno hodnotou znaménkového bitu dolní slabiky (pro kladné číslo budou všechny horní bity nulové, pro záporné číslo budou všechny jedničkové).

ADC — *Integer Addition with Carry*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce ADC přičte k cílovému operandu obsah zdrojového operandu 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.

SYNTAX: ⌜ ADC *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
ADC *r/m8,r8*	`ADC AL,CL`	`; AL := AL + CL + CF`
	Další varianty viz ADC	

INC — *Increment by 1*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

POPIS:

Instrukce INC zvyšuje hodnotu operandu o jedničku. Tato instrukce neovlivňuje příznak CF.

SYNTAX: ⌜ INC *operand*

Instrukce	*Příklad*	*Komentář*
INC *r/m8*	`INC CL`	`; CL := CL + 1`
	`INC Slab`	`; Slab := Slab + 1`
INC *r/m16*	`INC SP`	`; SP := SP + 1`
	`INC Slovo`	`; Slovo := Slovo + 1`

SUB — *Integer Subtraction*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce SUB odčítá od obsahu cílového operandu obsah operandu zdrojového a výsledek této operace ukládá do cílového operandu. Obsah zdrojového operandu zůstane nezměněn.

SYNTAX: ⌜ SUB *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
SUB *r/m8,r8*	`SUB AL,CL`	`; AL := AL - CL`
	Další varianty viz ADD	

SBB — *Integer Subtraction with Borrow*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce SBB odčítá od obsahu cílového operandu obsah operandu zdrojového 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.

SYNTAX: ⌜ SBB *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
SBB *r/m8,r8*	`SBB AL,CL`	`; AL := AL - CL - CF`
	Další varianty 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.

Syntax: ⌜ DEC *operand*

Instrukce	*Příklad*	*Komentář*
DEC *r/m8*	`DEC BL`	`; BL := BL - 1`
	`DEC Slab`	`; Slab := Slab - 1`
DEC *r/m16*	`DEC AX`	`; AX := AX - 1`
	`DEC Slovo`	`; Slovo := Slovo - 1`

IMUL — *Signed Multiplication*

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	*

Popis:

Instrukce IMUL provádí znaménkové násobení (tzn. respektuje znaménka činitelů) obsahu registru AL nebo AX operandem. Je-li operand typu slabika, násobí se obsah registru AL a výsledek se umístí do registru AX (AH obsahuje vyšší řády a AL nižší). Je-li operand 16bitový, pak se násobí obsah registru AX a výsledek se uloží do registrového páru DX&AX tak, že registr DX obsahuje vyšších 16 bitů a registr AX nižších 16 bitů součinu. Je-li vyšší polovina výsledku násobení znaménkovým rozšířením nižší poloviny, jsou příznaky CF a OF nulovány.

Syntax: ⌜ IMUL *operand*

Instrukce	*Příklad*	*Komentář*
IMUL *r/m8*	`IMUL BL`	`; AX := AL * BL`
	`IMUL Slab`	`; AX := AL * Slab`
IMUL *r/m16*	`IMUL CX`	`; DX&AX := AX * CX`
	`IMUL Slovo`	`; DX&AX := AX * Slovo`

MUL — *Unsigned Multiplication*

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	*

POPIS:

Instrukce MUL má stejnou syntax a funkci jako instrukce IMUL s tím rozdílem, že provádí neznaménkové násobení (bit nejvyššího řádu se neuvažuje jako znaménkový).

IDIV — *Signed Divide*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

POPIS:

Instrukce IDIV provádí celočíselné znaménkové dělení obsahu registru AX nebo DX&AX (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. Pokud je podíl větší než rozsah zobrazení registru AL (AX), tj. při dělení nulou, nastane přerušení INT 0.
Zbytek má stejné znaménko jako dělenec.

SYNTAX: ⌜ IDIV *operand*

Instrukce	*Příklad*	*Komentář*
IDIV *r/m8*	`IDIV BL`	`; AL := AX ÷ BL a AH := AX` *mod* `BL`
	`IDIV Slab`	`; AL := AX ÷ Slab a AH := AX` *mod* `Slab`
IDIV *r/m16*	`IDIV CX`	`; AX := DX&AX ÷ CX a DX := DX&AX` *mod* `CX`
	`IDIV Slovo`	`; AX := DX&AX ÷ Slovo a DX := DX&AX` *mod* `Slovo`

DIV — *Unsigned Divide*

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

POPIS:

Instrukce DIV má stejnou syntax a funkci jako instrukce IDIV s tím rozdílem, že provádí neznaménkové dělení.

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.

SYNTAX: ⌜ NEG *operand*

Instrukce	*Příklad*	*Komentář*
NEG *r/m8*	`NEG AH`	`; AH := -AH`
	`NEG Slab`	`; Slab := -Slab`
NEG *r/m16*	`NEG SI`	`; SI := -SI`
	`NEG Slovo`	`; Slovo := -Slovo`

CBW — *Convert Byte to Word*

O	D	I	T	S	Z	A	P	C

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.

SYNTAX: ⌜ CBW

Instrukce	*Příklad*	*Komentář*
CBW	`CBW`	`; Převede obsah AL do AX se zachováním znaménka`

CWD — *Convert Word to Doubleword*

O	D	I	T	S	Z	A	P	C

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).

SYNTAX: ⌜ CWD

Instrukce	*Příklad*	*Komentář*
CWD	`CWD`	`; Převede obsah AX do DX&AX se zachováním znaménka`

CMP — *Compare Two Operands*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce CMP nastavuje příznaky, podle kterých lze porovnat obsah zdrojového operandu s obsahem cílového operandu. Příznaky nastaví tak, že odečte zdrojový operand od cílového operandu a podle výsledku nastaví bity příznakového registru. Instrukce neukládá rozdíl do cílového operandu (oba operandy zůstávají po provedení operace nezměněny). Nastavených příznaků lze využít některou instrukcí ze skupiny J*cond*.

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*.

SYNTAX: ⌜ CMP *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
CMP *r/m8,r8*	`CMP AL,CL`	`; F nastav podle AL - CL`
	Další varianty viz ADD	

2.9.3 Logické instrukce

AND *Logical AND*

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

POPIS:

Instrukce AND provádí logický součin obsahu zdrojového operandu s obsahem cílového 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

SYNTAX: ⌜ AND *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
AND *r/m8,imm8*	`AND AL,7`	`; AL := AL ∧ 7`
	`AND Slab,15`	`; Slab := Slab ∧ 15`
AND *r/m16,imm16*	`AND CX,0FFFh`	`; CX := CX ∧ 0FFFh`
	`AND Slovo,1FFFh`	`; Slovo := Slovo ∧ 1FFFh`
AND *r/m16,imm8*	`AND Slovo,0Fh`	`; Slovo := Slovo ∧ 000Fh`
	`AND Slovo,0FFh`	`; Slovo := Slovo ∧ 0FFFFh`
AND *r/m8,r8*	`AND CH,CL`	`; CH := CH ∧ CL`
	`AND Slab,BL`	`; Slab := Slab ∧ BL`
AND *r/m16,r16*	`AND AX,BX`	`; AX := AX ∧ BX`
	`AND Slovo,DX`	`; Slovo := Slovo ∧ DX`
AND *r8,r/m8*	`AND CL,Slab`	`; CL := CL ∧ Slab`
AND *r16,r/m16*	`AND BP,Slovo`	`; BP := BP ∧ Slovo`

Je-li zdrojový operand kratší než cílový (zdrojový je slabika a cílový 16bitové slovo), rozšíří se obsah zdrojového operandu s respektováním znaménka na délku cílového a potom se provede teprve operace.

OR — *Logical Inclusive OR*

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Popis:

Instrukce OR provádí logický součet obsahu zdrojového operandu s obsahem cílového 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

Syntax: ⌜ OR *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
OR *r/m8,imm8*	`OR AL,7`	`; AL := AL ∨ 7`
	Další varianty 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 operandu s obsahem cílového 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

Syntax: ⌜ XOR *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
XOR *r/m8,imm8*	`XOR AL,7`	`; AL := AL ≠ 7`
	Další varianty viz AND	

NOT — *One's Complement Negation*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce NOT nahradí obsah operandu jeho jedničkovým doplňkem. Jedničkový doplněk je inverze všech bitů (všechny jedničky se nahradí nulami a obráceně). Instrukce NOT neovlivňuje registr příznaků.

SYNTAX: ⌜ NOT *operand*

Instrukce	*Příklad*	*Komentář*
NOT *r/m8*	`NOT AH`	`; AH :=` $\overline{\texttt{AH}}$
	`NOT Slab`	`; Slab :=` $\overline{\texttt{Slab}}$
NOT *r/m16*	`NOT SI`	`; SI :=` $\overline{\texttt{SI}}$
	`NOT Slovo`	`; Slovo :=` $\overline{\texttt{Slovo}}$

TEST — *Logical Compare*

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

POPIS:

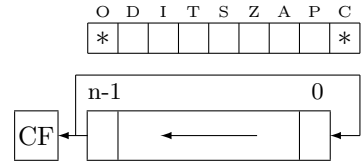
Instrukce TEST provádí logický součin (viz instrukce AND) obsahu zdrojového operandu s obsahem cílového operandu. Výsledek neuloží do cílového operandu, ale podle výsledku nastaví příznaky v příznakovém registru. Oba operandy jsou po provedení instrukce nezměněny.

SYNTAX: ⌜ TEST *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
TEST *r/m8,imm8*	`TEST AL,7`	`; F := AL ∧ 7`
	`TEST Slab,15`	`; F := Slab ∧ 15`
TEST *r/m16,imm16*	`TEST CX,0FFFh`	`; F := CX ∧ 0FFFh`
	`TEST Slovo,1FFFh`	`; F := Slovo ∧ 1FFFh`
TEST *r/m8,r8*	`TEST CH,CL`	`; F := CH ∧ CL`
	`TEST Slab,BL`	`; F := Slab ∧ BL`
TEST *r/m16,r16*	`TEST AX,BX`	`; F := AX ∧ BX`
	`TEST Slovo,DX`	`; F := Slovo ∧ DX`

2.9.4 Rotace a posuvy

ROL *Rotate Left*



POPIS:

Instrukce ROL provádí rotaci bitů cílového operandu o jeden nebo více bitů vlevo.

Každý bit cílového operandu se posune, bit nejvyššího řádu se přenese na pozici nejnižšího řádu a současně se zkopíruje do příznakového bitu CF. Hodnota zdrojového operandu udává, o kolik bitů se cílový operand rotuje. Na místě zdrojového operandu smí být buď přímý operand s hodnotou 1, nebo odkaz na registr CL, který obsahuje počet bitů. Při rotaci se žádný bit neztrácí.

SYNTAX: ROL *cílový_operand,zdrojový_operand*

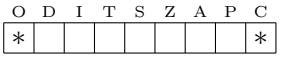
Instrukce	*Příklad*	*Komentář*
ROL *r/m8,1*	`ROL AL,1`	`; Rotace obsahu AL o 1 bit vlevo`
	`ROL Slab,1`	`; Rotace obsahu Slab o 1 bit vlevo`
ROL *r/m8,CL*	`ROL BH,CL`	`; Rotace BH o CL bitů vlevo`
	`ROL Slab,CL`	`; Rotace obsahu Slab o CL bitů vlevo`
ROL *r/m16,1*	`ROL BP,1`	`; Rotace obsahu BP o 1 bit vlevo`
	`ROL Slovo,1`	`; Rotace obsahu Slovo o 1 bit vlevo`
ROL *r/m16,CL*	`ROL DX,CL`	`; Rotace obsahu DX o CL bitů vlevo`
	`ROL Slovo,CL`	`; Rotace obsahu Slovo o CL bitů vlevo`

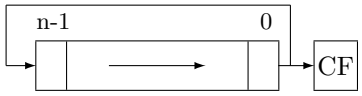
Procesor 8086 nijak neomezuje počet rotací umístěný v registru CL. Vyšší typy procesorů (80286, 80386, …) berou z CL v úvahu pouze nejnižších 5 bitů.

Hodnota příznaku OF je definována pouze po provedení rotace o jeden bit (zdrojový operand je 1). V jiných případech je hodnota OF nedefinována. Při rotaci vlevo je OF := CF $\neq$ bit$_{n-1}$ a při rotaci vpravo je OF := bit$_{n-1}$ $\neq$ bit$_{n-2}$, tj. OF se nastaví, pokud se hodnota CF nerovná při rotaci vlevo novému (při rotaci vpravo starému) nejvyššímu bitu.

ROR *Rotate Right*

Popis:



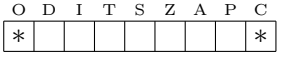


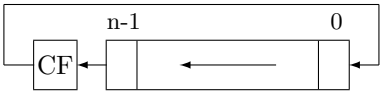
Instrukce ROR provádí rotaci bitů cílového operandu o jeden nebo více bitů vpravo.

Každý bit cílového operandu se posune vpravo, bit nejnižšího řádu se přenese na pozici nejvyššího řádu a současně se zkopíruje do příznakového bitu CF. Zdrojový operand udává počet rotací cílového operandu. Další podrobnosti jsou shodné s instrukcí ROL.

RCL *Rotate Left through Carry*

Popis:



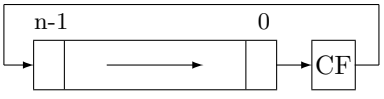


Instrukce RCL provádí rotaci bitů cílového operandu o jeden nebo více bitů vlevo.

Každý bit cílového 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:



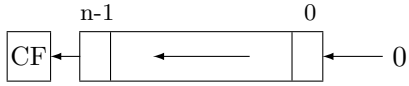
Instrukce RCR provádí rotaci bitů cílového operandu o jeden nebo více bitů vpravo.

Každý bit cílového 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 — *Shift Arithmetic Left*
SHL — *Shift Logical Left*

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	?	*	*



Obě instrukce SAL a SHL (synonymní označení jedné akce) posouvají všechny bity cílového operandu vlevo.

Přitom je původní hodnota nejvyššího bitu cílového operandu přenesena do příznaku CF a nejnižší uvolněný bit je naplněn nulou. Jde o operaci znaménkového aritmetického násobení 2. Hodnota zdrojového operandu udává, o kolik bitů se cílový operand posouvá. Na místě zdrojového operandu smí být buď přímý operand s hodnotou 1, nebo odkaz na registr CL obsahující počet bitů. Při posuvech se „ztrácí“ hodnota alespoň jednoho bitu.

Syntax: ⌜ SAL *cílový_operand,zdrojový_operand*
⌜ SHL *cílový_operand,zdrojový_operand*

Varianty instrukce viz ROL.

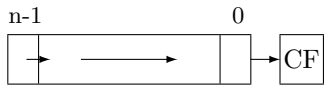
Procesor 8086 nijak neomezuje počet posuvů umístěný v registru CL. Vyšší typy procesorů (80286, 80386, …) berou z CL v úvahu pouze nejnižších 5 bitů.

Hodnota příznaku OF je definována pouze po provedení posuvu o jeden bit (zdrojový operand je 1). V jiných případech je hodnota OF nedefinována. V instrukcích SAL/SHL je OF vynulován tehdy, pokud nejvyšší 2 bity měly před provedením instrukce stejnou hodnotu. Byly-li různé, došlo během operace k aritmetickému přeplnění a OF je nastaven.

SAR — *Shift Arithmetic Right*

Popis:

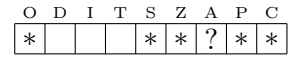
O	D	I	T	S	Z	A	P	C
*				*	*	?	*	*



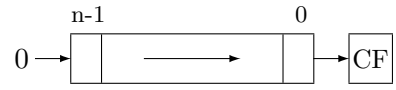
Instrukce SAR posouvá všechny bity cílového operandu vpravo.

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 cílového 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 SAL/SHL.

SHR — *Shift Logical Right*



Popis:

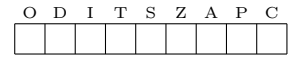


Instrukce SHR posouvá všechny bity cílového operandu vpravo.

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 SAL/SHL.

2.9.5 Větvení programu

JMP — *Unconditional Jump*



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:IP. Tuto operaci nazýváme **vzdálený skok** (Far Jump).

– Instrukce neměnící obsah segmentového registru CS. Instrukce obsahuje pouze „vzdálenost“ místa, kam se má provést skok, od místa výskytu instrukce. Tato relativní adresa se přičte ke stávajícímu obsahu registru IP. Potom podle délky skoku (tj. rozdílu adresy právě prováděné instrukce JMP a cílové adresy) rozlišujeme další dva typy:

  * Relativní adresa v instrukci JMP je 16bitová. Taková instrukce dovoluje předávat řízení uvnitř celého 64 KB segmentu tím, že k současnému obsahu IP přičte relativní adresu, která je číslo v intervalu 0 až 65 535 (adresa je číslo bez znaménka a ve skocích „vzad“ se při přičítání k IP ignoruje přeplnění). Tuto operaci nazýváme **blízký skok** (Near Jump).
  * Relativní adresa v instrukci JMP je 8bitová a je chápána jako číslo se znaménkem. Potom lze řízení předat pouze v intervalu −128 až 127 slabik od místa uložení právě prováděné instrukce JMP. 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 2.8 od strany 26.

SYNTAX:

```
JMP operand              ;  Blízký a nepřímý skok
JMP FAR PTR operand      ;  Vzdálený skok
JMP SHORT operand        ;  Krátký skok
```

Instrukce	*Příklad*	*Komentář*
JMP *rel8*	`JMP SHORT Navesti`	`; Krátký skok na Navesti`
		`;  IP := IP + vzdálenost Navesti`
JMP *rel16*	`JMP Navesti`	`; Blízký skok na Navesti`
		`;  IP := IP + vzdálenost Navesti`
JMP *ptr16:16*	`JMP FAR PTR Navesti`	`; Vzdálený skok na Navesti`

```
                                ;  CS:IP := segment:offset Navesti
JMP r/m16   JMP [BX]            ; Nepřímý skok na adresu v BX
                                ;  IP := BX
            JMP [Slovo]         ; Nepřímý skok podle obsahu Slovo
                                ;  IP := Slovo
JMP m16:16  JMP [DvojSlovo]     ; Nepřímý skok podle obsahu DvojSlovo
                                ;  CS:IP := DvojSlovo
```

J*cond* — *Conditional Jumps*

O	D	I	T	S	Z	A	P	C

POPIS:

Zkratkou J*cond* jsou zde výjimečně myšleny 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 skok zadaný 8bitovou relativní adresou. Neodpovídá-li, pokračuje se další instrukcí.

Instrukcí podmíněného skoku lze předávat řízení pouze v intervalu −127 až 128 slabik v okolí instrukce J*cond*. Další podrobnosti o krátkém skoku viz instrukce JMP. V instrukci J*cond* se neuvádí SHORT.

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 JCXZ se liší od ostatních instrukcí podmíněného skoku jen tím, že testuje místo jednobitových příznaků obsah registru CX. Pokud je CX=0, provede se krátký skok na specifikovanou adresu. Instrukce se používá pro řízení cyklů.

SYNTAX: ⌜ J*cond operand*

Instrukce *Příklad* *Komentář*

```
Jcond rel8 JZ   Navesti ; Krátký podmíněný skok při ZF=1
JCXZ rel8  JCXZ Navesti ; Krátký podmíněný skok při CX=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 zero	roven nulový	ZF=1
JNE/JNZ	not equal not zero	různý nenulový	ZF=0
JP/JPE	parity parity even	sudé parity	PF=1
JNP/JPO	not parity parity odd	liché parity	PF=0
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 not above nor equal	nz. menší	CF=1
JAE/JNB	above or equal not below	nz. větší nebo roven	CF=0
JBE/JNA	below or equal not above	nz. menší nebo roven	(CF=1)∨ ∨(ZF=1)
JA/JNBE	above not below nor equal	nz. větší	(CF=0)∧ ∧(ZF=0)
JL/JNGE	less not greater nor equal	z. menší	SF≠OF
JGE/JNL	greater or equal not less	z. větší nebo roven	SF=OF

JLE/JNG	less or equal not greater	z. menší nebo roven	(ZF=1)∨ ∨(SF≠OF)
JG/JNLE	greater not less nor equal	z. větší	(ZF=0)∧ ∧(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.

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.

Různé varianty instrukce CALL vyplývají z různých mechanismů zadávání cílové adresy a jsou shodné s instrukcí JMP kromě varianty krátkého skoku. Oproti instrukci JMP musí CALL ukládat do zásobníku návratovou adresu. V případě blízkého volání podprogramu se ukládá pouze offsetová část návratové adresy. U vzdáleného volání se do zásobníku vloží nejprve současný obsah registru CS, a teprve potom offsetová část návratové adresy.

Syntax: ⌜ CALL *operand* ; Blízké a nepřímé volání
CALL FAR PTR *operand* ; Vzdálené volání

Instrukce	*Příklad*	*Komentář*
CALL *rel16*	`CALL Navesti`	`; Blízké volání Navesti` `;  IP := IP + vzdálenost Navesti`
CALL *ptr16:16*	`CALL FAR PTR Navesti`	`; Vzdálené volání Navesti` `;  CS:IP := segment:offset Navesti`

CALL *r/m16*	`CALL [BX]`	`; Nepřímé volání podle obsahu BX`
		`;  IP := BX`
	`CALL [Slovo]`	`; Nepřímý skok podle obsahu Slovo`
		`;  IP := Slovo`
CALL *m16:16*	`CALL [DvojSlovo]`	`; Nepřímý skok podle DvojSlovo`
		`;  CS:IP := DvojSlovo`

Další podrobnosti viz JMP.

RET — *Near Return From Procedure*
RETF — *Far Return From Procedure*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce RET vrací řízení z podprogramu volaného instrukcí CALL. Návratovou adresu převezme z vrcholu zásobníku.

Z podprogramu volaného blízkým voláním CALL (v zásobníku je uložena jenom offsetová část návratové adresy) se návrat musí provést pouze instrukcí blízkého návratu RET (ze zásobníku převezme pouze jedno 16bitové slovo jako offsetovou část návratové adresy). Registr CS se provedením instrukce RET nezmění.

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 RETF (ze zásobníku převezme dvě 16bitová slova: nejprve offsetovou, pak segmentovou část návratové adresy), která návratovou adresou naplní registry CS:IP.

Za instrukcí RET/RETF se může jako přímá hodnota uvést číslo, které udává, kolik slabik 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 operandu k registru SP.

SYNTAX: ⌜ RET
RET *operand*
RETF
RETF *operand*

Instrukce	*Příklad*	*Komentář*
RET	`RET`	`; Návrat z podprogramu`
RET *imm16*	`RET 2`	`; Návrat a odstranění 2 slabik ze zásobníku`
RETF	`RET`	`; Vzdálený návrat z podprogramu`
RETF *imm16*	`RET 6`	`; Vzdálený návrat a odstranění 6 slabik ze zás.`

Poznamenejme, že chybná kombinace typů instrukcí CALL (blízké nebo vzdálené volání) a RET povede ke zhroucení programu.

2.9.6 Zásobník a příznakový registr

PUSH — *Push a Word onto the Stack*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce PUSH uloží hodnotu operandu do zásobníku. Operaci provede ve dvou krocích (viz též str. 21): sníží obsah registru ukazatele vrcholu zásobníku (SP) o 2 a na nový vrchol zásobníku (SS:SP) uloží obsah operandu. Operandem instrukce může být 16bitové slovo v paměti nebo 16bitový registr (všeobecný nebo segmentový).

SYNTAX: ⌜ PUSH *operand*

Instrukce	*Příklad*	*Komentář*
PUSH *m16*	`PUSH Slovo`	`; Do zásobníku uloží obsah Slovo`
PUSH *r16*	`PUSH AX`	`; Do zásobníku uloží obsah AX`
PUSH *Sreg*	`PUSH CS`	`; Do zásobníku uloží obsah CS`

Instrukce PUSH SP ukládá do zásobníku novou hodnotu SP (sníženou o 2). Tím se liší od vyšších typů procesorů (80286, …), které ukládají hodnotu SP před provedením instrukce (nesníženou 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.

POP — *Pop a Word from the Stack*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce POP vybere slovo ze zásobníku a uloží je do operandu instrukce. Činnost provede ve dvou krocích (viz též str. 21): z vrcholu zásobníku (SS:SP) vezme slovo a naplní jím operand, registr ukazatele vrcholu zásobníku zvýší o 2. Operandem instrukce může být 16bitový všeobecný registr, segmentový registr (**kromě CS**) nebo 16bitové slovo v paměti.

SYNTAX: ⌜ POP *operand*

Instrukce	*Příklad*	*Komentář*
POP *m16*	`POP Slovo`	`; Vybere hodnotu a uloží ji do Slovo`
POP *r16*	`POP BX`	`; Vybere hodnotu a uloží ji do BX`
POP DS	`POP DS`	`; Vybere hodnotu a uloží ji do DS`
POP ES	`POP ES`	`; Vybere hodnotu a uloží ji do ES`
POP SS	`POP SS`	`; Vybere hodnotu a uloží ji do SS`

Instrukce POP SS zakazuje všechna přerušení (vč. NMI) do skončení následující instrukce. Předpokládá se, že bude následovat instrukce POP SP. Tím procesor umožní naplnit registry SS:SP, aniž by na krátký okamžik byl k dispozici chybný ukazatel do zásobníku.

PUSHF — *Push Flag Register onto the Stack*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce PUSHF ukládá na vrchol zásobníku registr příznaků (viz též instrukci PUSH a tvar registru příznaků na str. 19).

SYNTAX: ⌜ PUSHF

Instrukce	*Příklad*	*Komentář*
PUSHF	`PUSHF`	`; Do zásobníku uloží obsah registru F`

POPF — *Pop from the Stack into the Flag Register*

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

POPIS:

Instrukce POP vybere z vrcholu zásobníku hodnotu a uloží ji do registru příznaků (viz též instrukci POP a tvar registru příznaků na str. 19).

SYNTAX: ⌜ POPF

Instrukce	*Příklad*	*Komentář*
POPF	`POPF`	`; Vybere hodnotu a uloží do registru F`

STx — *Set x Flag*
CLx — *Clear x Flag*

O	D	I	T	S	Z	A	P	C
	*	*						*

POPIS:

Do této skupiny patří instrukce nastavující (STx) a nulující (CLx) příznaky **I**F, **D**F a **C**F z příznakového registru.

SYNTAX: ⌜

STx	;	Jedničkování příznaku x
CLx	;	Nulování příznaku x

Instrukce	*Příklad*	*Komentář*
STI	`STI`	`; IF := 1 (povolení přerušení)`
STD	`STD`	`; DF := 1 (zpracovávání řetězců odzadu)`
STC	`STC`	`; CF := 1 (nastavení příznaku přenosu)`
CLI	`CLI`	`; IF := 0 (zákaz přerušení)`
CLD	`CLD`	`; DF := 0 (zpracování řetězců odpředu)`
CLC	`CLC`	`; CF := 0 (nulování příznaku přenosu)`

Je-li přerušení zakázáno po delší dobu, je pravděpodobné, že se budou požadavky na přerušení, přicházející od různých zdrojů, hromadit (přijde-li další přerušení od stejného zdroje, předchozí se ztratí). Z toho vyplývá, že po odblokování přerušovacího systému se s poměrně velkou pravděpodobností uplatní čekající přerušení. Aby se v zásobníku nehromadilo příliš mnoho adres přerušených procesů, povolí instrukce STI přerušení **až po dokončení následující instrukce**. Předpokládá se, že za touto instrukcí následuje návrat do vyšší vrstvy (RET), čímž se ze zásobníku průběžně odstraňují adresy přerušených procesů.

Přerušení se povolí po provedení následující instrukce jenom tehdy, pokud jej tato nezakázala. Např. během provádění posloupnosti instrukcí `STI` a `CLI` (ani po provedení posloupnosti) není přerušení povoleno.

CMC — *Complement Carry Flag*

O	D	I	T	S	Z	A	P	C
								*

Popis:

Instrukce CMC invertuje obsah bitu CF v registru příznaků F. Pokud byla hodnota CF=1, pak nastaví CF=0, a naopak.

Syntax: ⌜ CMC

Instrukce	*Příklad*	*Komentář*
CMC	`CMC`	`; CF :=` $\overline{\texttt{CF}}$

LAHF — *Load Flags into AH Register*

O	D	I	T	S	Z	A	P	C

Popis:

Instrukce LAHF kopíruje nižších 8 bitů příznakového registru do registru AH (viz tvar registru příznaků na str. 19).

Syntax: ⌜ LAHF

Instrukce	*Příklad*	*Komentář*

LAHF `LAHF ; Uloží do AH nižší slabiku registru F`

SAHF — *Store AH Register into Flags Register*

O	D	I	T	S	Z	A	P	C
				*	*	*	*	*

POPIS:

Instrukce SAHF naplní dolních 8 bitů příznakového registru hodnotou uloženou v registru AH (viz tvar registru příznaků na str. 19).

SYNTAX: ⌜ SAHF

Instrukce	*Příklad*	*Komentář*
SAHF	`SAHF`	`; Zapíše do nižší slabiky F registr AH`

2.9.7 Přerušovací systém

INT — *Call to Interrupt Procedure*
INTO — *Call to Interrupt Procedure if Overflow*

O	D	I	T	S	Z	A	P	C
		0	0					

POPIS:

Instrukce INT generuje programové přerušení. Operand instrukce (0 až 255) je použit jako index do tabulky přerušovacích vektorů (viz obr. 2.6 na str. 22), kde je uložena adresa rutiny obsluhující přerušení odkazované operandem. V rámci této instrukce se do zásobníku uloží registr příznaků, CS:IP **ukazující na instrukci následující** za INT, vynulují se příznaky IF a TF a naplní se registry CS:IP (viz též str. 22). Instrukce se používá zpravidla 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).

Instrukce INTO je jednoslabiková instrukce (operační kód 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 kódem 0CCh. Tato se používá pro aktivaci ladicího systému (ladicí bod – Breakpoint). Ostatní instrukce INT *n* jsou dvouslabikové.

SYNTAX: ⌜ INT *operand*
INTO

Instrukce	*Příklad*	*Komentář*
INT *imm8*	`INT 10h`	`; Vyvolání přerušení číslo 10h`
INTO	`INTO`	`; Vyvolání přerušení číslo 4 při OF=1`

IRET — *Return from Interrupt*

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

POPIS:

Instrukce IRET provádí návrat z rutiny vyvolané přerušením nebo instrukcí INT. Z vrcholu zásobníku jsou instrukcí odstraněna tři slova a uložena do registrů IP, CS a F (viz též str. 23).

SYNTAX: ⌜ IRET

Instrukce	*Příklad*	*Komentář*
IRET	`IRET`	`; Návrat z přerušení`

2.9.8 Cykly

LOOP — *Unconditional Loop Control*

O	D	I	T	S	Z	A	P	C

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, 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.

SYNTAX: ⌜ LOOP *operand*

LOOP*cond* — *Conditional Loop Control*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce podmíněného provádění cyklu 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.

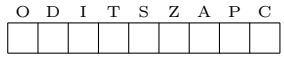
SYNTAX: ⌜ LOOP*cond* *operand*

Instrukce	*Příklad*	*Komentář*
LOOPE *rel*8	`LOOPE Navesti`	`; CX:=CX-1 a skok při (CX≠0)∧(ZF=1)`
LOOPZ *rel*8	`LOOPZ Navesti`	`; CX:=CX-1 a skok při (CX≠0)∧(ZF=1)`
LOOPNE *rel*8	`LOOPNE Navesti`	`; CX:=CX-1 a skok při (CX≠0)∧(ZF=0)`
LOOPNZ *rel*8	`LOOPNZ Navesti`	`; CX:=CX-1 a skok při (CX≠0)∧(ZF=0)`

2.9.9 Ovládání V/V

IN — *Input from Port*

Popis:



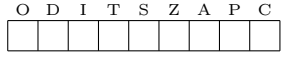
Instrukce IN přenáší data o délce slabiky nebo slova z V/V brány do registru AL nebo AX podle cílového operandu. Zdrojový operand určuje adresu V/V brány. Zde může být zadána buď přímá 8bitová hodnota bez znaménka (0 až 255), nebo registr DX (obsahující hodnotu 0 až 65 535). Pokud je použita přímá 8bitová hodnota, je vyšší slabika adresy brány nulová. Přenáší-li se slovo, uloží se do cílového registru obsahy bran na adresách *zdrojový_operand* a *zdrojový_operand+1*.

Syntax: ⌜ IN *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
IN AL,*imm8*	`IN AL,0Fh`	`; Přenos slabiky z brány 0Fh do AL`
IN AX,*imm8*	`IN AX,0Fh`	`; Přenos slova z bran 0Fh a 10h do AX`
IN AL,DX	`IN AL,DX`	`; Přenos slabiky z brány podle DX do AL`
IN AX,DX	`IN AX,DX`	`; Přenos slova z brány podle DX do AX`

OUT — *Output to Port*

Popis:



Instrukce OUT přenáší data v délce slabiky nebo slova z registru AL nebo AX do V/V brány. Cílový operand určuje adresu V/V brány (viz instrukci IN).

Syntax: ⌜ OUT *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
OUT *imm8*,AL	`OUT 0Fh,AL`	`; Přenos slabiky z AL do brány 0Fh`
OUT *imm8*,AX	`OUT 0Fh,AX`	`; Přenos slova z AX do bran 0Fh a 10h`
OUT DX,AL	`OUT DX,AL`	`; Přenos slabiky z AL do brány podle DX`
OUT DX,AX	`OUT DX,AX`	`; Přenos slova z AX do brány podle DX`

2.9.10 Přesuny dat

XCHG *Exchange Memory/Register with Register*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce XCHG vzájemně zamění obsahy zdrojového a cílového operandu. Výměna je možná mezi dvěma registry nebo mezi registrem a pamětí.

SYNTAX: ⌜ XCHG *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
XCHG *r*/*m*8,*r*8	`XCHG AL,AH`	`; Zamění obsahy AH a AL`
	`XCHG Slab,BH`	`; Zamění obsahy BH a Slab`
XCHG *r*8,*r*/*m*8	`XCHG CL,Slab`	`; Zamění obsahy Slab a CL`
XCHG *r*/*m*16,*r*16	`XCHG AX,BX`	`; Zamění obsahy BX a AX`
	`XCHG Slovo,BP`	`; Zamění obsahy BP a Slovo`
XCHG *r*16,*r*/*m*16	`XCHG DX,Slovo`	`; Zamění obsahy Slovo a DX`

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.

XLAT *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:BX uložena adresa tabulky a v registru AL index (neznaménkové číslo 0 až 255), který se sečte s obsahem registru BX. Slabikou z vypočtené adresy se naplní opět registr AL.

Instrukce se používá pro transformaci mezi kódy (např. pro převod mezi ASCII a EBCDIC).

SYNTAX: ⌜ XLAT

Instrukce	*Příklad*	*Komentář*
XLAT	`XLAT`	`; AL := DS:[BX+AL]`

Některé asemblery pro tuto instrukci vyžadují zadání operandu, který slouží pouze pro kontrolu typů. Operand musí být typu slabika. Instrukce XLATB bývá zpravidla instrukce neprovádějící kontrolu a nevyžadující operand.

LEA — *Load Effective Address*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce LEA ukládá do cílového operandu, kterým je jeden z všeobecných 16bitových registrů, offsetovou část adresy zdrojového operandu, na jehož místě je uložena adresa objektu v paměti. LEA je rozšířením instrukce MOV *r*16,*imm*16 (v asemblerech se tato instrukce zapisuje např. `MOV BX,OFFSET Slovo`) o možnost zadat adresu ve zdrojovém operandu indexovaně nebo bázovaně podle pravidel z části 2.8.

SYNTAX: ⌜ LEA *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
LEA *r16,m*	`LEA BX,Slab[BP][SI]`	`; Naplní BX offsetem` `;  adresy Slab[BP][SI]`

LDS — *Load Doubleword Pointer Using DS*
LES — *Load Doubleword Pointer Using ES*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce LDS a LES naplňují segmentový registr (DS a ES) a současně jeden z 16bitových všeobecných registrů 32bitovou adresou (segment a offset) uloženou v paměti v objektu specifikovaném zdrojovým operandem.

První 16bitové slovo 32bitové adresy v paměti (offset) je uloženo do registru, který je specifikován cílovým operandem, druhé slovo adresy (segment) je uloženo do segmentového registru určeného mnemonikou instrukce (tzn. DS pro LDS a ES pro LES).

SYNTAX: ⌜ LDS *cílový_operand,zdrojový_operand*
LES *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
LDS *r16,m16:16*	`LDS BX,DvojSlovo[SI]`	`; DS:BX := DvojSlovo[SI]`
LES *r16,m16:16*	`LES DI,DvojSlovo[4]`	`; ES:DI := DvojSlovo[4]`

2.9.11 Řetězcové instrukce

Instrukce ze skupiny řetězcových instrukcí pracují zpravidla se dvěma místy v paměti. Zdrojové místo v paměti bývá adresováno dvojicí DS:SI a cílové místo dvojicí ES:DI. Poněvadž řetězce jsou určeny adresami uloženými v těchto registrech, instrukce nepotřebují operandy. Řetězcové instrukce pracují buď se slabikami, nebo s 16bitovými slovy. Jsou-li v instrukci operandy, používá je asembler pro kontrolu typů a pro generování instrukce určené pro práci buď se slabikami, nebo se slovy. Instrukce s názvem končícím písmenem „B“ nevyžaduje operandy, protože předpokládá, že obsah dvojice DS:SI a ES:DI ukazuje na slabikovou strukturu. Instrukce, jejíž název končí „W“, pracuje s 16bitovými slovy. Třetí typ instrukcí (nekončící „B“ ani „W“) vyžaduje operandy. Překladač zkontroluje, jsou-li operandy stejného typu (oba slabika nebo slovo), a podle typu vygeneruje buď instrukci pracující se slabikami, nebo se slovy.

Po provedení instrukce se upravuje obsah SI a DI tak, aby ukazoval na další zpracovávaný prvek. Obsah SI a DI se zvyšuje nebo zmenšuje o velikost zpracovávaného prvku (o 1 pro slabiku a o 2 pro slovo). Zda se řetězce zpracovávají zepředu nebo zezadu, určuje momentální hodnota příznaku DF. Hodnota DF=0 způsobí zvětšování obsahu SI a DI po provedení operace a hodnota DF=1 zmenšování obsahu SI a DI 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. 68).

MOVS/MOVSB/MOVSW — *Move String*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce MOVS přesouvá obsah slabiky (slova) z adresy DS:SI do paměťového místa s adresou ES:DI. Instrukce může být uvedena buď bez operandů (MOVSB/MOVSW), 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) je generována odpovídající instrukce. 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 a 2 pro slovní operandy.

SYNTAX: ⌜ MOVS *cílový_operand,zdrojový_operand*
MOVSB
MOVSW

Instrukce	*Příklad*	*Komentář*
MOVSB	`MOVSB`	`; Slabiku z adresy DS:SI přepíše na adresu ES:DI`
MOVSW	`MOVSW`	`; Slovo z adresy DS:SI přepíše na adresu ES:DI`

CMPS/CMPSB/CMPSW — *Compare String Operands*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

Instrukce CMPS porovnává slabiku (slovo) cílového operandu (na adrese ES:DI) se zdrojovým operandem (na adrese DS:SI). Porovnání se provede tak, že se odečte cílový operand od zdrojového, tj. (obsah podle DS:SI) −

(obsah podle ES:DI). 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 SI a DI.

SYNTAX: ⌜ CMPS *zdrojový_operand,cílový_operand*
CMPSB
CMPSW

Instrukce	*Příklad*	*Komentář*
CMPSB	`CMPSB`	`; F:=(obsah podle DS:SI)-(obsah podle ES:DI)`
CMPSW	`CMPSW`	`; F:=(obsah podle DS:SI)-(obsah podle ES:DI)`

SCAS/SCASB/SCASW — *Compare String Data*

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

POPIS:

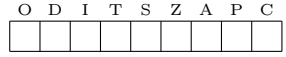
Instrukce SCAS porovnává prvek řetězce s obsahem registru AL nebo AX. Je-li prvkem řetězce slabika, pracuje se s registrem AL, je-li prvkem slovo, použije se registr AX. Porovnání se provede tak, že se od obsahu registru AL (AX) odečte obsah prvku řetězce na adrese ES:DI. Podle výsledku se nastaví příznakový registr, rozdíl se nikam neukládá. Po provedení porovnání se aktualizuje registr DI.

SYNTAX: ⌜ SCAS *cílový_operand*
SCASB
SCASW

Instrukce	*Příklad*	*Komentář*
SCASB	`SCASB`	`; F := AL - (obsah podle ES:DI)`
SCASW	`SCASW`	`; F := AX - (obsah podle ES:DI)`

LODS/LODSB/LODSW — *Load String Operand*

Popis:



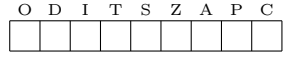
Instrukce LODS naplní registr AL nebo AX obsahem prvku řetězce uloženého na adrese DS:SI. Po provedení operace se aktualizuje registr SI.

Syntax: ⌜ LODS *zdrojový_operand*
LODSB
LODSW

Instrukce	*Příklad*	*Komentář*
LODSB	`LODSB`	`; AL := (obsah podle DS:SI)`
LODSW	`LODSW`	`; AX := (obsah podle DS:SI)`

STOS/STOSB/STOSW — *Store String Data*

Popis:



Instrukce STOS uloží obsah registru AL nebo AX do prvku řetězce na adrese ES:DI. Po provedení operace se aktualizuje registr DI.

Syntax: ⌜ STOS *cílový_operand*
STOSB
STOSW

Instrukce	*Příklad*	*Komentář*
STOSB	`STOSB`	`; Podle ES:DI := AL`
STOSW	`STOSW`	`; Podle ES:DI := AX`

REP/REP*cond* — *Repeat Following String Operation*

O	D	I	T	S	Z	A	P	C
					*			

Popis:

Instrukční prefix REP lze použít s takovou instrukcí ze skupiny řetězcových operací, která nenastavuje příznaky. Prefix způsobí opakované provádění instrukce. Při každém opakování se zmenší obsah registru CX o jedničku. Prefix instrukci opakuje tak dlouho, dokud je CX nenulové. Podmíněné varianty prefixu REP*cond* testují navíc ještě příznak ZF a má význam je používat s instrukcemi CMPS nebo SCAS, které nastavují příznaky.

Přesný algoritmus činnosti prefixů REP je následující:

1. Testování CX. Je-li CX=0, provede se skok na další instrukci (opakování se již neprovádí).

2. Test, nežádá-li někdo o přerušení. Pokud ano, přerušení se uplatní.

3. Jedno provedení požadované instrukce.

4. Snížení CX o jedničku. Toto odečtení nemění nastavení registru příznaků a nemá na něj vliv ani hodnota příznaku DF.

5. Je-li prefix nepodmíněný (REP), pokračuje se v algoritmu bodem 1. Je-li prefix podmíněný, testuje se nastavení příznaku ZF. Je-li prefix REPE nebo REPZ a ZF=0 (poslední porovnávání detekovalo nerovnost), provede se skok na další instrukci (opakování se již neprovádí). Je-li prefix REPNE nebo REPNZ a ZF=1 (poslední porovnávání detekovalo rovnost), provede se skok na další instrukci. V opačných případech se v algoritmu pokračuje bodem 1.

Příklad:

```
LDS  SI,adresa_zdrojového_řetězce
LES  DI,adresa_cílového_řetězce
MOV  CX,počet_prvků_řetězce
REP  MOVSW   ; Zkopíruje prvky řetězce typu slovo.
```

Příklad:

```
LDS  SI,adresa_zdrojového_řetězce
LES  DI,adresa_cílového_řetězce
MOV  CX,počet_prvků_řetězce
REPE CMPSB  ; Srovnává obsahy dvou řetězců typu slabika.
JE   Shodne ; Řetězce mají shodný obsah.
; Řetězce se liší.
```

REP a REP*cond* jsou pouze symbolické zápisy skutečných instrukčních prefixů, které bývají uvedeny před operačním kódem instrukce. Operační kódy diskutovaných prefixů jsou: F3h (REP, REPE) a F2h (REPNE).

2.9.12 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. Proto nebudeme uvádět ani zápis syntaxe ani příklady.

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.

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*

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	?

POPIS:

Instrukce AAD převádí dvě BCD číslice v nezhuštěném tvaru uložené v registru AX (v AH je číslice vyššího řádu) na binární číslo do registru AX.

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.

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.

Obsah registru AX dělí číslem 10, celá část výsledku se uloží do registru AH a zbytek po dělení do registru AL.

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.

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*

O	D	I	T	S	Z	A	P	C
?				*	*	*	*	*

Popis:

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.

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.

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.

2.9.13 Řídicí instrukce

NOP — *No Operation*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce NOP neprovádí žádnou akci. Má délku 1 slabiky nesoucí operační kód 90h (stejný operační kód má instrukce XCHG AX,AX). Hojně se užívá tam, kde je potřeba doplnit slabiku na zarovnání adresy následující instrukce na hodnotu dělitelnou 2.

SYNTAX: ⌜ NOP

HLT — *Halt CPU*

O	D	I	T	S	Z	A	P	C

POPIS:

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.

SYNTAX: ⌜ HLT

ESC *Escape*

O	D	I	T	S	Z	A	P	C

POPIS:

Pomocí ESC procesor rozpoznává, že jde o příkaz určený externímu koprocesoru (např. 8087).

Příkazy pro koprocesor jsou součástí instrukcí pro procesor 8086. Liší se pouze tím, že jsou uvedeny vzorkem ESC. Jeho výskyt procesoru 8086 oznamuje, že tuto instrukci nemá zpracovat, ale že ji má předat koprocesoru včetně operandů. Činnost obou procesorů je možno synchronizovat pomocí instrukce WAIT (viz dále).

ESC není součásti asemblerů, protože jde jenom o posloupnost pěti bitů (11011) zařazených před příkaz koprocesoru. Příkazy koprocesoru 8087 zpravidla překladače zpracovat umějí.

WAIT *Wait until BUSY Pin Is Inactive*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce WAIT uvede procesor do čekacího stavu. Instrukce je určena k čekání na dokončení operace koprocesorem (např. 8087), který svoji nečinnost oznamuje signálem $\overline{\text{TEST}}$ (ve vyšších typech procesorů je tento signál nazýván $\overline{\text{BUSY}}$). Činnost procesoru je pozastavena, dokud je signál $\overline{\text{TEST}}$ aktivní. Během čekání se může uplatnit povolené přerušení.

SYNTAX: ⌜ WAIT

LOCK — *Assert LOCK Signal Prefix*

O	D	I	T	S	Z	A	P	C

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. Prefix LOCK se smí použít pouze u těchto instrukcí: ADD, OR, ADC, SBB, AND, SUB, XOR, NOT, NEG, INC, DEC, mají-li jeden z operandů uložen v paměti. Automaticky se signál $\overline{\text{LOCK}}$ generuje při provádění instrukce XCHG a INT. Použití tohoto prefixu s jinými instrukcemi způsobí u vyšších typů procesorů přerušení s chybou „nedefinovaný operační kód“.

Syntax: ⌜ LOCK instrukce

Instrukce	*Příklad*	*Komentář*
LOCK	`LOCK AND Slovo,00FFh`	`; Nedělitelně Slovo := Slovo ∧ 00FFh`

V tomto příkladu LOCK zajistí, aby mezi okamžikem čtení cílového operandu z paměti a okamžikem zápisu cílového operandu do paměti jiný procesor s tímto operandem nepracoval.

3 Intel 80286

Procesor 80286 je následníkem 8086. Jde o 16bitový procesor s pokročilejší architekturou podporující práci ve dvou režimech. V *reálném režimu* je slučitelný s 8086 a v *chráněném režimu* poskytuje vlastnosti směřující k víceúlohovému zpracování. Na čipu je s procesorem integrována také jednotka správy paměti, která v *chráněném režimu* dovoluje adresovat 16 MB reálné paměti a až 1 GB virtuální paměti. Jednotka rovněž poskytuje prostředky pro 4úrovňovou ochranu částí paměti proti neoprávněným přístupům.

V *reálném režimu* lze v procesoru spouštět programy určené pro 8086 bez jejich modifikace. Pro zpracovávání v *chráněném režimu* se vyžaduje rekompilace, případně úprava programů.

K procesoru je přivedena 16bitová datová sběrnice (D_0 až D_{15}) a 24bitová adresová sběrnice (A_0 až A_{23}). Procesor je integrován do čtvercového integrovaného obvodu s 68 vývody. Vyžaduje jediné napájecí napětí +5 V.

3.1 Architektura 80286

Procesor 80286 je složen ze 4 nezávislých paralelně pracujících jednotek a tím umožňuje částečné proudové zpracování instrukcí.

Sběrnicová jednotka (Bus Unit) zajišťuje přístup k reálné paměti a vybírá z ní slabiky nezávisle na činnosti ostatních jednotek. Vybrané operační kódy a data ukládá do 6 slabik dlouhé fronty, odkud je vybírá instrukční jednotka.

Instrukční jednotka (Instruction Unit) dekóduje vybrané instrukce a ukládá maximálně 3 do fronty pro prováděcí jednotku.

Prováděcí jednotka (Execution Unit) realizuje dekódované instrukce. Pro přístup k paměti a V/V operace používá sběrnicovou jednotku.

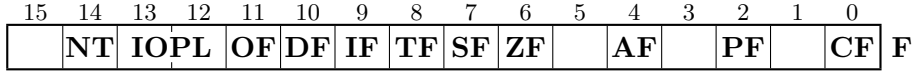
Adresovací jednotka (Address Unit) je správcem paměti. Zajišťuje ochranu částí paměti před neoprávněnými přístupy a přepočítává virtuální

adresy na reálné.

Výhodou takové paralelní struktury je možnost souběžně provádět instrukci i, dekódovat instrukci $i + 1$ a z paměti vybírat instrukci $i + 2$.

3.2 Registry procesoru 80286

Registrová struktura procesoru 80286 je stejná jako procesoru 8086 (viz obr. 2.3 na str. 18). V příznakovém registru **F** (Flags) jsou využity navíc 3 bity (viz obr. 3.1) **NT** a **IOPL**, které se uplatní pouze v *chráněném režimu*.



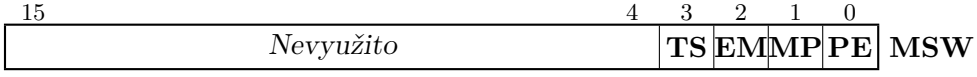
Obr. 3.1. Příznakový registr procesoru 80286

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.

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

Ostatní příznaky mají stejný význam jako u procesoru 8086 (viz str. 19).

Registrová struktura 80286 je rozšířena o jeden 16bitový registr **MSW** (Machine Status Word), ve kterém mají význam pouze dolní 4 bity (viz obr. 3.2). Ostatní jsou nevyužity.



Obr. 3.2. Registr MSW procesoru 80286

PE (Protected Mode Enable) zapíná *chráněný režim* procesoru. Po inicializaci procesoru (signálem RESET) je zapnut *reálný režim*. Nastavením tohoto příznaku se procesor přepne do *chráněného režimu*. Zpět do *reálného režimu* lze procesor vrátit pouze inicializací procesoru (RESET).

MP (Monitor Processor Extension) indikuje fyzickou přítomnost koprocesoru (např. matematického koprocesoru 80287).

EM (Emulate Processor Extension) zapíná programovou emulaci koprocesoru tehdy, není-li koprocesor instalován (viz INT 7 na str. 107).

TS (Task Switch) se nastavuje vždy přepnutím procesu. Je používán koprocesorem ke zjištění, že v procesoru se vyměnil „zadavatel“ úkolů.

3.3 Adresace paměti v chráněném režimu 80286

V *reálném režimu* 80286 je chování procesoru analogické s 8086. Filozofie adresování paměti v *chráněném režimu* je jiná. Paměť je rozdělena na tzv. **segmenty**. Segment je souvislý prostor paměti velikosti až 64 KB, který může začínat na libovolné adrese. Rozdělení paměti na segmenty podporuje modulární strukturu programů dělených na vlastní kód, data a zásobník. Každý segment je definován těmito parametry:

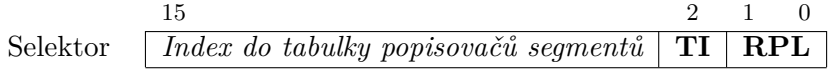
1. bází segmentu (adresou začátku segmentu),
2. limitem segmentu (délkou segmentu ve slabikách – 1),
3. přístupovými právy a typem segmentu.

V rámci jednoho segmentu se pohybujeme pomocí zadání 16bitového offsetu, který přičtením k bázi určí reálnou 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.

3.3.1 Virtuální adresa

Adresa v *chráněném režimu*, stejně jako v *reálném*, je složena ze dvou 16bitových složek. Interpretace složky nazývané **offset** je stejná. Použití druhé šestnáctice bitů je v *chráněném režimu* jiné. Tuto složku nazvěme **selektor segmentu** (Segment Selector). Kompletní adresu v *chráněném režimu* (selektor a offset) nazýváme **virtuální adresa**.



Obr. 3.3. Struktura selektoru segmentu procesoru 80286

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:

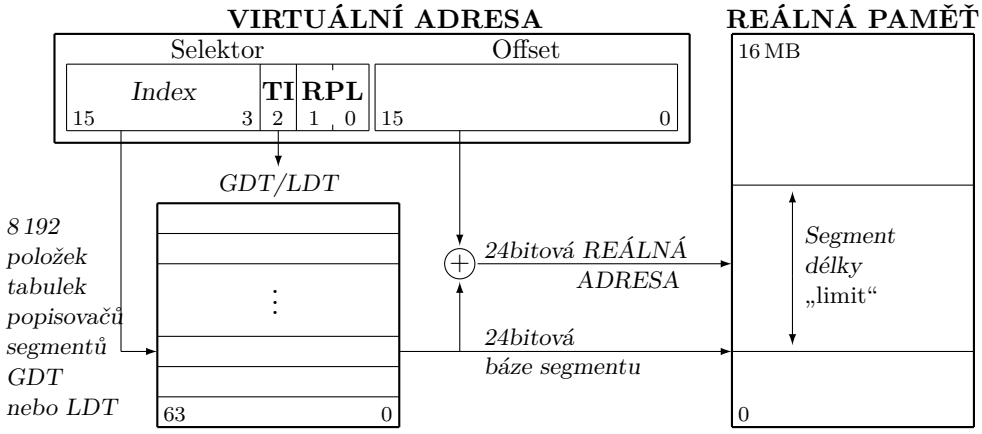
RPL (Requested Privilege Level) představuje úroveň oprávnění, kterou proces nabízí při přístupu k tomuto segmentu.

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. 88).

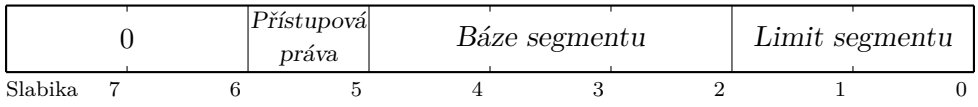
3.3.2 Tabulky popisovačů segmentů

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 virtuální adresy nazvaná **selektor** ukazuje do tabulky popisovačů segmentů buď globálního adresového prostoru (TI=0), nebo lokálního adresového prostoru (TI=1). Všechny tyto tabulky jsou rezidentně uloženy v paměti a jejich prostřednictvím se transformují 32bitové virtuální adresy na 24bitové reálné adresy (viz obr. 3.4).



Obr. 3.4. Transformace virtuální adresy na reálnou pomocí tabulek popisovačů segmentů v procesoru 80286

Tabulky popisovačů segmentů mají 8 192 položek. Každá položka má délku 8 slabik, které obsahují následující informace: **bázi segmentu** (adresu začátku segmentu – 24 bitů), **limit segmentu** (nejvyšší adresu uvnitř segmentu – 16 bitů), **přístupová práva** (8 bitů) a dvě slabiky jsou rezervovány pro použití v procesoru 80386 (z důvodu kompatibility programů s vyššími typy procesorů musí být v 80286 tyto slabiky vynulovány). Rozmístění jednotlivých informací v položce je patrné z obr. 3.5.



Obr. 3.5. Položka tabulky popisovačů segmentů

Typ popisovaného segmentu je definován obsahem slabiky **přístupová práva**. Podle typu segmentu rozlišujeme v 80286 tyto čtyři základní třídy popisovačů:

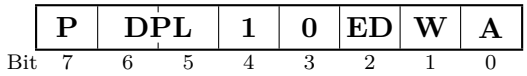
1. popisovač segmentu obsahujícího data (datový segment),
2. popisovač segmentu obsahujícího instrukce (instrukční segment),

3. popisovač segmentu obsahujícího informace pro systém (systémový segment),

4. popisovač brány (bude vysvětleno v odstavci „Brány“ na str. 89).

3.3.3 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. 3.6).



Obr. 3.6. Přístupová práva popisovače datového segmentu

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

P (Segment Present) je nastaven na jedničku tehdy, je-li obsah popisovače platný a segment je uložen v reálné paměti. Je-li nulový, je obsah popisovače neplatný.

Pokus o přístup k segmentu, který má nulovou hodnotu bitu P v popisovači, vyvolá přerušení INT 11 (Segment not Present) nebo INT 12 (Stack Exception) v závislosti na typu uložených dat.

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

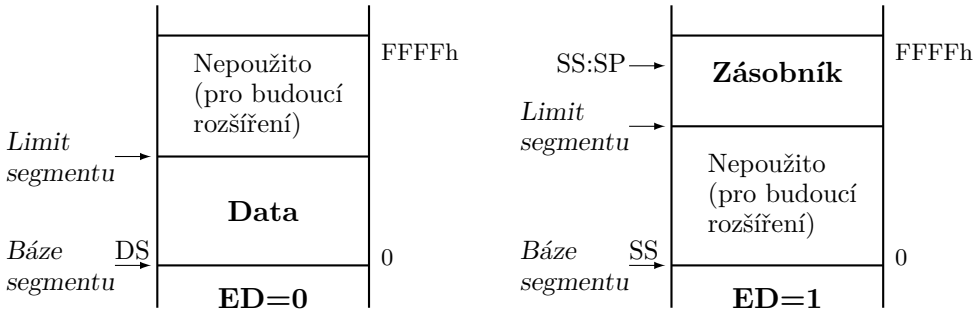
ED (Expansion Direction) indikuje, kterým směrem se bude obsah segmentu rozšiřovat. Datové segmenty mohou obsahovat klasická data nebo zásobníky. Je pravděpodobné, že při požadavku na zvětšení klasických dat se bude obsah rozšiřovat směrem k vyšším adresám. U zásobníku tomu bude naopak, protože ten se plní od vyšších adres k nižším (viz str. 21).

Je-li nastaveno ED=0 (data), bude se obsah segmentu rozšiřovat směrem k vyšším adresám. Data se ukládají (v rámci 64 KB segmentu)

od adresy 0000 směrem k adrese 0FFFFh. Při požadavku na zvětšení obsahu se musí zvětšit hodnota limitu segmentu.

Je-li nastaveno ED=1 (zásobník), bude se obsah segmentu rozšiřovat směrem k nižším adresám. Položky zásobníku se ukládají od adresy 0FFFFh směrem k adrese 0000 (uvnitř 64 KB segmentu). Při požadavku na zvětšení obsahu se musí zmenšit hodnota limitu segmentu (ten se totiž stále počítá od adresy 0). Pro srovnání viz obr. 3.7.

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.



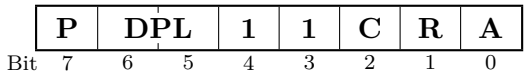
Obr. 3.7. Srovnání normálního datového a zásobníkového segmentu

W (Writable) je nastaven na 1, pokud je povoleno čtení i zápis do segmentu. 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 použití instrukce testující selektor). Procesor tento příznak nenuluje. Je určen operačnímu systému ke sledování četnosti přístupů ke konkrétním segmentům.

3.3.4 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. 3.8).



Obr. 3.8. 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.

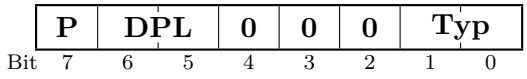
C (Conforming) nulový sděluje, že podprogramy volané v tomto segmentu budou mít nastavenu úroveň oprávnění odpovídající úrovni segmentu, v němž se nacházejí. Je-li C=1, bude volanému podprogramu v tomto segmentu přidělena úroveň oprávnění segmentu, z něhož je volán.

R (Readable) nulový zakazuje čtení obsahu segmentu. Je povoleno pouze obsah segmentu spustit. Jedničková hodnota bitu povoluje jak spuštění, tak i čtení segmentu.

A (Accessed) viz popisovač datového segmentu.

3.3.5 Popisovač systémového segmentu

Systémový segment obsahuje informace určené procesoru a operačnímu systému. Popisovač se od předchozích typů opět liší pouze slabikou přístupová práva (viz obr. 3.9). Tento popisovač smí být umístěn **pouze v GDT**.



Obr. 3.9. Přístupová práva popisovače systémového segmentu

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

Typ=1 označuje segment stavu procesu (TSS – Task State Segment) pro právě neaktivní proces.

Typ=3 označuje segment stavu procesu pro právě aktivní proces.

Typ=2 označuje segment lokální tabulky popisovačů segmentů (LDT – Local Descriptor Table).

Do systémového segmentu smí zapisovat pouze procesor. Programátor (operační systém) si pro účely modifikace obsahu musí přes systémový segment překrýt datový (viz odstavec 3.3.8).

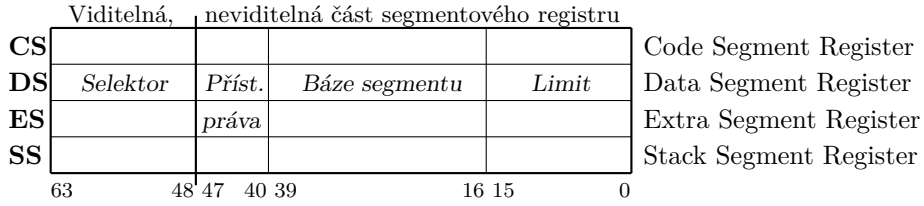
Každá tabulka popisovačů segmentů je sama uložena v některém ze segmentů. Tabulek LDT může být v jednom okamžiku v paměti více. Tabulka popisovačů globálního adresového prostoru (GDT – Global Descriptor Table) je právě jedna. Pro uchování adresy jejího uložení je vyhrazen speciální registr **GDTR** (Global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

3.3.6 Segmentové registry

Segmentové registry CS, DS, ES a SS jsou v chráněném režimu procesoru 80286 složeny ze dvou částí: 16bitového selektoru (část programátorovi „viditelná“) a 48bitové programátorovi neviditelné části obsahující 1 slabiku přístupových práv, 3 slabiky báze segmentu a 2 slabiky limitu segmentu (viz obr. 3.10).

Obsah selektoru registrů SS, DS a ES 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.

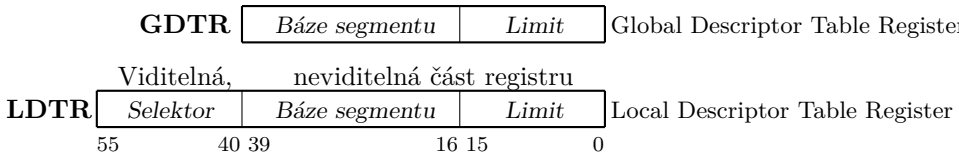


Obr. 3.10. Struktura segmentových registrů v chráněném režimu procesoru 80286

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 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.

3.3.7 Registry GDTR a LDTR

Registr **GDTR** (Global Descriptor Table Register) uchovává adresu uložení tabulky popisovačů globálního adresového prostoru. GDTR obsahuje 3 slabiky báze segmentu s uloženou tabulkou a 2 slabiky limitu tohoto segmentu. Registr je přístupný pouze instrukcemi LGDT (Load GDT) a SGDT (Store GDT).

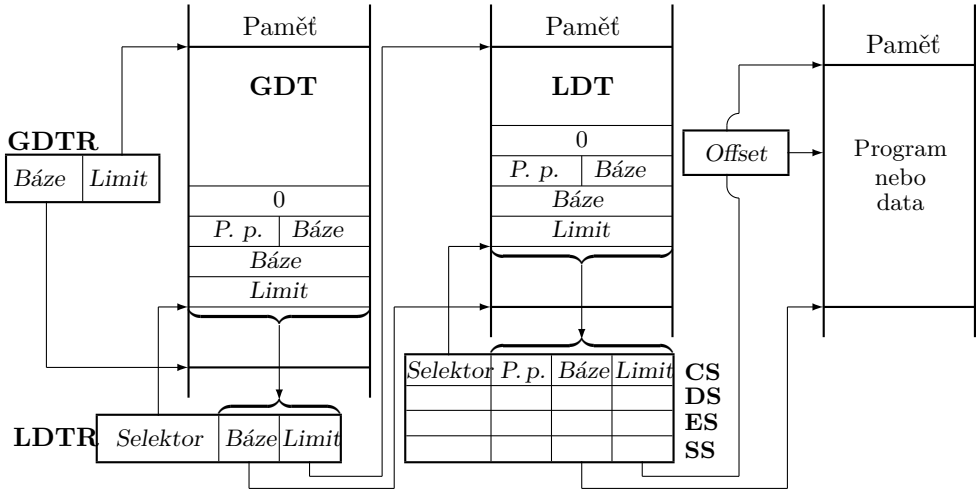


Obr. 3.11. Struktura registrů GDTR a LDTR

Registr **LDTR** (Local Descriptor Table Register) je složený ze dvou částí: z viditelného 16bitového selektoru a z neviditelné 40bitové části obsahující

bázi segmentu s LDT a limit segmentu s LDT. Selektor LDTR (viditelná část) má stejný tvar jako ve virtuální 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. 3.12.



Obr. 3.12. Použití GDTR, LDTR a segmentových registrů

3.3.8 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 reálné 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.

3.4 Systém ochran 80286

Systém ochran v chráněném režimu procesoru 80286 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.).

3.4.1 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 80286 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 (Requested 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.

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

3.4.2 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 chybové přerušení. Procesor bude toto přerušení 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.

3.4.3 Předání řízení do instrukčního segmentu

Proces smí předat řízení pouze do segmentu se stejnou úrovní oprávnění, musí tedy platit **CPL = DPL**. Bez použití dalších prostředků (volání podprogramů na jiné úrovni oprávnění pomocí brány) nelze předat řízení do segmentu s jinou úrovní oprávnění. Předávání řízení se uskutečňuje instrukcemi CALL, JMP a RETF.

3.4.4 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 v procesoru 80286:

1. uvnitř jednoho segmentu (krátký a blízký JMP, blízké CALL a RET),
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.

3.4.5 Brány

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

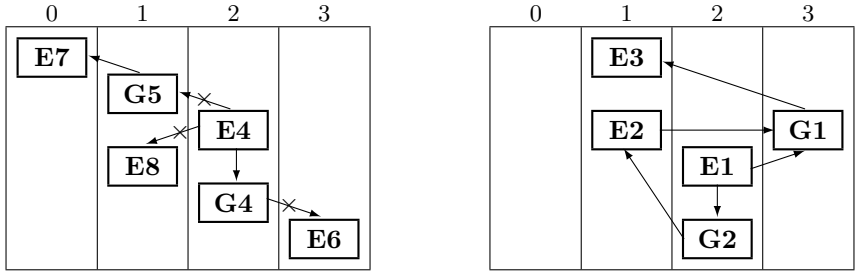
1. brána pro předání řízení do segmentu vyšší úrovně oprávnění (Call Gate),
2. brána pro nemaskující přerušení (Trap Gate),
3. brána pro maskující přerušení (Interrupt Gate) – obě popsány na str. 104,
4. brána zpřístupňující segment stavu procesu (Task Gate) – popsána na str. 98.

3.4.6 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**

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

brány. Ovšem (podle odstavce 3.4.4) musí platit, že úroveň oprávnění volaného podprogramu musí být stejná nebo vyšší než volajícího, tj. **CPL ≥ DPL podprogramu**.



Obr. 3.13. Příklad předávání řízení pomocí bran

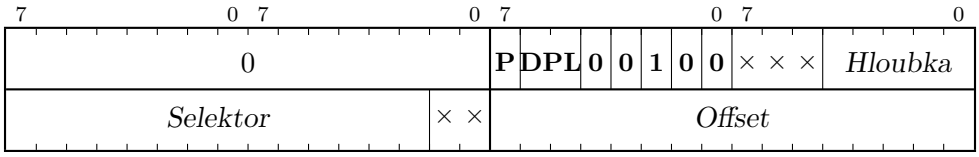
Na obr. 3.13 je několik příkladů volání bran a instrukčních segmentů prostřednictvím těchto bran. Objekt označený „**G**“ představuje bránu a „**E**“ instrukční segment. Čísla v horní řadě označují oblasti jednotlivých úrovní oprávnění. V tomto odstavci byla vysvětlena správnost volání bran E1-G2, E1-G1, E2-G1 a E4-G4. Rovněž bylo vysvětleno, proč je volání E4-G5 nesprávné. V odstavci 3.4.4 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.

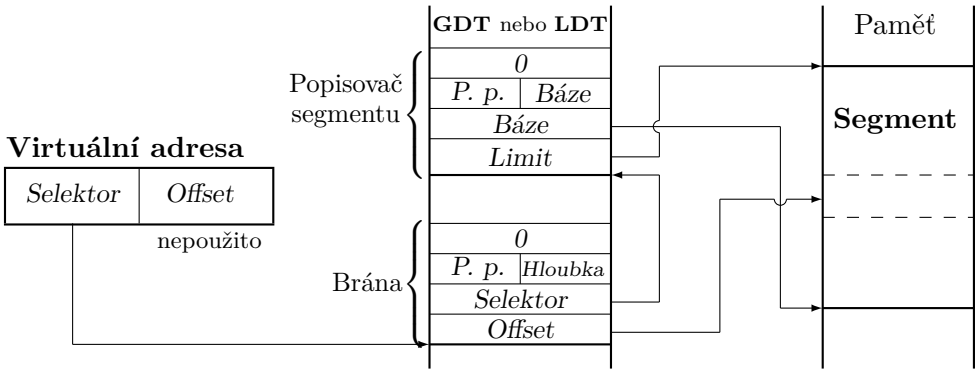
Má-li instrukční segment, v němž je volaný podprogram, v popisovači nastaven bit C (Conforming) na jedničku, bude volaný podprogram prováděn s úrovní oprávnění volajícího segmentu.

Při volání brány (rozuměj popisovače brány v tabulce popisovačů LDT nebo GDT) se z virtuální adresy uplatní pouze část selektor. Offset je ignorován. Z popisovače brány (viz obr. 3.14, na kterém bity 0 a 1 slabiky



Obr. 3.14. Formát popisovače brány pro předání řízení v GDT nebo LDT

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. 3.15).



Obr. 3.15. Použití brány k předávání řízení instrukčnímu segmentu

Pro zajištění vzájemné izolace procesů nabízí chráněný režim 80286 všem úrovním oprávnění každého procesu vlastní **zásobníky**. SS:SP 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:

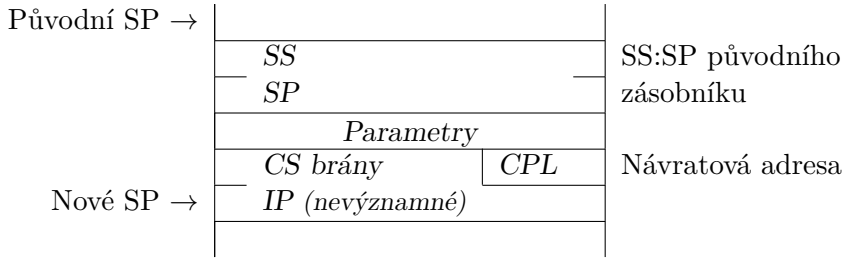
```
PUSH        Par1
PUSH        Par2
CALL        Podprogram
```

Brána pro předávání řízení zkopíruje parametrem **hloubka** (WC – Word Count) (viz obr. 3.14) zadaný počet 16bitových slov ze zásobníku úrovně oprávnění volajícího modulu do zásobníku úrovně oprávnění volaného modulu. Maximální hodnota parametru *hloubka* je 31. Detailně popíšeme činnost brány při předávání řízení třemi kroky:

1. Ukazatel vrcholu zásobníku (SS:SP) volajícího modulu (starý zásobník) se uloží do zásobníku volaného podprogramu (nový zásobník). SS:SP se naplní obsahem ukazujícím do zásobníku úrovně oprávnění odpovídající volanému podprogramu.

2. Ze starého zásobníku se zkopíruje *hloubka* slov do nového zásobníku.

3. Do nového zásobníku se vloží jako návratová adresa (CS:IP) adresa této brány. Tím může zásobník být použit volaným podprogramem.



Obr. 3.16. Použití zásobníku při předávání parametrů bránou

Počáteční hodnota SS:SP 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 nasta-

ne, 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.

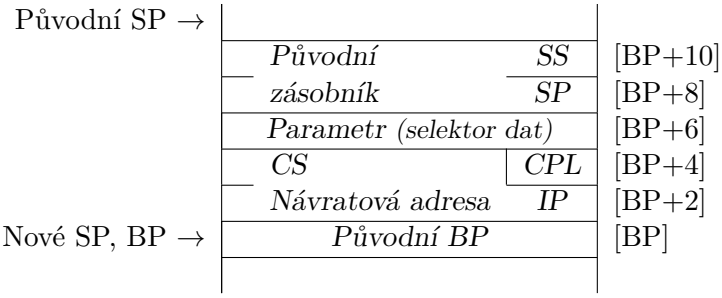
Bity 0 a 1 každého selektoru specifikují RPL (jde-li o selektor z registru CS, hovoříme o CPL). Přístup k datovému segmentu procesor povolí tehdy, platí-li: CPL≤DPL a zároveň RPL≤DPL. Při použití EPL můžeme zapsat pouze EPL≤DPL.

Hodnoty EPL vypočítává instrukce ARPL (Adjust RPL), která testuje hodnoty bitů 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.

Použití ARPL si ukážeme na příkladu podprogramu volaného prostřednictvím brány. Podprogram má jeden vstupní parametr – selektor pro přístup k datovému segmentu:

```
APP   PROC FAR        ; Deklarace formální hlavičky podprogramu.
      PUSH BP         ; Uchování obsahu BP volajícího modulu.
      MOV  BP,SP      ; Naplnění BP současnou hodnotou SP.
      MOV  AX,[BP+6]  ; Naplnění AX obsahem parametru (selektor dat).
      MOV  BX,[BP+4]  ; Naplnění BX selektorem CS volajícího modulu.
      ARPL AX,BX      ; RPL v AX přiřadí Max(AX,BX) = vypočtené EPL.
      MOV  DS,AX      ; Naplnění DS a kontrola oprávněnosti.
       ⋮
      POP  BP         ; Obnovení obsahu BP volajícího modulu.
      RET  2          ; Odstranění jednoho parametru ze zásobníku.
APP   ENDP            ; Formální konec podprogramu.
```

Na obr. 3.17 je zachycen obsah zásobníku v tomto příkladu po provedení instrukce `MOV BP,SP`. Podprogram do registru DS zapisuje hodnotu selektoru podle parametru s upravenou hodnotou RPL. Do RPL byla přiřazena EPL procesu v okamžiku přístupu k datovému segmentu.



Obr. 3.17. Stav zásobníku po provedení instrukce `MOV BP,SP` v příkladu na straně 94

3.4.7 Privilegované instrukce

Procesor 80286 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 V/V instrukce, které lze provádět pouze od jisté úrovně oprávnění.
Pozn.: Originální mnemonika 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 vnitřní přerušení INT 13. Do této skupiny patří instrukce modifikující obsah registrů: GDTR, LDTR, TR, IDTR a MSW. 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),
LMSW	naplnění registru MSW,
CLTS	nulování bitu TS (Task Switch) v registru MSW,
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, INSB, INSW	čtení ze V/V brány,
OUT, OUTS, OUTSB, OUTSW	zápis na V/V bránu,
STI, CLI	změna příznaku IF,
prefix LOCK	blokování sběrnice.

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 FLAGS (např. POPF). Po provedení takové instrukce zůstává vždy příznak IF nezměněn a není generováno žádné přerušení.

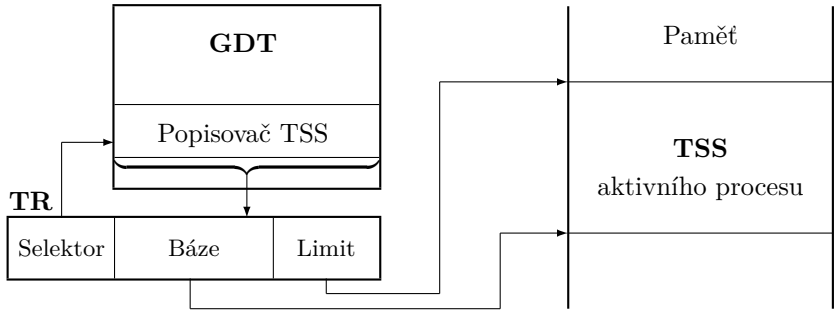
3.5 Přepínání procesů

Jedním z hlavních rysů chráněného režimu procesoru 80286 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.

3.5.1 Segment stavu procesu

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. 3.18. Na každý TSS smí ukazovat právě jeden popisovač TSS (TSS není reentrantní).



Obr. 3.18. 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 24bitovou bázi segmentu a 16bitový limit ukazující na skutečný TSS.

Tvar popisovače systémového segmentu ukazujícího na TSS je na obr. 3.5 (na straně 79) a slabika *přístupová práva* je popsána na obr. 3.9 (na straně 82). Položka **Typ=3** sděluje, že jde o TSS právě aktivního procesu, a **Typ=1** určuje, že jde o TSS právě neaktivního procesu.

Každý TSS je dlouhý alespoň 22 16bitových slov. Informaci uloženou v TSS rozdělujeme do čtyř polí:

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

1. Zpětný ukazatel. Zde je uložen selektor TSS přerušeného procesu. Ukazatel se použije při obnovení přerušeného procesu instrukcí IRET.

2. Ukazatele zásobníků. Toto pole obsahuje momentální stavy ukazatelů zásobníků pro úrovně oprávnění 0, 1 a 2.

3. Registry procesoru. Tady jsou uloženy obsahy všech registrů, které procesor programátorovi poskytuje.

4. Selektor LDT. Toto pole obsahuje selektor položky GDT specifikující LDT náležející procesu.

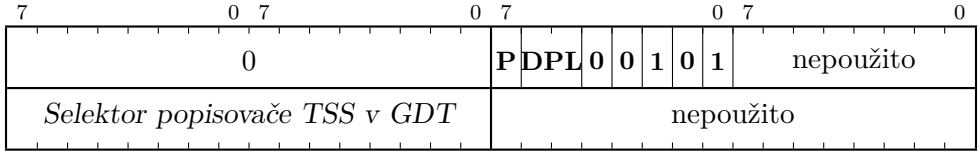
Obr. 3.19. Tvar segmentu stavu procesu (TSS)

Limit TSS není pevně stanoven na hodnotu 43, může být větší. Volnou část lze použít např. operačním systémem pro uložení doplňujících informací o procesu.

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 **CPL≤DPL**, tj. lze přepnout na proces, který je na stejné nebo nižší úrovni oprávnění.

3.5.2 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. 3.20.



Obr. 3.20. 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.

3.5.3 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:

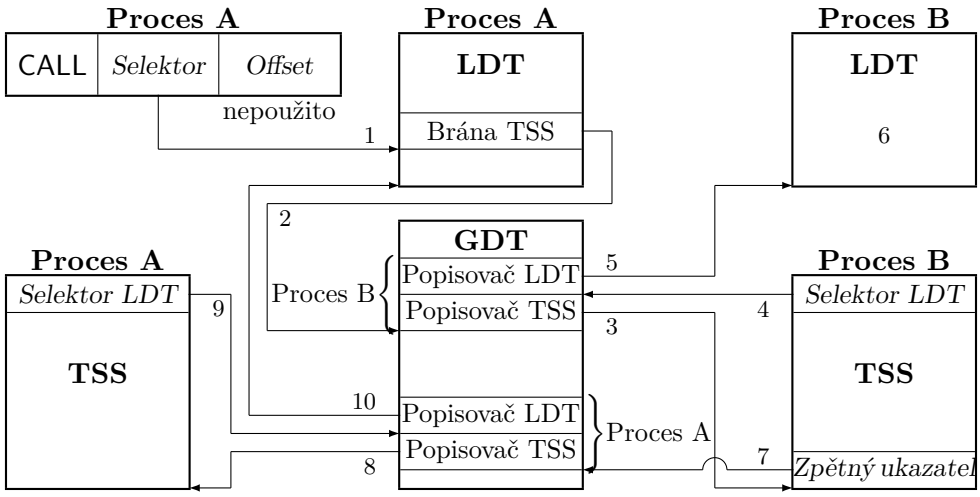
vzdáleným JMP nebo CALL, jehož selektor ukazuje na popisovač TSS nového procesu v GDT. Offsetová část adresy se ignoruje.

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. 3.21 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. 3.21. 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.

3.5.4 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 MSW).
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 MSW).

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.

3.5.5 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:IP a F.
- 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 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.

3.6 Přerušení

Přerušení je v chráněném režimu obsluhováno diametrálně odlišně než v reálném režimu. Zásadní změnou je neexistence tabulky přerušovacích vektorů a zavedení nové struktury podobající se GDT.

3.6.1 Tabulka popisovačů segmentů obsluhy přerušení

Tabulka popisovačů segmentů obsluhy přerušení **IDT** (Interrupt Descriptor Table) obsahuje až 256 ukazatelů na rutiny obsluhující přerušení.

Adresa IDT je uložena ve speciálním registru **IDTR** (Interrupt Descriptor Table Register). Registr obsahuje 24bitovou bázi a 16bitový limit IDT. Plní se privilegovanou instrukcí LIDT (lze ji provést pouze s úrovní oprávnění 0). V IDT se vyskytují pouze tyto tři typy popisovačů:

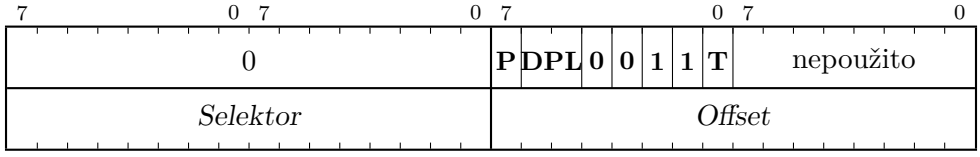
- brána zpřístupňující TSS (popsána na str. 98),
- brána pro maskující přerušení (Interrupt Gate),
- brána pro nemaskující přerušení (Trap Gate).

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.

3.6.2 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.



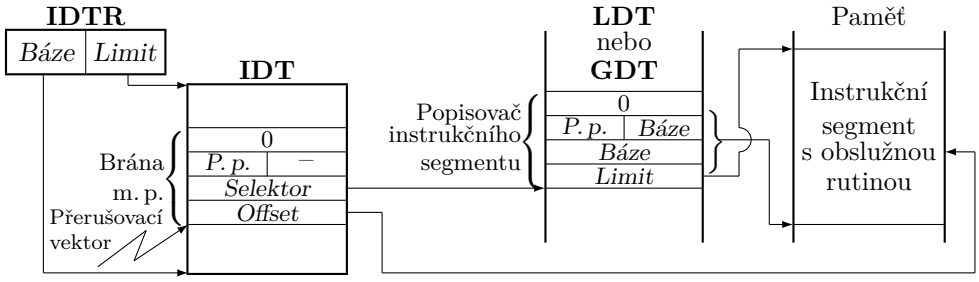
Obr. 3.22. Formát popisovače brány přerušení v IDT

Formát brány pro maskující i nemaskující přerušení je na obr. 3.22. 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. 3.23).

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.

Z hlediska informací ukládaných do zásobníku rozlišujeme tři typy přerušení (viz též obr. 3.24):

1. Vnější přerušení a většina vnitřních přerušení nepředávají žádné chybové hlášení a do zásobníku ukládají F, CS, IP tak, jak to bylo popsáno

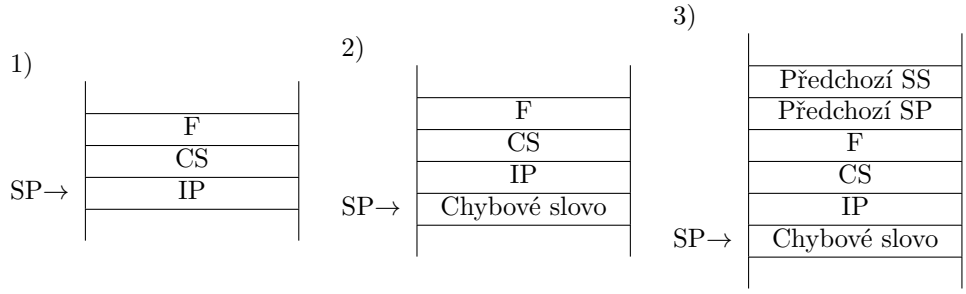


Obr. 3.23. Obsluha přerušení bránou v IDT

u procesoru 8086 na str. 22.

2. Předává-li přerušení obslužné rutině kód chyby a rutina pracuje na stejné úrovni oprávnění, jakou má přerušení, uloží se do zásobníku pouze **chybové slovo**.

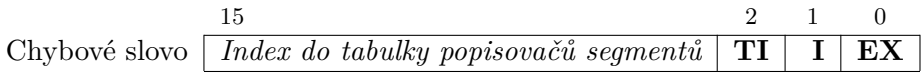
3. Předává-li přerušení obslužné rutině kód chyby a rutina pracuje na jiné úrovni oprávnění, musí se také uložit obsah SS:SP původní úrovně oprávnění, protože každá úroveň oprávnění má vlastní zásobník.



Obr. 3.24. Možné obsahy zásobníku při aktivaci obsluhy přerušení

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. 3.25).

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



Obr. 3.25. Formát chybového slova předávaného přerušeními 10 až 13

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

3.6.3 Rezervovaná přerušení

Procesor 80286 generuje přerušení, která jsou popsána v tabulce na obr. 3.26. Přerušení INT 0 až 4 jsou shodná s 8086 (viz str. 24).

Přerušení generovaná procesorem 80286 dělíme do tří kategorií:

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

Trap do zásobníku uloží CS:IP ukazující **za instrukci** (na následující instrukci), která přerušení způsobila,

Abort způsobí, že v procesu nelze pokračovat a musí být násilně ukončen.

INT 5 **Kontrola mezí** (Bound Check)

Přerušení vyvolá instrukce BOUND, je-li její operand mimo zadané meze (viz popis instrukce BOUND na str. 117). Instrukce IRET umístěná v přerušovací rutině způsobí opakování BOUND, protože CS:IP uložené do zásobníku ukazuje na instrukci, která přerušení způsobila.

INT 6 **Chybný operační kód** (Invalid Opcode)

Toto přerušení procesor generuje, nedokázal-li rozpoznat operační kód instrukce nebo nalezl-li chybnou kombinaci operačního kódu a operandu. CS:IP uložené do zásobníku ukazuje na instrukci, která přerušení způsobila.

Číslo vektoru	*Určení vektoru*	*Typ přerušení*	*Chybové slovo?*
0	Dělení nulou	Fault	ne
1	Krokovací režim	Trap	ne
2	Nemaskovatelná přerušení	–	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)
9	Překročení segmentu koprocesorem	Abort	ne
10	Chybný TSS	Fault	ano
11	Výpadek segmentu	Fault	ano
12	Výpadek segmentu se zásobníkem	Fault	ano
13	Obecná chyba ochrany	Fault	ano
16	Chyba koprocesoru	Fault	ne

Obr. 3.26. Přerušení generovaná procesorem 80286

INT 7 **Nedostupnost koprocesoru** (Processor Extension Not Available)

Přerušení mohou vyvolat instrukce obsluhující koprocesor v závislosti na indikátorech zaznamenaných v registru MSW (viz obr. 3.2 na str. 76):

- 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 určené koprocesoru způsobí přerušení INT 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)

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

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

INT 9 **Překročení segmentu koprocesorem** (Processor Extension Segment Overrun)

Toto přerušení je vyvoláno, překročí-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.

INT 10 **Chybný TSS** (Invalid Task State Segment)

Přerušení nastane při pokusu o přepnutí na proces mající chybný TSS. Aby nenastalo, musí být splněny tyto podmínky:

- velikost TSS $\geq$ 44 (u 80386 $\geq$ 104) slabik,
- selektor LDT je správný,
- segment s LDT je uložen v paměti (P=1),
- selektor CS je správný,
- RPL z CS (CPL) = DPL instrukčního segmentu,
- selektor SS je správný,
- RPL z SS = DPL zásobníkového segmentu,
- RPL z SS = DPL instrukčního segmentu,
- selektory datových segmentů jsou správné,
- DPL datových segmentů $\geq$ CPL.

Toto přerušení do zásobníku uloží chybové slovo. Je-li splněna první podmínka (velikost TSS), provede se přepnutí procesu. Z toho plyne, že při nespl-

není první podmínky se přerušení INT 10 obsluhuje v rámci starého procesu. Je-li první podmínka splněna a není splněna některá z dalších, obsluhuje se přerušení v rámci nového procesu.

INT 11 **Výpadek segmentu** (Segment Not Present)

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

- při plnění registrů CS, DS nebo ES,
- 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)

Přerušení je generováno 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)

Toto přerušení se vyvolá při chybě detekované systémem ochran v těch případech, ve kterých nepatří pod žádné z dosud diskutovaných přerušení. Může nastat při pokusech o:

- 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,
- jakékoli jiné porušení pravidel systému ochrany.

INT 16 **Chyba koprocesoru** (Processor Extension Error)

Je-li v době čekání na dokončení operace koprocesoru detekována chyba aktivovaným signálem $\overline{\text{ERROR}}$, vyvolá procesor toto přerušení. Chyba pravděpodobně vznikla při výpočtu v koprocesoru (např. u matematického koprocesoru 80287: chybná operace, přeplnění, dělení nulou atd.).

Požádá-li více přerušení o obsluhu v jednom okamžiku, procesor použije toto **pořadí priorit**:

1. přerušení generované procesorem a instrukce INT,
2. krokovací režim,
3. NMI,
4. překročení segmentu koprocesorem,
5. vnější přerušení.

3.6.4 Přerušení v reálném režimu

V reálném režimu procesoru 80286 pracuje přerušovací systém shodně s procesorem 8086 (viz část „Přerušení“ na str. 22). Oproti 8086 se v reálném režimu procesoru 80286 mohou navíc vyskytnout tato přerušení generovaná procesorem: INT 5, 6, 7, 9, 16. Přerušení v reálném režimu **nepředávají chybové slovo**. Přerušení INT 8, 13 mají odlišný význam:

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

Toto přerušení 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)

Toto přerušení se vyskytne v reálném režimu tehdy, pokud nelze operand umístit do segmentu. Např. pokus umístit do segmentu slovo od offsetu 0FFFFh.

3.6.5 Spolupráce procesoru s koprocesorem

Jakmile procesor rozpozná na místě prvních pěti bitů operačního kódu instrukce vzorek „11011“ (ESC – příkaz pro koprocesor) nebo instrukci WAIT (čekání na dokončení operace koprocesoru), testuje v registru MSW tyto bity: EM (má-li se koprocesor programově emulovat), TS (zda došlo k přepnutí procesu v procesoru) a MP (je-li koprocesor fyzicky instalován). Poznamenejme, že všechny tyto bity může programátor nastavovat podle vlastní libovůle. Na nastavení těchto tří bitů reagují jinak příkazy koprocesoru (ESC) a jinak instrukce WAIT.

Je-li nastaveno EM=1 a procesor na místě operačního kódu rozpozná vzorek „11011“, generuje přerušení INT 7. Tím lze zahájit programovou emulaci požadovaného příkazu pro neinstalovaný koprocesor.

Přepnutí procesu v procesoru se nedotkne datových struktur v koprocesoru. Aby nedocházelo ke kolizím, nastavuje procesor bit TS na jedničku vždy, když dojde k přepnutí procesu, bez ohledu na to, bylo-li přepnutí vyvoláno programově nebo technicky. Detekování TS=1 při zahájení operace koprocesoru musí vyvolat přerušení, které předá řízení rutině zabezpečující výměnu datových struktur koprocesorem.

Při instrukci WAIT se přerušení INT 7 vyvolá tehdy, jsou-li současně nastaveny bity MP=TS=1. Je-li MP=0, koprocesor není instalován a WAIT neprovádí čekání.

Kombinace hodnot bitů EM, TS, MP a chování procesoru shrnuje tabulka na obr. 3.27.

3.7 Shrnutí pravidel pro předávání řízení

- Předávat řízení na jinou úroveň oprávnění lze pouze pomocí brány.

MSW			Instrukce	
EM	TS	MP	ESC	WAIT
0	0	0	*provede se*	*provede se*
0	0	1	*provede se*	*provede se*
0	1	0	INT 7	*provede se*
0	1	1	INT 7	INT 7
1	0	0	INT 7	*provede se*
1	0	1	INT 7	*provede se*
1	1	0	INT 7	*provede se*
1	1	1	INT 7	INT 7

Obr. 3.27. Nastavení bitů EM, TS, MP a chování procesoru

- 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:SP z TSS aktivního procesu hodnotami náležejícími momentální úrovni oprávnění. Původní hodnota SS:SP se uloží do zásobníku nové úrovně oprávnění. Při návratu (RETF) se tato hodnota vybere a použije pro nastavení SS:SP.

3.8 Počáteční nastavení procesoru

Po zapnutí napájení nebo po přijetí vnějšího signálu RESET jsou registry procesoru nastaveny podle obr. 3.28.

Registr	Obsah
F	0002h
MSW	FFF0h
IP	FFF0h
Selektor CS	F000h
Selektor DS	0000h
Selektor SS	0000h
Selektor ES	0000h
Báze CS	FF0000h
Báze DS	000000h
Báze SS	000000h
Báze ES	000000h
Limit CS	FFFFh
Limit DS	FFFFh
Limit SS	FFFFh
Limit ES	FFFFh
Báze IDT	000000h
Limit IDT	FFFFh

Procesor provádí tyto činnosti:

- zakáže přerušení (IF:=0),
- nastaví reálný režim bez koprocesoru (PE:=0, MP:=0, EM:=0),
- IDT se naplní nulami,
- DS, ES a SS jsou naplněny tak, aby ukazovaly do prvních 64 KB paměti,
- obsah CS:IP ukazuje na první instrukci, která musí být na adrese FF0000h:FFF0h=FFFFF0h,
- první instrukční segment zpřístupněný po inicializaci systému je posledních 64 KB paměti (CS=FF0000h).

Obr. 3.28. Nastavení registrů a provedení akcí po přijetí signálu RESET

Bezprostředně po inicializaci procesoru jsou adresové vodiče A_{20} až A_{23} nastaveny na jedničky při všech přístupech adresovaných přes registr CS. To umožní, i když jsme v reálném režimu, aby segment obsahující úvodní instrukce mohl být umístěn až na konci instalovaného adresového prostoru.

Tento stav trvá do první změny obsahu CS, potom jsou vodiče A_{20} až A_{23} vynulovány.

Další akce už provádí programové vybavení počítače. Bude-li procesor pracovat v **reálném režimu**, musí se provést tyto kroky:

1. nastavit příznakový registr F a registr stavu procesoru MSW,
2. v paměti vyhradit prostor pro zásobník a naplnit registry SS:SP,
3. nastavit všechny registry procesoru na požadované hodnoty,
4. nastavit tabulku přerušovacích vektorů,
5. inicializovat externí zařízení,
6. zavést do paměti program a data,
7. předat řízení programu (zpravidla instrukcí vzdáleného skoku JMP).

Bude-li procesor pracovat v **chráněném režimu**, musí se provést tyto kroky:

1. do paměti zavést programy a odpovídající tabulky popisovačů,
2. nastavit GDTR a IDTR,
3. zapnout chráněný režim nastavením bitu PE:=1 registru MSW,
4. provést blízký skok JMP proto, aby se zrušil obsah interních front procesoru, ve kterých jsou uloženy předvybrané instrukce (výběr instrukcí totiž závisí na zvoleném režimu procesoru),
5. naplnit segmentové registry DS a ES (viz též str. 172),
6. inicializovat ukazatel vrcholu zásobníku SS:SP,
7. provést vzdálený skok pro naplnění segmentového registru CS,
8. vytvořit TSS inicializačního procesu a nastavit obsah TR,
9. naplnit LDTR,

10. inicializovat externí zařízení,

11. zabezpečit obsluhu všech možných přerušení,

12. zahájit provádění prvního programu.

Pracuje-li procesor 80286 v chráněném režimu, nelze jej již programově přepnout zpět do reálného režimu. Jedinou možností je aktivovat signál RESET.

3.9 Rozšíření instrukcí 80286 oproti 8086

MOV V chráněném režimu při plnění segmentového registru procesor kontroluje přístupová práva a plní i neviditelnou část segmentového registru. Kontrola se neprovádí jenom při plnění neplatným selektorem (viz str. 88).

IMUL U znaménkového násobení lze zadat až tři operandy:

Instrukce	*Příklad*	*Komentář*
IMUL *r16,r/m16,imm8*	`IMUL DX,CX,10`	`; DX := CX * 10`
	`IMUL DX,Slovo,10`	`; DX := Slovo * 10`
IMUL *r16,imm8*	`IMUL DX,10`	`; DX := DX * 10`
IMUL *r16,r/m16,imm16*	`IMUL DX,CX,500`	`; DX := CX * 500`
	`IMUL DX,Slovo,500`	`; DX := Slovo * 500`

Osmibitový operand je před násobením znaménkově rozšířen na 16 bitů.

ROR, ROL, RCL, RCR, SAL, SAR, SHL, SHR Instrukce rotací a posuvů jsou rozšířeny o možnost zadání počtu rotací a posuvů přímou 8bitovou hodnotou větší než 1.

Instrukce	*Příklad*	*Komentář*
ROL *r/m8,imm8*	`ROL AL,7`	`; Rotace obsahu AH o 7 bitů vlevo`
ROL *r/m16,imm8*	`ROL BP,13`	`; Rotace obsahu BP o 13 bitů vlevo`

JMP Instrukce nepodmíněného skoku JMP je v chráněném režimu rozšířena o možnost předávat řízení do instrukčního segmentu pomocí brány (viz str. 88) a o možnost přepínat procesy (viz str. 98). Při instrukcích vzdálených skoků JMP *ptr16:16* (přímý skok) a JMP *m16:16* (nepřímý skok) je 32bitová adresa interpretována jako 16bitový selektor a 16bitový offset. Pomocí selektoru se vybere v tabulce LDT nebo GDT příslušný popisovač a kontroluje se úroveň oprávnění. Podle typu vybraného popisovače instrukce JMP provede: vzdálený skok na stejné úrovni oprávnění, vzdálený skok pomocí brány na stejné úrovni oprávnění, nebo přepnutí procesu skokem na popisovač TSS nebo bránu zpřístupňující TSS.

CALL Instrukce předání řízení do podprogramu CALL je v chráněném režimu rozšířena o možnost předávat řízení do instrukčního segmentu pomocí brány (viz str. 88) a o možnost přepínat procesy (viz str. 98). Při instrukcích vzdálených volání CALL *ptr16:16* (přímé volání) a CALL *m16:16* (nepřímé volání) je 32bitová adresa interpretována jako 16bitový selektor a 16bitový offset. Pomocí selektoru se vybere v tabulce LDT nebo GDT příslušný popisovač a kontroluje se úroveň oprávnění. Podle typu vybraného popisovače instrukce CALL provede: vzdálené volání na stejné úrovni oprávnění, vzdálené volání pomocí brány na stejné nebo do vyšší úrovně oprávnění, nebo přepnutí procesu voláním TSS nebo brány zpřístupňující TSS.

RET V chráněném režimu je provedena u mezisegmentových návratů navíc kontrola úrovně oprávnění selektoru CS plněného hodnotou ze zásobníku. Návrat se smí provést pouze do segmentu se stejnou nebo nižší úrovní oprávnění. Při změně úrovně oprávnění se ze zásobníku obnovují rovněž registry SS:SP a pokud segmentové registry DS a ES obsahují selektory ukazující do segmentů, které nelze na nové úrovni oprávnění používat, naplní se DS a ES neplatným selektorem.

PUSH Instrukce PUSH je rozšířena o možnost plnění zásobníku 8bitovou nebo 16bitovou přímou hodnotou:

Instrukce	*Příklad*	*Komentář*
PUSH *imm8*	`PUSH 10`	`; PUSH 10 znaménkově rozšířeno na 16 bitů`
PUSH *imm16*	`PUSH 500`	`; PUSH 500`

V procesoru 80286, stejně jako v 8086, se do zásobníku ukládá po dvojicích slabik. Proto je hodnota *imm8* znaménkově rozšířena na 16 bitů.

INT V chráněném režimu je operand uvedený za instrukcí interpretován jako index do tabulky popisovačů segmentů obsluhy přerušení (IDT – viz str. 103). V závislosti na typu popisovače se provede příslušná akce. Přerušením můžeme předat řízení pouze rutině pracující na stejné nebo vyšší úrovní oprávnění.

IRET V chráněném režimu je akce, kterou instrukce vyvolá, závislá na nastavení bitu NT v registru příznaků. Je-li NT=0, provede IRET návrat z přerušení bez přepnutí procesu. Instrukční segment, do kterého je předáváno řízení, musí mít stejnou nebo nižší úroveň oprávnění (úroveň oprávnění je zapsána v poli RPL selektoru CS vyzvednutém ze zásobníku). Při změně úrovně oprávnění se ze zásobníku obnovují rovněž registry SS:SP. Je-li NT=1, IRET obnoví původní proces přepnutím podle zpětného ukazatele TSS právě aktivního procesu.

Hodnotu bitů IOPL lze při plnění příznakového registru změnit pouze na úrovni oprávnění CPL=0.

WAIT viz též str. 111.

Viz též privilegované instrukce na str. 95.

3.10 Nové instrukce procesoru 80286

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. Zdrojový operand specifikuje

adresu paměti, od které je uložena 16bitová dolní a 16bitová horní hranice. Cílový operand obsahuje testovanou hodnotu. Instrukce zkontroluje, zda je hodnota cílového operandu větší nebo rovna dolní hranici, a zároveň zda je menší nebo rovna horní hranici. Pokud není podmínka splněna (index je mimo zadaný interval), generuje se přerušení INT 5.

SYNTAX: ⌜ BOUND *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
BOUND *r16,m16&16*	`BOUND BX,DvojSlovo`	`; Kontrola BX podle DvojSlovo`

Pro výše uvedený příklad mohou být meze uloženy následujícím způsobem:

```
DvojSlovo  LABEL  DWORD  ; DvojSlovo je ukazatel na 32bitový objekt
           DW     0      ; Dolní 16bitová mez je 0
           DW     99     ; Horní 16bitová mez je 99
```

Je-li $0 \leq BX \leq 99$, potom se přerušení negeneruje.

PUSHA — *Push All General Registers*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce PUSHA ukládá do zásobníku osm 16bitových všeobecných registrů v pořadí AX, CX, DX, BX, SP, BP, SI a DI. Hodnota SP se uloží taková, jaká byla před provedením instrukce. Po provedení instrukce se SP sníží o 16. Instrukce nemá operand.

POPA — *Pop All General Registers*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce POPA vybírá ze zásobníku osm 16bitových všeobecných registrů v pořadí DI, SI, BP, SP, BX, DX, CX a AX. Po provedení instrukce obsahuje SP hodnotu vybranou ze zásobníku (viz PUSHA). Instrukce nemá operand.

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. Parametr *lokální_prostor* je počet slabik, které se v zásobníku vyhradí pro uložení lokálních dat. Parametr *úroveň_vnoření* určuje míru 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 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 se naplní zapamatovaným obsahem SP (z bodu 2).

5. V zásobníku se vyhradí *lokální_prostor* slabik pro uložení lokálních proměnných tohoto podprogramu. Vyhrazení se provede tak, že se SP sníží o hodnotu *lokální_prostor*.

SYNTAX: ⌜ ENTER *lokální_prostor,úroveň_vnoření*

Instrukce	*Příklad*	*Komentář*
ENTER *imm16,imm8*	`ENTER 12,2`	`; Kopíruje 2 ukazatele a vyhradí 12 B`

Použití ENTER je ukázáno na tomto příkladu (viz obr. 3.29): 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.

Odstranění struktur definovaných v zásobníku instrukcí ENTER lze provést instrukcí LEAVE.

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 nemá žádný parametr a provádí posloupnost těchto akcí:

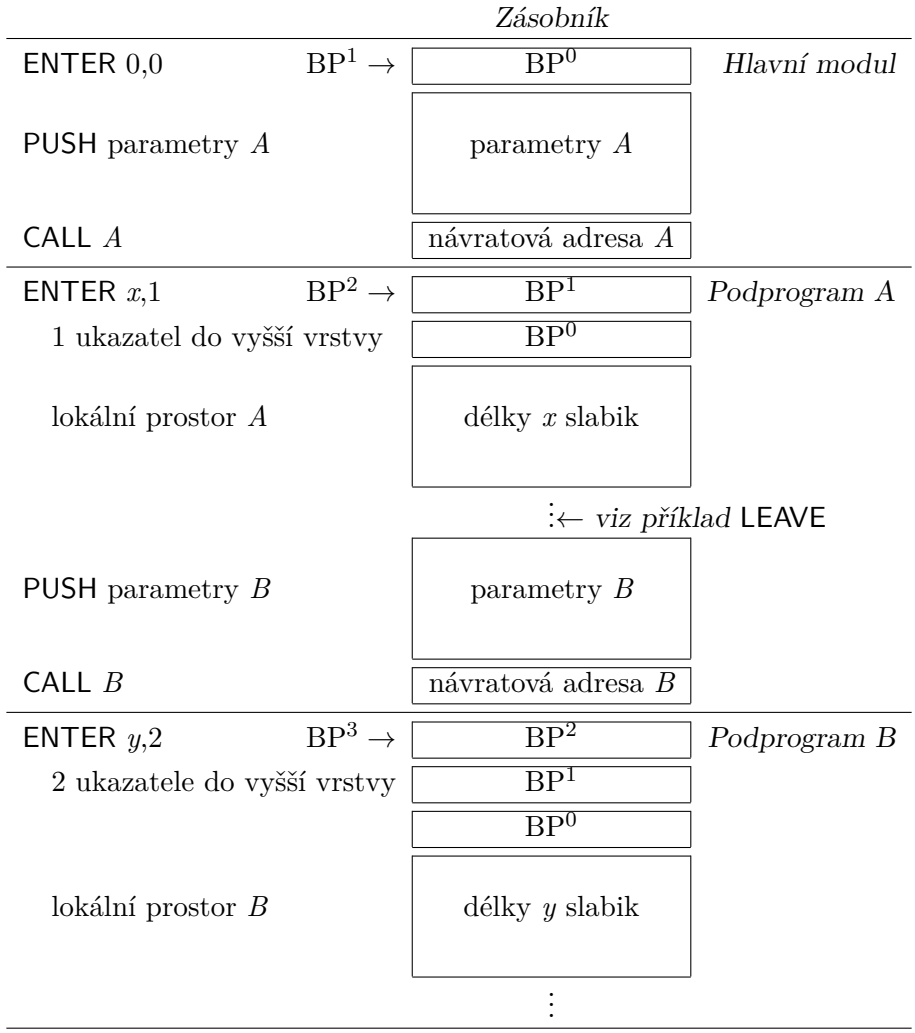
MOV	SP,BP	naplní SP obsahem BP,
POP	BP	ze zásobníku vyzvedne obsah BP.

SYNTAX: ⌜ LEAVE

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. 3.29 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.



Obr. 3.29. Příklad použití instrukce ENTER

INS/INSB/INSW — *Input From Port to String*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL≤IOPL**

Instrukce INS přenáší data určená hodnotou registru DX ze V/V brány do paměti na adresu ES:DI. Instrukce patří do skupiny řetězcových instrukcí a vztahují se na ni pravidla popsaná na str. 64. Pokud je v instrukci uveden cílový operand, použije se pouze pro kontrolu typu překladačem generované instrukce. Po provedení instrukce se aktualizuje hodnota registru DI podle příznaku DF. Před instrukcí může být uveden prefix REP.

SYNTAX: [INS *cílový_operand*,DX
INSB
INSW

Instrukce	*Příklad*	*Komentář*
INSB	`INSB`	`; Přenos slabiky z brány podle DX na adresu ES:DI`
INSW	`INSW`	`; Přenos slova z brány podle DX na adresu ES:DI`

OUTS/OUTSB/OUTSW — *Output String to Port*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL≤IOPL**

Instrukce OUTS přenáší data z paměti na V/V bránu určenou hodnotou registru DX. Čtené místo paměti je určeno adresou v registrech DS:SI. Po provedení instrukce se aktualizuje hodnota registru SI podle pravidel řetězcových instrukcí (viz str. 64).

SYNTAX: [OUTS DX,*zdrojový_operand*
[OUTSB
[OUTSW

Instrukce	*Příklad*	*Komentář*
OUTSB	`OUTSB`	`; Přenos slabiky z adresy DS:SI na bránu podle DX`
OUTSW	`OUTSW`	`; Přenos slova z adresy DS:SI na bránu podle DX`

LMSW *Load Machine Status Word*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Instrukce LMSW naplní registr MSW hodnotou uloženou v operandu. 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.

V chráněném režimu můžeme instrukci použít pouze na úrovni oprávnění CPL=0 (privilegovaná instrukce). Pomocí této instrukce **nelze** přepnout procesor zpět do reálného režimu.

SYNTAX: ⌜ LMSW *operand*

Instrukce	*Příklad*	*Komentář*
LMSW *r/m16*	`LMSW AX`	`; Naplní registr MSW obsahem AX`
	`LMSW Slovo`	`; Naplní registr MSW obsahem Slovo`

SMSW *Store Machine Status Word*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce SMSW ukládá obsah registru MSW do operandu. Instrukci lze provést v reálném režimu a na libovolné úrovni oprávnění chráněného režimu.

SYNTAX: ⌜ SMSW *operand*

Instrukce	*Příklad*	*Komentář*
SMSW *r/m16*	`SMSW DX`	`; Naplní DX obsahem MSW`
	`SMSW Slovo`	`; Naplní Slovo obsahem MSW`

CLTS — *Clear Task Switched Flag*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Instrukce CLTS nuluje bit TS registru MSW. CLTS je privilegovaná instrukce, kterou v chráněném režimu lze provést pouze s nejvyšší úrovní oprávnění (CPL=0).

SYNTAX: ⌜ CLTS

LTR — *Load Task Registr*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Privilegovaná instrukce LTR naplní registr TR obsahem operandu. Příslušný TSS je tím označen jako právě aktivní. Instrukce však nevyvolá přepnutí procesu. V reálném režimu není instrukce rozpoznána a generuje se přerušení INT 6.

SYNTAX: ⌜ LTR *operand*

Instrukce	*Příklad*	*Komentář*
LTR *r/m16*	`LTR AX`	`; Naplnění registru TR obsahem AX`
	`LTR Slovo`	`; Naplnění registru TR obsahem Slovo`

STR — *Store Task Register*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce STR ukládá obsah registru TR do operandu. V reálném režimu není instrukce rozpoznána a generuje se přerušení INT 6.

SYNTAX: ⌜ STR *operand*

Instrukce	*Příklad*	*Komentář*
STR *r/m16*	`STR AX`	`; Uložení obsahu TR do AX`

LGDT — *Load Global Descriptor Table Register*
LIDT — *Load Interrupt Descriptor Table Register*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Instrukce LGDT (LIDT) plní obsah registru GDTR (IDTR) 16bitovým limitem a 24bitovou bází segmentu, v němž je uložena tabulka GDT (IDT). Registr se plní obsahem operandu, který má délku 6 slabik. První dvě slabiky musí obsahovat 16bitový limit segmentu a tři následující slabiky obsahují bázi segmentu. Poslední slabika je nevyužita (rezerva pro 80386).

SYNTAX:
- LGDT *operand*
- LIDT *operand*

Instrukce	*Příklad*	*Komentář*
LGDT *m16&32*	`LGDT AdrGDT`	`; AdrGDT je 48bitový objekt v paměti`
LIDT *m16&32*	`LIDT AdrIDT`	`; Naplní registr IDT obsahem AdrIDT`

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 slabiky 24bitovou bází segmentu a obsah poslední slabiky zůstává nedefinován (rezerva pro 80386).

SYNTAX:
- SGDT *operand*
- SIDT *operand*

Instrukce	*Příklad*	*Komentář*
SGDT *m16&32*	`SGDT AdrGDT`	`; AdrGDT je 48bitový objekt v paměti`
SIDT *m16&32*	`SIDT AdrIDT`	`; Uloží obsah registru IDT do AdrIDT`

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 24bitovou bází a 16bitovým limitem segmentu tabulky LDT.

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ LLDT *operand*

Instrukce	*Příklad*	*Komentář*
LLDT *r/m16*	`LLDT BX`	`; Naplní registr LDTR obsahem BX`

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.

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ SLDT *operand*

Instrukce	*Příklad*	*Komentář*
SLDT *r/m16*	`SLDT AX`	`; Uloží selektor LDTR do AX`

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 cílovém operandu na nižší úroveň oprávnění z obou zadaných operandů. Zdrojový operand obsahuje libovolnou hodnotu. Je-li hodnota RPL cílového operandu menší než hodnota bitů 1 a 0 zdrojového operandu, je nastaven příznak ZF na jedničku a pole RPL cílového operandu je zvýšeno na hodnotu nejnižších dvou bitů zdrojového operandu. Jinak je příznak ZF nulován a cílový operand zůstane nezměněn. Viz též příklad na str. 94.

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ ARPL *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
ARPL *r/m16,r16*	`ARPL Slovo,AX`	`; RPL(Slovo) := max(RPL(Slovo),RPL(AX))`

LSL — *Load Segment Limit*

O	D	I	T	S	Z	A	P	C
					*			

POPIS:

Instrukce LSL naplní cílový operand 16bitovým limitem segmentu, jehož selektor je uložen ve zdrojovém operandu, a nastaví příznak ZF na jedničku tehdy, je-li přístup k segmentu povolen (CPL má odpovídající úroveň oprávnění vzhledem k DPL segmentu). Jsou-li přístupem k segmentu porušena přístupová práva, je ZF nulován a obsah cílového operandu zůstane nezměněn.

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ LSL *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
LSL *r16,r/m16*	`LSL AX,Slovo`	`; AX := limit ze selektoru podle Slovo`

LAR — *Load Access Rights Byte*

O	D	I	T	S	Z	A	P	C
					*			

POPIS:

Instrukce LAR naplní 16bitový cílový operand obsahem slabiky *přístupová práva* popisovače segmentu, který je adresován selektorem uloženým ve zdrojovém operandu. Selektor je uložen buď v 16bitovém registru, nebo v paměti.

Před naplněním cílového registru 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 cílového operandu slabikou přístupových práv popisovače, nuluje se dolní slabika a nastaví se ZF=1. Není-li podmínka splněna, nastaví se ZF=0.

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ LAR *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
LAR *r16,r/m16*	`LAR AX,Slovo`	`; AH:=přist. práva podle Slovo, AL:=0`

VERR — *Verify a Segment for Reading*
VERW — *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 a ES.

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

$$\mathrm{DPL} \geq \mathrm{Max}(\mathrm{CPL}, \mathrm{RPL})$$

(pouze v případě, že se jedná o instrukční segment s C=1, je tato podmínka splněna vždy).

Při použití instrukce v reálném režimu se generuje přerušení INT 6, protože instrukce není rozpoznána.

SYNTAX: ⌜ VERR *operand*
⌜ VERW *operand*

Instrukce	*Příklad*	*Komentář*
VERR *r/m16*	`VERR BX`	`; Kontrola selektoru na čtení segmentu`
VERW *r/m16*	`VERW Slovo`	`; Kontrola selektoru na zápis do segmentu`

4 Intel 80386

80386 je prvním 32bitovým procesorem firmy Intel. Ochranné známky **i386**, **80386DX** a **386DX** jsou obchodními názvy jednoho a téhož procesoru, který v dalším textu budeme podle předchozích konvencí označovat 80386.

Typ pojmenovaný **80386SX** nebo **386SX** je procesor vnitřně plně slučitelný s 80386, ale má vnější 16bitovou strukturu. Vztah 80386 a 80386SX je podobný jako 8086 a 8088. Vše, co v dalším textu platí pro 80386, platí i pro 80386SX. Výjimky budou uvedeny na závěr této kapitoly.

Procesor 80386 je integrován do keramického pouzdra PGA se 132 vývody (viz též str. 229). Zpracovává 32bitová vnější i vnitřní data a 32bitovou adresu. Na čipu je společně s procesorem i jednotka správy paměti obhospodařující 4 GB fyzické a 64 TB virtuální paměti. Základní rozdíly oproti 80286 jsou: procesor umí pracovat s 32bitovými operandy, segmenty mohou mít velikost až 4 GB, procesor podporuje stránkování paměti a *virtuální 8086* režim.

Procesor pracuje ve třech možných režimech: *reálném*, *chráněném* a *virtuálním 8086*. *Reálný režim* je slučitelný s 8086 (s řadou rozšíření), *chráněný režim* je vlastní 80386 a je slučitelný s chráněným režimem 80286 tak, že programy určené chráněnému režimu 80286 lze provozovat beze změny. V režimu *virtuální 8086*, do kterého lze přepnout proces v rámci chráněného režimu, je nabízena možnost sdílet např. MS-DOSu jako jeden proces operačního systému Unix.

K procesoru 80386 byl vyprojektován 32bitový matematický koprocesor dodávaný pod označením 80387. Tento koprocesor je opět programově slučitelný s předchozími typy.

V terminologii procesoru 80386 rozlišujeme tři druhy adres:

Logická adresa (v terminologii 80286 se nazývá virtuální adresa) je složena z 16bitového selektoru a 32bitového offsetu (adresuje 64 TB virtuální

paměti). Tato adresa je algoritmem segmentační jednotky převedena na lineární adresu.

Lineární adresa je 32bitová adresa (tj. adresuje 4 GB). Není-li v činnosti stránkovací jednotka, potom lineární adresa ukazuje už přímo do fyzické paměti.

Fyzická adresa je transformována činností stránkovací jednotky z lineární adresy. Je rovněž 32bitová (tj. adresuje 4 GB fyzické paměti). Není-li stránkovací jednotka zapnuta, je fyzická adresa totožná s lineární adresou.

4.1 Architektura 80386

Procesor je složen ze 6 paralelně pracujících jednotek. **Sběrnicová jednotka** (Bus Interface Unit) zprostředkovává styk obvodů procesoru s okolním světem. Přijímá požadavky na čtení instrukcí a přenos dat mezi procesorem a pamětí. Požadavky provádí podle priorit. Organizuje činnost sběrnic, a tím přenos dat mezi procesorem, pamětí, V/V sběrnicemi a koprocesorem. **Jednotka předvýběru instrukcí** (Code Prefetch Unit) v době, kdy sběrnicová jednotka nepracuje se sběrnicí, vybírá prostřednictvím sběrnicové jednotky z paměti slabiky tvořící další instrukce a ukládá je do 16 slabik dlouhé fronty. Z této fronty vybírá slabiky **instrukční jednotka** (Instruction Decode Unit) a převádí je na mikroinstrukce a ukládá do další fronty. Mikroinstrukce přebírá a zpracovává **prováděcí jednotka** (Execution Unit), která ke své činnosti vyžaduje i spolupráci ostatních jednotek.

Segmentační jednotka (Segmentation Unit) převádí logickou adresu na lineární podle požadavků prováděcí jednotky. Vypočtená lineární adresa je předána **stránkovací jednotce** (Paging Unit). Je-li tato zapnuta, transformuje lineární adresu na fyzickou, která je již skutečným ukazatelem do fyzické paměti. Stránkovací jednotka má implementovánu vyrovnávací paměť na několik posledních (a pravděpodobně častěji používaných) transformovaných adres (TLB).

Segmentační jednotka rovněž podporuje 4úrovňový systém ochrany paměti. Stránkovací jednotka slouží k realizaci některých mechanismů virtualizace paměti. Do fyzické paměti data ukládá po 4 KB stránkách. Tyto stránky

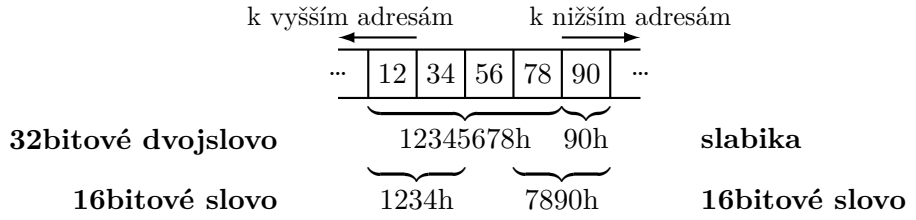
jsou zvnějšku stránkovací jednotky zpřístupněny lineární adresou a interně (stránkovací jednotkou) fyzickou adresou. Stránkovací jednotka napomáhá operačnímu systému manipulovat se stránkami podle instalované kapacity fyzické paměti a potřeb uživatelů.

Po přijetí signálu RESET je procesor nastaven do *reálného režimu*. V tomto režimu pracuje stejně jako 8086 s některými rozšířeními. Do chráněného režimu se procesor přepne po provedení instrukce MOV CR0,*operand* nastavující bit PE (pro kompatibilitu s 80286 lze provést i instrukci LMSW). Na rozdíl od 80286 lze v 80386 přejít z chráněného režimu do reálného vynulováním bitu PE.

Procesor 80386 umí spolupracovat s koprocesory 80387 a 80287. Poněvadž 80387 pracuje s 32bitovým komunikačním protokolem, musí programové vybavení vědět, který koprocesor je připojen. K tomu slouží bit ET v registru CR0.

4.2 Typy dat

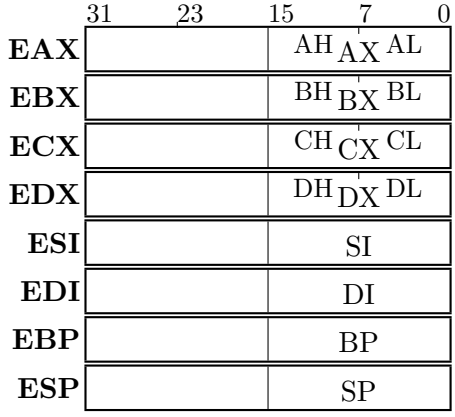
Oproti předchozím typům procesor 80386 zpracovává navíc 32bitová dvojslova. Dvojslovo smí začínat na libovolné slabice (tedy nikoli jenom na sudé nebo dělitelné čtyřmi). Podobně jako 16bitové slovo (viz obr. 2.1 na str. 15) je v paměti uloženo i 32bitové dvojslovo (viz obr. 4.1) včetně obráceného pořadí slabik ve slově.



Obr. 4.1. Formát 32bitového dvojslova a 16bitového slova v paměti

Dále procesor 80386, oproti předcházejícím typům, pracuje s bitovými poli a bitovými řetězci.

4.3 Registry procesoru 80386



Obr. 4.2. Všeobecné registry 80386

Registry procesoru 80386 jsou 32bitové, i když jejich struktura vychází ze starších typů procesorů. Ve skupině **všeobecných registrů** jsou 32bitové registry pojmenovány EAX, EBX, …, ESP. Můžeme také používat jejich 16bitové (AX, BX, …) a 8bitové (AH, AL, …) části.

Příznakový registr EFLAGS je rovněž 32bitový (viz obr. 4.3). Oproti příznakovému registru procesoru 80286 (viz obr. 3.1 na str. 76) je doplněn o následující bity:

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	0	0	0	0	**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. 4.3. Příznakový registr EFLAGS procesoru 80386

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.

RF (Resume Flag) maskuje opakování ladicího přerušení.

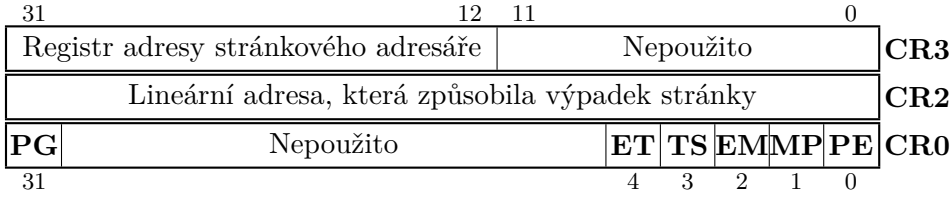
Do skupiny **segmentových registrů** patří šest 16bitových registrů určených pro uložení selektoru. Stejné jako ve starších typech procesorů jsou registry **CS** (Code Segment) a **SS** (Stack Segment). Nové v 80386 jsou registry pro uložení selektoru datových segmentů, jejich počet byl rozšířen na čtyři:

DS, **ES**, **FS** a **GS**. Velikost viditelných částí registrů se nezměnila (selektor je stále 16bitový), ale zvětšila se neviditelná část tak, že báze segmentu je 32bitová.

Skupina **registrů systémových adres** obsahuje čtyři registry známé z procesoru 80286: **GDTR**, **LDTR**, **IDTR** a **TR**. Obsahují adresy tabulek GDT, LDT, IDT a TSS. Všechny tyto registry mají 32bitovou bázi segmentu a 16bitový limit. Registry LDTR a TR mají navíc 16bitový selektor.

Ve skupině **řídicích registrů** je 32bitový registr **EIP** (Extended Instruction Pointer) obsahující 32bitový offset adresy právě prováděné instrukce. Dolních 16 bitů tohoto registru je pojmenováno (pro kompatibilitu s dřívějšími typy) IP.

Do této skupiny také patří tři 32bitové speciální řídicí registry (Control Registers) **CR0**, **CR2** a **CR3**. Tyto, společně s registry systémových adres, udržují základní informace o stavu procesoru. Plní se variantou instrukce MOV CR*i*.



Obr. 4.4. Řídicí registry CR0, CR2 a CR3 procesoru 80386

Registr **CR0** (viz obr. 4.4) obsahuje informace o celém systému a ne jenom o konkrétním procesu. Dolních 16 bitů je nazýváno **MSW** (pro kompatibilitu s 80286).

Jednotlivé bity CR0 mají následující význam:

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, **EM**, **TS** viz 80286 na str. 76.

ET (Extension Type) sděluje typ instalovaného matematického koprocesoru. Bit nastavuje procesor během inicializace (po přijetí signálu RESET). Detekuje-li 80287, nastaví ET=0, pro 80387 nastaví ET=1. Na základě bitu ET procesor používá buď 16bitový, nebo 32bitový protokol komunikace s koprocesorem. Bit lze nastavovat i programově.

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

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 přerušení INT 14. Registr je určen pouze ke čtení.

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.

Procesor 80386 disponuje šesti 32bitovými **ladicími registry**. V registrech **DR0 až DR3** mohou být uloženy lineární adresy ladicích bodů. Jakmile se má zpřístupnit obsah jedné z těchto adres, je generováno ladicí přerušení (INT 1). Registr **DR7** je příkazový a **DR6** stavový (podrobnosti viz část „Ladicí nástroje“).

Registry **TR6** a **TR7** jsou určeny pro přístup k TLB. Budou společně s TLB popsány v části o stránkování.

Na závěr části věnované registrům procesoru 80386 uveďme tabulku (obr. 4.5) informující o přístupnosti registrů v jednotlivých operačních režimech procesoru. Položka označená CPL=0 sděluje, že registr je přístupný pouze na úrovni oprávnění 0. Podrobnosti o použití registru příznaků v režimu *virtuální 8086* budou uvedeny ve zvláštní části.

4.4 Popisovače segmentů

Logická adresa se skládá z 16bitového **selektoru** a 32bitového **offsetu**.

Režim:	*reálný*		*chráněný*		*virtuální 8086*	
Registr	*Zápis*	*Čtení*	*Zápis*	*Čtení*	*Zápis*	*Čtení*
Všeobecné r.	ano	ano	ano	ano	ano	ano
Segmentové r.	ano	ano	ano	ano	ano	ano
R. příznaků	ano	ano	ano	ano		
CR*i*	ano	ano	CPL=0	CPL=0	ne	ano
GDTR	ano	ano	CPL=0	ano	ne	ano
IDTR	ano	ano	CPL=0	ano	ne	ne
LDTR	ne	ne	CPL=0	ano	ne	ne
TR	ne	ne	CPL=0	ano	ne	ne
DR*i*	ano	ano	CPL=0	CPL=0	ne	ne
TR*i*	ano	ano	CPL=0	CPL=0	ne	ne

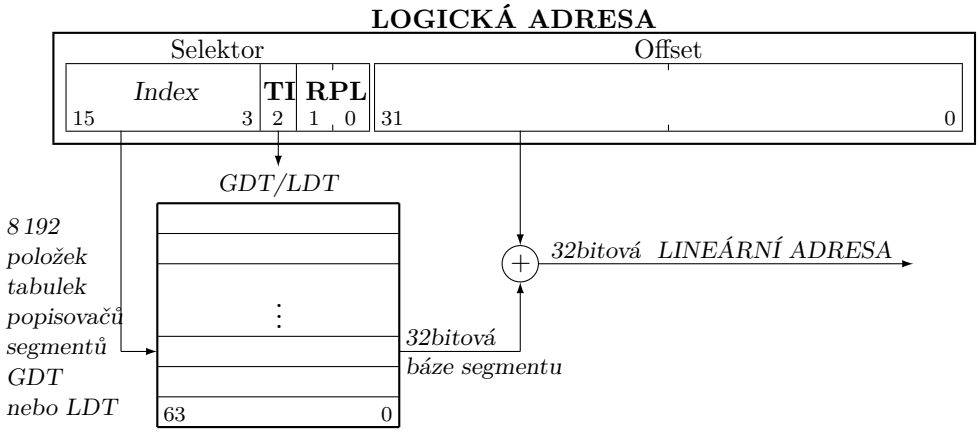
Obr. 4.5. Zpřístupnění registrů v operačních režimech procesoru 80386

Z toho vyplývá, že velikost **segmentu** je až 4 GB. Adresový prostor, stejně jako v 80286, dělíme na lokální a globální. Procesor 80386 pracuje rovněž se 4 úrovněmi oprávnění. Proto je tvar selektoru shodný s 80286 (viz str. 78). Tabulky popisovačů jsou také dvou typů: **GDT** a **LDT**, obě mají 1 až 8 192 popisovačů segmentů délky 8 B (tj. kapacitu 8 B až 64 KB). Prostřednictvím těchto tabulek se logická adresa transformuje na **lineární adresu** (viz obr. 4.6).

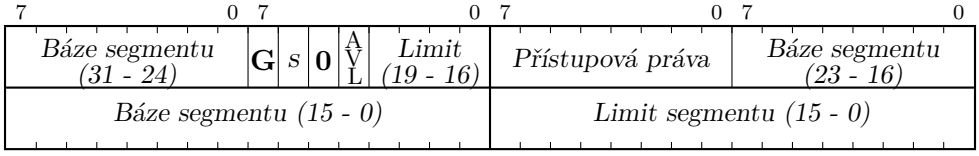
První položku GDT procesor nepoužívá. Selektor, který má index a TI nulové, se nazývá **neplatný selektor** (Null Selector). Lze jím naplnit segmentové registry (mimo CS a SS), aniž by procesor generoval chybové přerušení. Procesor bude toto přerušení generovat až při pokusu o použití segmentového registru,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

Popisovače segmentů mají velikost a tvar shodný s 80286, ale využívají ty slabiky, které v 80286 musely mít nulový obsah (viz obr. 4.7).

Protože v procesoru 80386 je offsetová část adresy 32bitová, musel být odpovídajícím způsobem zvětšen i rozsah báze a limitu segmentu. V popisovači segmentu je pro uložení báze rezervováno 32 bitů a pro uložení limitu 20



Obr. 4.6. Transformace logické adresy na lineární pomocí tabulek popisovačů segmentů v procesoru 80386



Obr. 4.7. Formát popisovače segmentů procesoru 80386

bitů a bit G:

G (Granularity) nulový 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).

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.

4.4.1 Popisovač datového segmentu

Popisovač datového segmentu má stejný obsah slabiky *přístupová práva* jako 80286 (viz str. 80). Bit *s* (viz obr. 4.7) 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 adresy 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 80286.

4.4.2 Popisovač instrukčního segmentu

Popisovač instrukčního segmentu má stejný obsah slabiky *přístupová práva* jako 80286 (viz str. 82). Bit *s* (viz obr. 4.7) je v popisovači datového segmentu označován D a má tento význam:

D (Default) sděluje implicitní velikost adres a operandů používaných v tomto segmentu. Je-li D=0, je implicitní velikost adres a operandů 16 bitů (odpovídá 80286) 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. 4.8.

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 přerušení INT 13.

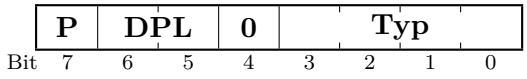
Hodnota bitu D (resp. prefixu změny velikosti adresy) má také vliv na výběr adresovacích technik (viz odstavec „Adresovací techniky“ na str. 182).

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. 4.8. Použití instrukčních prefixů 66h a 67h v závislosti na parametru D

4.4.3 Popisovač systémového segmentu

V 80386 je více typů systémových segmentů než v 80286. Obsah slabiky *přístupová práva* je však konstruován tak, aby byla zachována kompatibilita s 80286 (viz obr. 4.9). Bit *s* (viz obr. 4.7) v popisovači systémového segmentu není použit.



Obr. 4.9. Přístupová práva popisovače systémového segmentu

Typ=0 … nepovolená hodnota,
1 … TSS neaktivního procesu 80286 (str. 97),
2 … LDT 80286 (str. 83) a 80386,
3 … TSS aktivního procesu 80286 (str. 97),
4 … brána pro předání řízení 80286 (str. 91),
5 … brána zpřístupňující TSS 80286 (str. 98) a 80386,
6 … brána pro maskující přerušení 80286 (str. 104),
7 … brána pro nemaskující přerušení 80286 (str. 104),
8 … nepovolená hodnota,
9 … TSS neaktivního procesu 80386,
A … nepovolená hodnota,
B … TSS aktivního procesu 80386,
C … brána pro předání řízení 80386,
D … nepovolená hodnota,

E … brána pro maskující přerušení 80386,
F … brána pro nemaskující přerušení 80386.

4.5 Systém ochran 80386

Význam a funkce systému ochran v 80386 jsou stejné jako v procesoru 80286. Procesor poskytuje 4 stejně rozvrstvené úrovně oprávnění a indikátory DPL, CPL, RPL a EPL (viz str. 87). Ta pravidla, která se v 80286 vztahovala na registry DS a ES, se vztahují zde i na nové registry FS a GS.

4.5.1 Privilegované instrukce

Seznam privilegovaných instrukcí je oproti 80286 (viz str. 95) rozšířen o instrukce pracující s registry CR*i*, DR*i* a TR*i*, tj. o instrukce:

MOV do/z CR*i*
MOV do/z DR*i*
MOV do/z TR*i*

V/V instrukce lze v chráněném režimu 80386 bez dalších kontrol provádět na úrovni oprávnění CPL≤IOPL (IOPL je dvoubitová hodnota v příznakovém registru). Do V/V instrukcí patří IN, OUT, INS, OUTS, REP INS a REP OUTS. Je-li CPL>IOPL a proces je řízen TSS typu 80286, je při výskytu V/V instrukce generováno přerušení INT 13. Je-li CPL>IOPL a proces je řízen TSS typu 80386, je při výskytu instrukce kontrolována mapa přístupných V/V bran uložená v TSS 80386. Je-li pro daný proces mapou povolen přístup k odkazované bráně, provede se V/V instrukce. Není-li povolen, generuje se INT 13. V režimu *virtuální 8086* se IOPL neuplatňuje a testuje se pouze mapa. Další podrobnosti viz str. 150.

Stejně jako v 80286 lze instrukce CLI a STI provádět pouze na úrovni oprávnění CPL≤IOPL. V opačném případě výskyt instrukce manipulující s příznakem IF způsobí přerušení INT 13. Na rozdíl od 80286 **nepatří** prefix LOCK do této skupiny a není tedy závislý na IOPL.

4.5.2 Stránková ochrana

Procesor 80386 rozšiřuje systém zabezpečení o stránkovou ochranu při zapnutí stránkovací jednotky (PG=1). Procesy jsou pro účely stránkové ochrany děleny do dvou kategorií – na uživatelské (CPL=3) a neuživatelské (CPL<3) – viz bity U a W v části „Stránkování“.

4.6 Stránkování

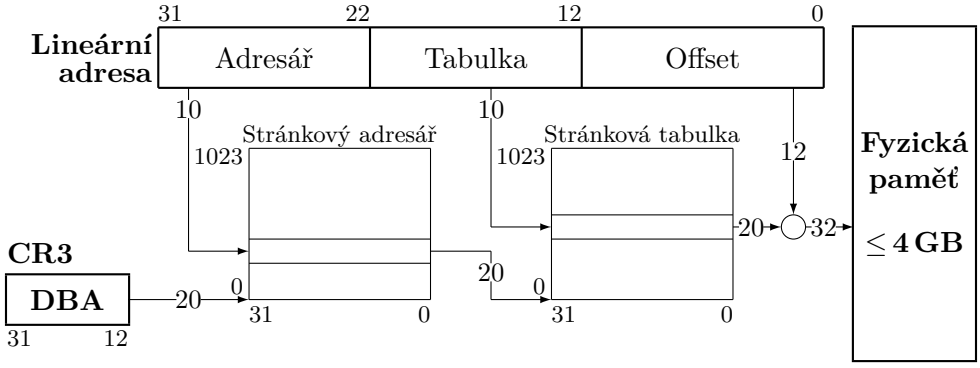
Procesor 80386 poskytuje operačnímu systému dvě úrovně transformací adres. V první, vyšší úrovni je **logická adresa** mechanismem segmentační jednotky (selektor ukazuje do GDT nebo LDT na položku obsahující bázi segmentu, ke které se přičítá offset) transformována na **lineární adresu** (viz obr. 4.6). Ve druhé úrovni je lineární adresa stránkováním transformována na **fyzickou adresu**. Fyzická adresa ukazuje již přímo do paměťových obvodů na adresovaný objekt.

Stránkování je, na rozdíl od segmentování, jiný způsob práce s pamětí. Je velmi užitečné zvláště ve víceúlohových systémech. Stránkováním jsou programy a data rozděleny do stránek pevných velikostí, segmentováním se programy a data dělí na různě velké segmenty podle modulární struktury programu. Kombinujeme-li segmentování se stránkováním, pracuje stránkování pod segmentováním, tzn., že adresy produkované segmentováním jsou dále zpracovány stránkováním.

Stránkovací jednotka v procesoru 80386 realizuje mechanismus virtuální paměti, který dovoluje programům používat větší kapacitu paměti, než je skutečně instalovaná fyzická paměť tak, že celá kapacita virtuální paměti je uložena po **stránkách** v externím zařízení (např. na disku) a 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. Dále zajišťuje stránkovou ochranu na základě přístupových práv.

V procesoru 80386 je segmentování povinnou částí zpracování adresy, naopak stránkování je volitelné. Stránkování je zapnuto jenom tehdy, je-li bit

PG v CR0 nastaven na jedničku. Hodnotu tohoto bitu můžeme měnit instrukcí MOV CR0,*operand*. 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. 4.10.



Obr. 4.10. Transformace lineární adresy na fyzickou pomocí stránkového adresáře a stránkové tabulky v procesoru 80386

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**.

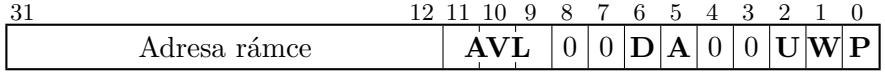
V CR3 není lineární adresa, ale fyzická, protože tento registr ukazuje na stránkový adresář, a tudíž jím nemůže být mapován. Stránka, ve které je umístěn stránkový adresář, může být mechanismem virtualizace paměti odložena do vnější paměti až tehdy, není-li proces, který adresář používá, aktivní. Každý 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 přerušení startující rutinu

zavádějící stránky do fyzické paměti.

Vybraná položka stránkového adresáře ukazuje na začátek **stránkové tabulky** (Page Table), kterých může být teoreticky až 1 024. Stránkové tabulky opět smějí 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. 4.11. Jednotlivé bity mají tento význam:



Obr. 4.11. Tvar specifikátoru stránkového adresáře a stránkové tabulky

Adresa rámce je horních 20 bitů adresy rámce, na který položka ukazuje. Dolních 12 bitů se doplní nulami.

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

D (Dirty) nastavuje procesor na jedničku při změně obsahu rámce, na který specifikátor právě ukazuje. Ve stránkovém adresáři je tento bit nedefinován. 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 při každém použití tohoto specifikátoru. Bit má stejný význam jako v popisovači segmentů (viz str. 81).

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.

P (Present) jedničkové 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 může operační systém disponovat na jiné účely (např. kde na disku je stránka momentálně uložena). Pokus o přístup ke stránce, která má nastaveno P=0 (ve stránkovém adresáři nebo stránkové tabulce), vyvolá přerušení INT 14.

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. 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.

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.

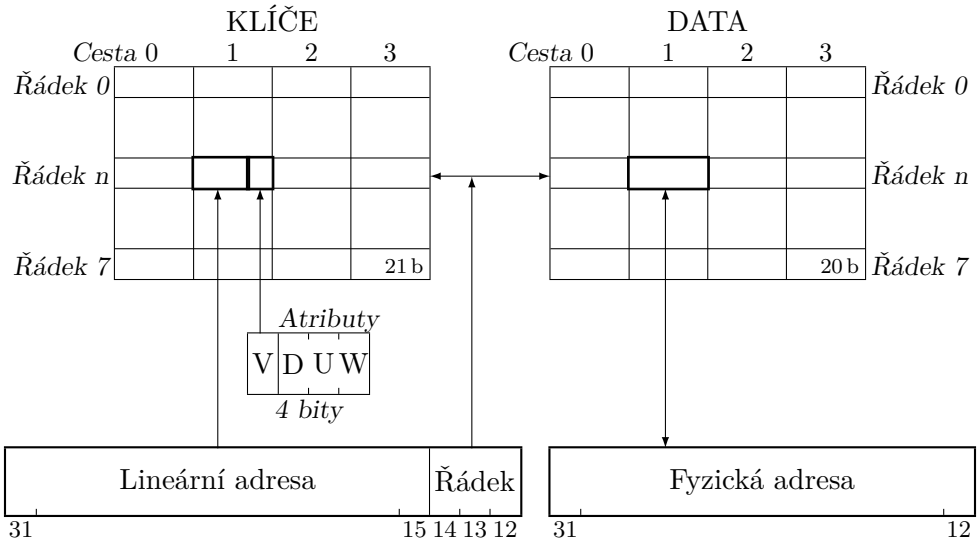
Na závěr poznamenejme, že není žádný vztah mezi hranicemi stránek a hranicemi segmentů. Stránka má velikost právě 4 KB a začíná na adrese

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

4.6.1 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 80386 implementovánu **TLB** (Translation Look-aside Buffer), která je vyrovnávací pamětí pro posledně transformované adresy včetně příznaků.

TLB 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 je patrná z obr. 4.12.



Obr. 4.12. Struktura TLB 80386

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 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 Intel nesděluje) a ta se vyřadí.

TLB je běžně obsluhována stránkovací jednotkou 80386 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. 4.13.

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

Obr. 4.13. Testovací registry TR6 a TR7 procesoru 80386

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).

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 $\overline{\mathrm{W}}$, $\overline{\mathrm{U}}$, $\overline{\mathrm{D}}$ 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 80386 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 $\overline{\mathrm{D}}$, $\overline{\mathrm{U}}$ a $\overline{\mathrm{W}}$ 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.

4.7 Přepínání procesů

Základní principy přepínání procesů v 80386 jsou stejné jako v procesoru 80286 (viz str. 96). Stav každého z rozpracovaných procesů je uložen v segmentu stavu procesu (TSS). Poněvadž je v 80386, oproti 80286, rozšířen počet a velikost registrů, je také odpovídajícím způsobem rozšířen TSS.

Ukazatel na TSS právě aktivního procesu je uložen, stejně jako v procesoru 80286, v registru **TR**. Stejná jsou rovněž pravidla pro používání **brány zpřístupňující TSS** (viz str. 98).

4.7.1 Segment stavu procesu

Segment stavu procesu může být umístěn kdekoli v lineárním adresovém prostoru. Není vhodné jej 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.

Tvar TSS procesoru 80386 je na obr. 4.14. Maximální délka TSS je 4 GB. Minimální je 104 slabik. V TSS jsou, stejně jako v TSS 80286 (viz str. 97), položky **zpětný ukazatel** (selektor TSS přerušeného procesu), **ukazatele zásobníků** všech úrovní oprávnění (registry pro uložení selektoru jsou 16bitové, offsetové registry jsou 32bitové), **registry procesoru** (32bitové všeobecné a 16bitové registry pro uložení selektorů segmentů) a **selektor LDT**. Navíc, oproti 80286, jsou zde uloženy tyto informace:

CR3 je obsah registru CR3 s uloženou fyzickou adresou začátku stránkového adresáře daného procesu. Hodnota se použije jenom tehdy, je-li zapnuto stránkování.

T (Trap) je bit, který, je-li nastaven na jedničku, způsobí v okamžiku přepnutí na tento proces ladicí přerušení INT 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. Každý bit této mapy odpovídá jedné 8bitové V/V bráně tak,

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. 4.14. Tvar TSS procesoru 80386

že např. brána 41 má zpřístupňující bit na adrese *offset mapy*+5 (bit č. 1).

Podle pravidel uvedených na str. 141 se mapa přístupných V/V bran používá pro kontrolování V/V instrukcí pouze tehdy, je-li CPL>IOPL. 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 přerušení INT 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*. Mezi poslední slabikou mapy a adresou, na kterou ukazuje *Limit*, **musí** být vždy slabika obsahující samé jedničky (FFh). Umístění mapy přístupných V/V bran je patrné též z obr. 4.15.

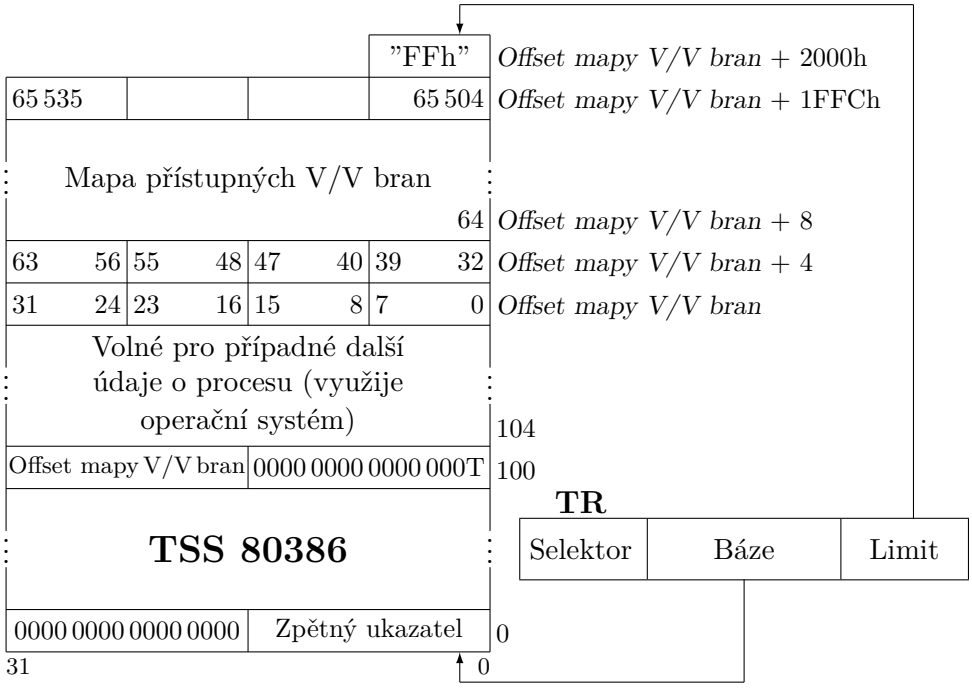
V *reálném režimu* nejsou žádné TSS, a proto V/V instrukce mohou pracovat se všemi V/V bránami. V *chráněném režimu* se mapa použije pouze tehdy, je-li CPL>IOPL. Je-li CPL≤IOPL, mohou V/V instrukce adresovat všechny V/V brány bez ohledu na obsah mapy. V režimu *virtuální 8086* je TSS a není IOPL, a proto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

4.8 Přerušení

Základní principy přerušovacího systému procesoru 80386 jsou shodné s 80286 (viz str. 103). V IDT smějí být uloženy popisovače tří typů: brána zpřístupňující TSS, brána pro maskující přerušení a brána pro nemaskující přerušení.

Formát popisovače brány pro maskující (T=0) a nemaskující (T=1) přerušení je na obr. 4.16. Od bran pro 80286 se liší bitem 3 slabiky *přístupová práva* a naplněním nejvyšších dvou slabik dalšími bity offsetu.

Chybové slovo ukládáné přerušeními 10 až 13 do zásobníku má rovněž stejný tvar jako v procesoru 80286 (viz obr. 3.25 na str. 106) pouze s tím rozdílem, že je prodlouženo na 32 bitů (horních 16 bitů je nevyužito).



Obr. 4.15. Umístění mapy přístupných V/V bran v TSS 80386

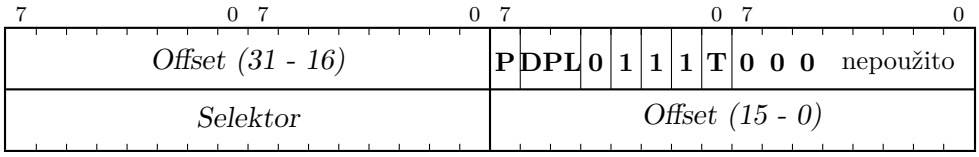
4.8.1 Rezervovaná přerušení

V procesoru 80386 jsou využity přerušovací vektory 0 až 16 pro přerušení generovaná procesorem. Přerušení 17 až 31 výrobce procesoru doporučuje nepoužívat s tím, že k volnému použití jsou určena přerušení 32 až 255.

Přerušení známá z nižších typů procesorů mají v 80386 stejný význam (viz str. 106). Rozšířena je pouze interpretace pouze přerušení číslo 1.

INT 1 **Ladicí přerušení** (Debug Exceptions)

Protože 80386 má, na rozdíl od předcházejících typů procesorů, instalován soubor ladicích nástrojů, může být toto přerušení generováno větším počtem událostí. Rutina, která INT 1 obsluhuje, zjistí příčinu čtením ladicího registru DR6. Existuje pět příčin, pro které procesor generuje INT 1:



Obr. 4.16. Formát popisovače brány přerušení v IDT

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).

Navíc procesor 80386 generuje přerušení číslo 14 (Typ: Fault, vrací chybové slovo speciálního formátu).

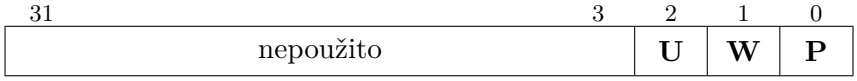
INT 14 **Výpadek stránky** (Page Fault)

Přerušení 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 přerušení 10 až 13 má zde chybové slovo jiný tvar. Indikuje, vzniklo-li přerušení 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. 4.17). V chybovém slově INT 14 jsou využity pouze tři bity nejnižších řádů:



Obr. 4.17. Formát chybového slova předávaného přerušením 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.

4.9 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* procesorů 80286 a 80386, 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 80386. 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.

4.9.1 Zapnutí a vypnutí režimu V86

Režim V86 je zapnut tehdy, je-li nastaven příznak **VM** v příznakovém registru procesoru 80386. 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 typu 80386, nikoli typu 80286 (ten obsahuje pouze 16bitový registr příznaků).

Procesor 80386 vypíná režim V86 v okamžiku přerušení:

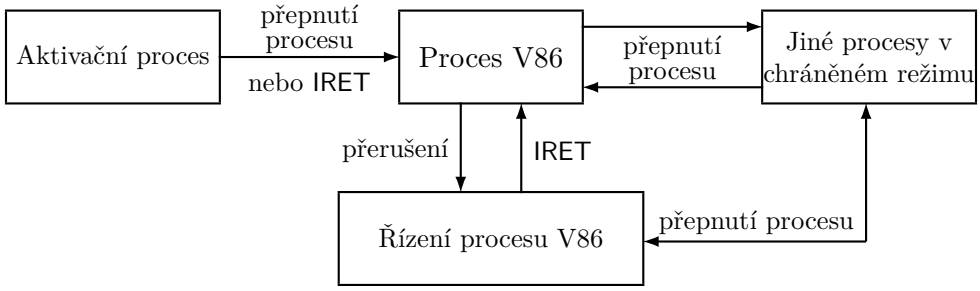
- Procesor nuluje příznak VM po uložení registru EFLAGS do zásobníku při přerušení, které aktivovalo obslužnou rutinu s CPL=0. Obslužná rutina potom probíhá normálně v chráněném režimu. Pokud by měl být přerušením aktivován instrukční segment s C=1 nebo rutina s CPL jiné úrovně než 0, generuje se chybové 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 TSS typu 80386 s VM=0 nebo TSS typu 80286.

Možné způsoby předávání řízení v operačním systému povolujícím V86 úlohy jsou na obr. 4.18.

4.9.2 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 80386 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



Obr. 4.18. Způsoby předávání a vracení řízení z režimu V86

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 přerušení INT 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 generováno přerušení „Obecná chyba ochrany“, je-li úroveň IOPL=3, lze tyto instrukce v V86 procesu používat bez omezení.

Výše uvedené instrukce jsou „hlídány“ 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 80386, protože plánovač operačního systému, který procesům přiděluje řízení po časových kvantech, nemůže pracovat. Po zachycení těchto instrukcí přerušením INT 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 v 80386 mají stejnou funkci.

4.9.3 Přerušení v režimu V86

V okamžiku přerušení procesu V86 není předáno řízení podle zvyklostí 8086 na tabulku přerušovacích vektorů uloženou od adresy 0:0, 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. 4.19.

Rutina obsluhující přerušení musí vědět, zda byla aktivována přerušením

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. 4.19. Obsah zásobníku úrovně 0 po přerušení procesu V86

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 80386 (nikoli 80286), 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 80386, proces chráněného režimu 80286 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 zá-

sobníku 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).

4.9.4 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.

4.9.5 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. 4.19) 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 odpovídají-

cím 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 přerušení „Obecná chyba ochrany“ (INT 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.

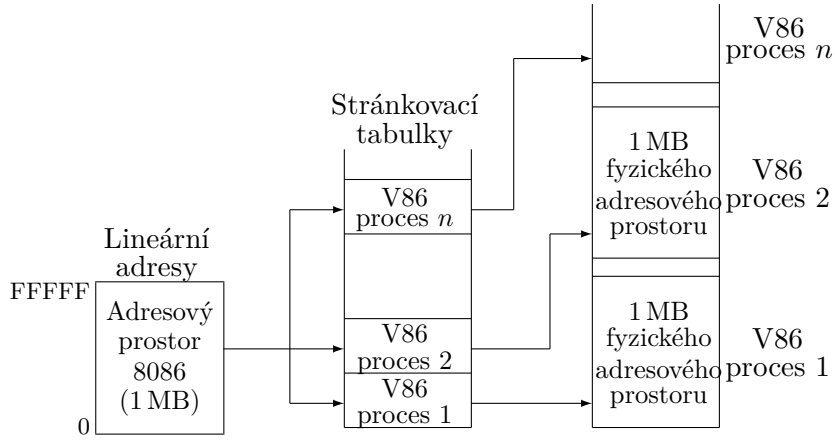
Pokud v kroku 3 nastavíme IOPL=3, není provedením instrukce IRET v kroku 5 vyvoláno přerušení „Obecné chyby ochrany“, ale je uvnitř V86 procesu proveden návrat z přerušení. Potom se kroky 6 až 8 neprovedou. Tato varianta je jednodušší na implementaci, ale nastavením IOPL=3 povolujeme obsluze přerušení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

4.9.6 Stránkování v režimu V86

Pracuje-li proces v režimu V86, je vypnut mechanismus segmentování 80386, 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. 2.2 na str. 16. 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. 4.20. 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. 4.20. 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 má procesor 80386 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 je 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. 4.21).

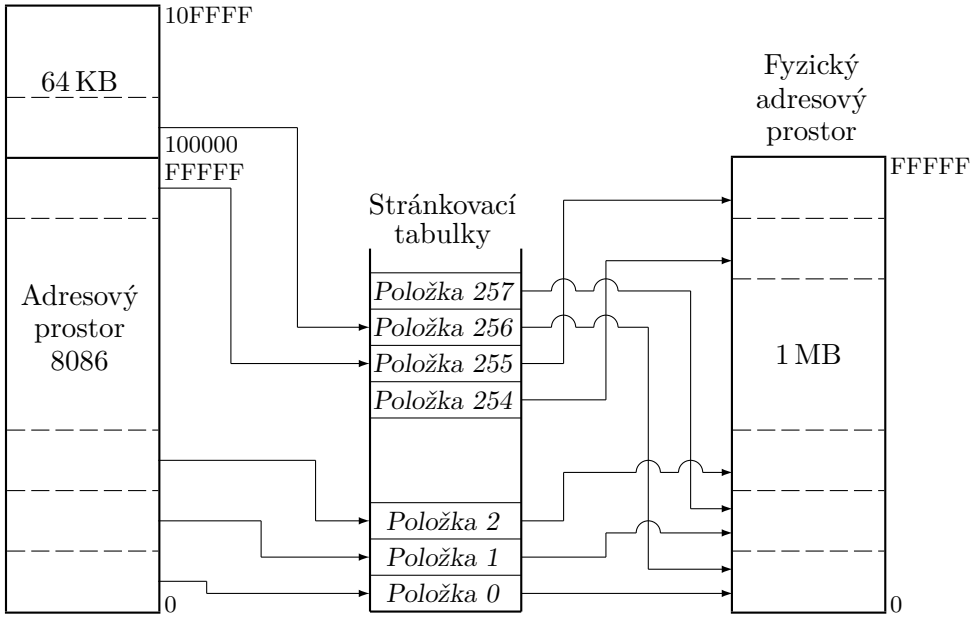
Poněvadž ve V86 režimu není v činnosti segmentování, a s tím spojené ochrany, lze použít pouze stránkové ochrany. Ochran se využije např. pro označení stránek „pouze ke čtení“, takové stránky mohou simulovat paměť ROM. Poněvadž V86 proces pracuje vždy jenom na úrovni CPL=3, jsou těmto procesům dostupné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

4.9.7 Rozdíly V86 oproti 8086

Z rysů procesoru 80386 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 80386 je 32bitový procesor. Rozdíly nejsou zásadního charakteru, a proto zpravidla nezpůsobují problémy.

Je opraveno přerušení INT 0. V 80386 se v přerušení generovaném přeplněním při dělení ukládá do zásobníku adresa *na* instrukci způsobující přerušení (DIV a IDIV). V 8086 se ukládá adresa *za* instrukci, která přerušení způsobila.

Je rozšířen rozsah IDIV**.** Zvětšením velikosti slova v 80386 je také zvětšena maximální hodnota podílu, a proto 80386 nemusí vždy generovat INT 0



Obr. 4.21. Mapování stránek 256 až 271 do stránek 0 až 15 v režimu V86

tak, jak jej 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. 163).

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 procesor 80386 v režimu V86 hlídá, a pokud je offset větší než 64 KB, generuje přerušení. Je-li takový offset spojen se segmentovým registrem DS, ES, FS nebo GS, je vyvoláno přerušení INT 13, ve spojení se SS je vyvoláno INT 12.

Je změněna instrukce PUSH SP. Instrukce v 80386 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. Procesor 80386 generuje INT 6 (Chybný operační kód) tehdy, je-li prefix použit s jinou instrukcí než s jednou z následujících:

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

Je rozšířen registr příznaků. V 80386 mají korektní hodnoty i ty bity příznakového registru, o které je příznakový registr 80386 rozšířen.

Je opraveno chybové přerušení koprocesoru. Když matematický koprocesor 8087 přerušením oznamoval chybu, ukládal procesor do zásobníku adresu chybující instrukce. V 80386 je do zásobníku ukládána adresa prvního z prefixů chybující instrukce.

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

Je omezen počet posuvů a rotací. V 80386 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. Procesor 80386 po dobu obsluhy NMI blokuje všechna přerušení včetně dalších NMI až do provedení instrukce IRET.

Je opraveno přerušení řetězcových operací. Je-li v 80386 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 80386 být delší než 15 slabik. Je-li při dekódování instrukce rozpoznáno překročení tohoto limitu (instrukce má nadbytečné prefixy), generuje se přerušení „Obecná chyba ochrany“.

Nedefinované operační kódy 8086 mohou být instrukce 80386. Poněvadž se rozšířil počet instrukcí 80386 a většinu z nich lze v režimu V86 používat, mohou mít operační kódy, které byly v 8086 nedefinovány, přiřazenu

smysluplnou instrukci.

80386 je 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 80386. Na škodu může být u těch programů přenesených z 8086, které odpočítávaly čas počtem provedených instrukcí.

4.10 Počáteční nastavení procesoru 80386

Procesor je inicializován po zjištění aktivní úrovně signálu RESET. Algoritmus inicializace se skládá z několika etap. Nejprve jsou ukončeny všechny rozpracované činnosti, tj. provádění instrukce, aktivita paměti, sběrnic, obsluha přerušení atd. Dále následuje **test vnitřní logiky** procesoru. Tento krok je nepovinný. Provede se jenom tehdy, je-li v okamžiku příchodu signálu RESET nastavena vnějším technickým vybavením aktivní úroveň signálu $\overline{\text{BUSY}}$. Vlastní test trvá 20 až 35 ms a podle sdělení firmy Intel odhalí až 90 % možných závad. Bezchybné dokončení testu signalizuje nulový obsah registru EAX. Je-li EAX nenulový, byla detekována chyba (obsah EAX však chybu neupřesňuje). Je na programátorovi, jak na tuto situaci bude reagovat. Výrobce doporučuje chybu oznámit uživateli a zastavit procesor (HALT). Pokud se test neprováděl, je obsah EAX po inicializaci nedefinován.

Během inicializace procesor nastaví registr **DH=3**, který může programátorovi sdělit, že jeho programy jsou spuštěny v procesoru 80386. Jde o novou vlastnost, kterou nižší typy neposkytují (80286 neplní DH dvojkou atd.), a tím je tato samodetekce zpochybněna.

Registr DL je naplněn číslem **verze procesoru**. Změní-li výrobce integrovaný obvod (zpravidla za účelem odstranění nějakých chyb v logice procesoru), změní i číslo, kterým plní DL. Programové vybavení potom může správně reagovat na drobné odlišnosti v chování procesoru. Intel však negarantuje, že při každé sebemenší změně logiky procesoru změní i toto číslo.

V další etapě inicializace procesor zjišťuje, je-li v počítači instalován **koprocesor** 80387 nebo 80287. Procesor využívá toho faktu, že koprocesor 80387 v okamžiku příchodu signálu RESET nastaví signál $\overline{\text{ERROR}}$ do aktivní úrovně (80287 nikoli). Podle hodnoty $\overline{\text{ERROR}}$ se nastaví bit ET v registru

CR0. Po dokončení inicializace už nelze rozlišit, je-li instalován 80387 nebo 80287 jinak, než čtením bitu ET.

Pokud v systému není instalován koprocesor, bylo v předchozím kroku nastaveno ET=0 (tj. koprocesor 80287). Zda je v systému koprocesor 80287 nebo žádný koprocesor, musí zjistit až programátor např. touto posloupností instrukcí:

```
FINIT      ; Inicializace koprocesoru.
FSTSW AX   ; Do AX se uloží stav koprocesoru po inicializaci.
CMP   AL,0 ;
JE    JeKoprocesor   ; AL=0 ... V systému je koprocesor.
; AL není nulové ... v systému není koprocesor.
```

V další etapě inicializace se provádí počáteční nastavení registrů (viz obr. 4.22). V registru EFLAGS je nastaven bit 1 (nemá však žádný význam). Všechny segmentové registry, vyjma CS, jsou vynulovány (zpřístupňují nejnižších 64 KB paměti). I když je po inicializaci nastaven reálný režim, ukazuje IDTR na segment začínající na adrese 0 mající délku 1 024 slabik.

Registr	*Obsah*	*Registr*	*Obsah*
EAX	Stav testu	CS	F000
EBX	*nedef.*	DS	0000
ECX	*nedef.*	ES	0000
EDX	DH=3, DL=verze	FS	0000
ESI	*nedef.*	GS	0000
EDI	*nedef.*	SS	0000
EBP	*nedef.*	EIP	IP=FFF0
ESP	*nedef.*	EFLAGS	00000002
GDTR	*nedef.*	CR0	0000
IDTR	báze=0, limit=3FF	DR7	0000

Obr. 4.22. Nastavení registrů po inicializaci procesoru

Instrukce, která se má jako první po inicializaci procesoru provést, je uložena na pevně dané adrese. Poněvadž tato adresa ukazuje 16 slabik před

konec paměti, což programátorovi ponechává málo místa, bude první instrukcí zpravidla instrukce JMP. Pokud by programátor nepoužil instrukci JMP a nechal by obsah IP přeplnit, došlo by v tomto okamžiku k zastavení procesoru.

Ačkoli obsah registrů CS a IP vytváří adresu 000FFFF0h, je první instrukce čtena z adresy FFFFFFF0h (4 GB-16 B), protože po inicializaci jsou adresové vodiče A_{20} až A_{31} při výběru instrukce nastaveny na jedničky. To umožňuje umístit paměť ROM se zaváděcími programy na úplný konec operační paměti a nikoli těsně pod hranici 1 MB, jak je zvykem v 8086. Automatické doplňování jedniček na vodiče A_{20} až A_{31} se týká přístupů do paměti segmentovaných přes registr CS (tj. výběru instrukcí z paměti). Odkazy na zásobník a data jsou vedeny na začátek paměti tak, jak ukazují segmentové registry. Data v posledních 64 KB paměti lze zpřístupnit tak, že v instrukci je explicitně uveden segmentový registr CS. Doplňování jedniček na adresové vodiče A_{20} až A_{31} se vypne v tom okamžiku, kdy je změněn obsah segmentového registru CS. To, v reálném režimu, lze provést instrukcemi vzdáleného skoku a volání: JMP, CALL a RET. Blízké skoky obsah CS nemění. Potom, poněvadž je zapnut reálný režim, není již adresový prostor nad 1 MB přístupný.

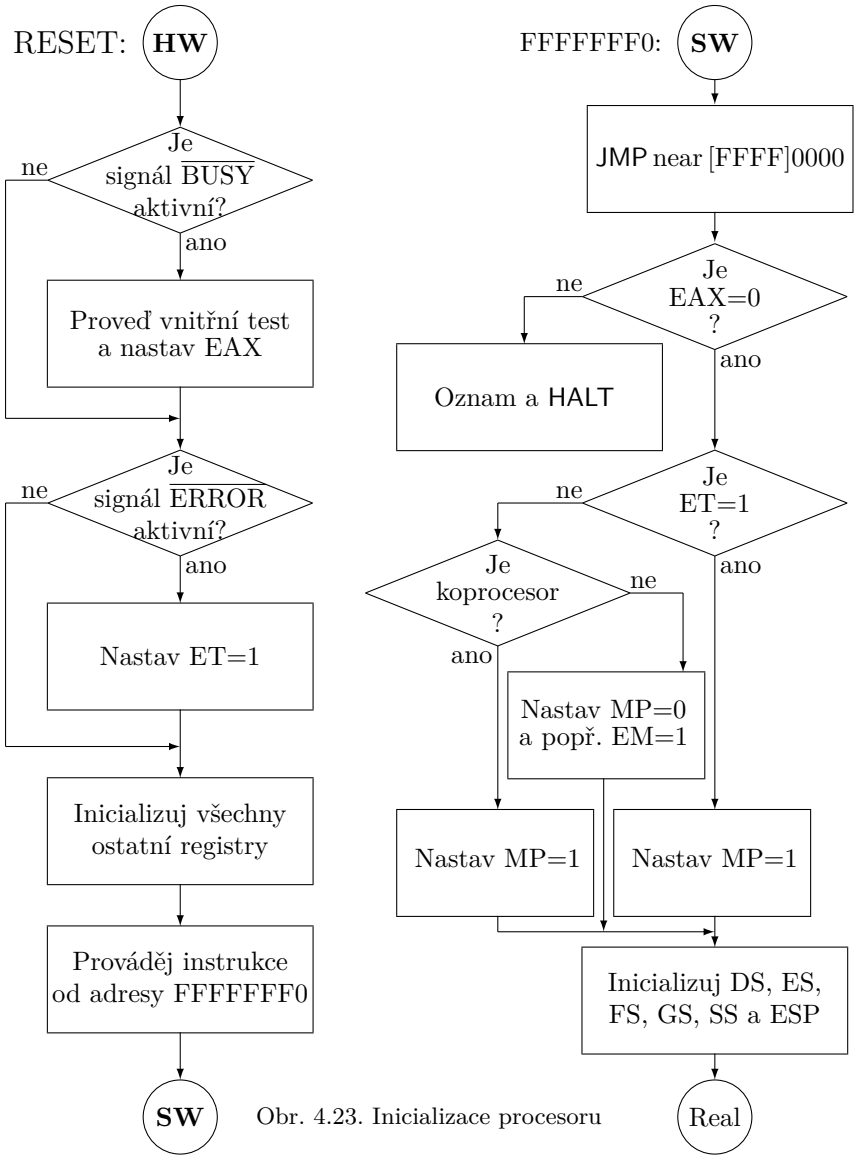
4.11 Reálný režim 80386

Podobně jako režim *virtuální 8086* má *reálný režim* oproti procesoru 8086 některá rozšíření. Může pracovat s 32bitovými všeobecnými, ladicími a řídicími registry. Má-li v reálném režimu instrukce (např. MOV) plnit 32bitový EAX místo 16bitového AX, musí být před instrukcí uveden prefix změny velikosti operandu (66h). V reálném režimu nejsou dostupné pouze registry TR (Task Register) a LDTR (Local Descriptor Table Register). V reálném režimu lze používat rovněž segmentové registry FS a GS po uvedení instrukčních prefixů 64h a 65h. V reálném režimu se nesmějí použít tyto instrukce:

VERR, VERW
LAR, LSL
LTR, STR
LLDT, SLDT
ARPL

Činnost procesoru po přijetí signálu RESET.

Doporučená činnost programového vybavení určeného pro inicializaci procesoru.



Obr. 4.23. Inicializace procesoru

Tyto instrukce nejsou procesorem v reálném režimu rozpoznány a to vede ke generování přerušení INT 6. Všechny ostatní instrukce, jako jsou manipulace s bitovými řetězci, podmíněné skoky s 16bitovým offsetem atd., lze používat bez omezení.

Systém ochran není v reálném režimu zapnut. Můžeme si však představit, že v tomto režimu se provádějí všechny operace na úrovni oprávnění 0. Tím se reálný režim liší od režimu virtuální 8086. A protože vše pracuje s nejvyšší úrovní oprávnění, nezávisí provádění V/V operací na nastavení IOPL v příznakovém registru. Stejně jako v 8086 jsou všechny V/V operace povoleny a také v tomto režimu nerozlišujeme mezi privilegovanými a neprivilegovanými instrukcemi.

4.11.1 Adresace v reálném režimu

Adresy jsou v reálném režimu vytvářeny stejným způsobem jako v procesoru 8086 *(segment×16) + offset*. Poněvadž adresová sběrnice není 20bitová, ale je 32bitová, může maximální adresa dosáhnout hodnoty 10FFEFh. Tento problém byl již diskutován v rámci režimu virtuální 8086 (na str. 163). V reálném režimu nemůže být v činnosti stránkovací jednotka, a proto zde, na rozdíl od režimu virtuální 8086, nelze mapovat stránky umístěné nad hranicí 1 MB do nejnižších adres.

V instrukcích reálného režimu lze použít prefix 67h, který mění velikost adresového operandu na 32 bitů. Užitečnost této vlastnosti je vzápětí procesorem potlačena tak, že se kontroluje, zda hodnota takového operandu nepřekročila FFFFh. Pokud ano, generuje se přerušení 13 nebo 12. Tato přerušení, stejně jako všechna přerušení v reálném režimu, neukládají do zásobníku chybové slovo. Stejně jako v 8086 se zde do zásobníku ukládají 16bitový FLAGS, CS a IP.

4.11.2 Přerušení v reálném režimu

Obsluha přerušení v režimu virtuální 8086 se významně liší od obsluhy přerušení v reálném režimu. Ve virtuální 8086 každé přerušení přepne do chráněného režimu, ve kterém podle IDT přerušení obslouží jeden z procesů. V reálném režimu se používá tabulka přerušovacích vektorů podle zvyklostí

8086 (viz str. 22) a z ní se vybere adresa rutiny, která má dané přerušení obsloužit.

Jediným rozdílem ve filozofii obsluhy přerušení v reálném režimu a v 8086 je fakt, že v reálném režimu je stále v činnosti registr IDTR. Registr však neukazuje na standardní IDT chráněného režimu, ale na tabulku přerušovacích vektorů. Implicitně tento registr obsahuje bázi 0 a limit 3FF (viz obr. 4.22). Zvětšení limitu nemá význam, protože přerušení čísla >255 se nevyskytne. Je-li limit změněn na hodnotu <3FF, nastane při výskytu přerušení, která mají obslužnou adresu za změněným limitem, přerušení 8 (toto přerušení v chráněném režimu oznamuje dvojnásobný výpadek segmentu). Je-li limit malý i na přerušení 8, procesor se zastaví.

Stejně jako limit lze změnit v registru IDTR i bázi. V tom případě procesor předpokládá, že je tabulka přerušovacích vektorů umístěna od této adresy. Tabulka může být umístěna na libovolném místě paměťového prostoru reálného režimu. Protože 8086 registr IDTR nemá a v každém případě předpokládá umístění tabulky od absolutní adresy 0:0, je toto rozšíření reálného režimu pochybné.

4.12 Přepnutí do chráněného režimu

Teoreticky se chráněný režim zapne nastavením bitu PE v registru CR0 na jedničku. Avšak prakticky je nutno předtím připravit celou řadu podmínek. Nastavení bitu PE bez předchozí přípravy vede s největší pravděpodobností k zastavení procesoru.

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 (ani 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 (nejlépe 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 tak, jak ukazovaly v reálném režimu, a to 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 (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. 4.24.

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

Obr. 4.24. Doporučené činnosti po přepnutí do chráněného režimu

4.12.1 Víceúlohové zpracování

Budou-li se v chráněném režimu přepínat procesy, musí se připravit TSS. TSS lze však 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 80386 do tohoto „inicializačního“ TSS uloží stav právě aktivního procesu.

4.12.2 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 adresovém 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.

4.13 Přepnutí do reálného režimu

Na rozdíl od 80286 dovoluje 80386 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=FFFF, 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. Po vynulování bitu PE je vhodné provést instrukci vzdáleného skoku, aby se aktualizovaly údaje.

Ihned po přepnutí do reálného režimu se musí naplnit registr IDTR správným obsahem (např. viz obr. 4.22). Doporučený postup při přepínání do reálného režimu je na obr. 4.25.

- 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. 4.25. Doporučené činnosti při přepínání do reálného režimu

4.14 Ladicí nástroje procesoru 80386

Procesor 80386 má oproti předchozím typům i oproti většině procesorů jiných firem do detailů propracované ladicí nástroje. Z procesoru 8086 známe dva ladicí prostředky: **krokovací režim** (INT 1) a **ladicí bod** (INT 3). Procesor 80286 ladicí možnosti rozšířil v chráněném režimu o kontrolu dodržování limitů segmentů, respektování typové kompatibility operací nad segmenty a dodržování pravidel systému ochran. Procesor 80386 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 80386 instalována sada ladicích registrů DR*i*.

Do těchto 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 přerušení INT 1 na str. 24 a **ladicí bod** v rámci přerušení INT 3 tamtéž.

Chráněný režim procesoru 80286 poskytuje ladicímu systému možnost kontroly limitů segmentů, typové kompatibility operací nad segmenty a dodržování pravidel přístupů k segmentům převážně prostřednictvím přerušení INT 13 a ve speciálních případech prostřednictvím přerušení INT 10 až INT 12 (viz str. 108).

4.14.1 Sledování přepínání procesů

Možnost sledovat přepínání procesů v 80386 je výhodná zvláště při ladění víceúlohových systémů. Sledování se zapne nastavením bitu **T** (Trap) v TSS (viz str. 150). Jakmile 80386 při přepnutí procesu rozpozná, že nový proces má v TSS nastaven bit T=1, vygeneruje ladicí přerušení INT 1 ještě dříve, než je provedena první instrukce nového procesu. Z toho plyne, že přerušení INT 1 je již obslouženo v kontextu nového procesu za předpokladu, že přerušení není obsluhováno vlastní branou zpřístupňující jiný TSS. Je-li ladicí přerušení obsluhováno tímto způsobem (přepnutím procesu), je logické, že takový obslužný proces nesmí mít T=1.

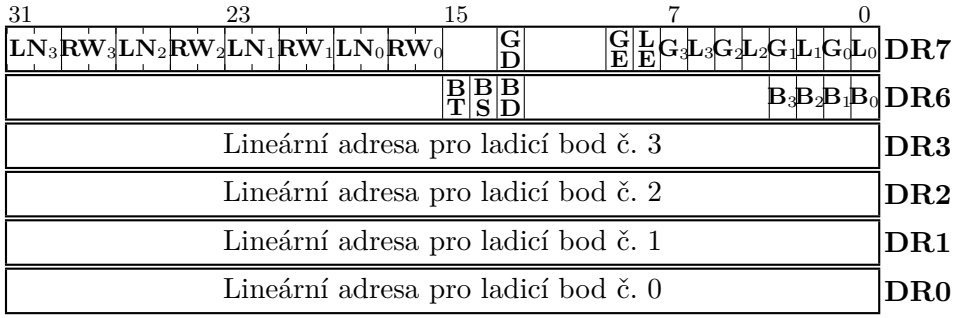
Samotné nastavení bitu T na jedničku nevyvolá žádnou aktivitu ladicích nástrojů. Jeho hodnota se čte pouze při přepnutí na tento proces. Bit T není procesorem nulován. Pokud již ladicí systém nepotřebuje sledovat aktivaci tohoto procesu, musí T vynulovat sám.

V okamžiku vygenerování INT 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.

4.14.2 Ladicí registry

Procesor 80386 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 generováno ladicí přerušení (INT 1).

Registr **DR7** je **příkazový registr** ladicího systému procesoru. Obsahuje tyto bity (viz obr. 4.26, i je v intervalu 0 až 3):



Obr. 4.26. Ladicí registry DR0 až DR3, DR6 a DR7 procesoru 80386

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 některým z registrů DR*i* vyvolá ladicí přerušení. Čtyři možné kombinace těchto dvou bitů jsou popsány na obr. 4.27.

LN$_i$ kvalifikují délku operandu, který při shodě adresy začátku tohoto operandu s některým z registrů DR0 až DR3 vyvolá ladicí přerušení. 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. 4.27.

RW	Typ operace
00	Výběr instrukce
01	Zápis dat
10	Nepoužito
11	Čtení nebo zápis dat

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

Obr. 4.27. Obsah bitů RW a LN v registru DR7

LE (Local Exact) sděluje procesoru, kdy má provádět testování shody adresy. Procesor 80386 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) 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.

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 (příp. na první z jejích prefixů). 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 o INT 1 v 80386, jehož aktivaci způsobuje pět různých událostí. Stavový registr indikuje, co bylo příčinou vyvolání tohoto ladicího přerušení. Jeho obsah procesor nenuluje, to je záležitostí programového vybavení. Pokud by ladicí systém bity stavového registru nenuloval, nebylo by možné přerušení identifikovat, protože by se příznaky kumulovaly. Jednotlivé bity (viz obr. 4.26) mají tento význam:

B$_i$ (Breakpoint i Hit) je nastaven procesorem na jedničku tehdy, bylo-li ladicí přerušení vyvoláno shodou adresy s registrem DRi.

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

BS (Break for Single-Step) je nastaven na jedničku, bylo-li ladicí přerušení vyvoláno krokovacím režimem procesoru (TF=1).

BT (Break for Task Switch) je nastaven na jedničku tehdy, bylo-li ladicí přerušení vyvoláno 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 přerušením číslo 1.

Rutina obsluhující přerušení INT 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 přerušení více podmínek splněných současně.

V takovém případě není 80386 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 systém 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 systém 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.

4.14.3 Příznak RF

Je-li přerušení INT 1 způsobeno shodou obsahu jednoho z registrů DR*i* s adresou právě dekódované instrukce, je do zásobníku uložena adresa takto označené instrukce. Po obsloužení INT 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. 4.3 na str. 134).

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 zmíněné ladicí přerušení. RF není nastaven přerušeními vyvolanými technickými prostředky, instrukcí INT a přerušeními klasifikovanými „Trap“ a „Abort“.

Je-li RF nastaven, jsou ignorována všechna přerušení ladicích prostředků procesoru. Příznak je umístěn v příznakovém registru proto, aby byl přerušením uložen do zásobníku a posléze obnoven instrukcí IRET, a tím bylo potlačeno ladicí přerušení spojené 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 přerušení ladicího systému a nejsou jím dotčena vnější přerušení ani přerušení generovaná procesorem (obecná chyba ochrany apod.).

4.14.4 Ladicí body

Nastavením lineární adresy do jednoho z registrů DR0 až DR3 a příslušných bitů do registru DR7 zapínáme sledování aktivity procesoru dotýkající se 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. 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.

Zarovnání adresy objektů delších než slabika a ignorování nižších 1 nebo 2 bitů lineární adresy v DRi dovolí 80386 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.

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.

4.14.5 Ladicí body pro datové přístupy

Aktivace ladicího bodu určeného pro sledování instrukce (označeného RW=0 a LN=0) vyvolá přerušení, které 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 přerušení typu „Trap“. Přerušení 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 80386 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.

4.14.6 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 přerušení (Fault), které vynuluje bit GD, aby bylo možné zjistit příčinu přerušení 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.

4.15 Adresovací techniky procesoru 80386

Adresovací techniky procesoru 80386 jsou v 32bitovém adresovém režimu (v instrukčním segmentu je D=1, viz str. 139, nebo je použit prefix změny velikosti adresy 67h) rozšířeny o možnost násobení indexu měřítkem. Měřítko smí nabývat hodnot 1, 2, 4 a 8. Kompletní schéma (viz též str. 31) výpočtu adresy pro 80386 je:

Vypočtená adresa = *přímá adresa + báze + (index × měřítko)*

Potom lze tabulku z obr. 2.9 na str. 31 rozšířit o kombinace uvedené na obr. 4.28.

přímá adresa + (index × měřítko)
báze + (index × měřítko)
přímá adresa + báze + (index × měřítko)

měřítko
je
1, 2, 4 nebo 8

Obr. 4.28. Rozšíření adresovacích technik 80386

Příklady instrukcí používajících tyto nové adresovací techniky zapsané v asembleru mohou být následující:

```
MOV AH,Adresa[ESI*4]
MOV AH,[EBX][EDI*4]
MOV AH,Adresa[EBP][ESI*2]
```

Dalším novým rysem 80386 (při D=1, resp. s prefixem změny velikosti adresy) je rozšířená interpretace indexových a bázových registrů:

bázovým registrem může být kterýkoli z osmi všeobecných registrů,

indexovým registrem může být kterýkoli z všeobecných registrů kromě ESP.

Implicitní segmentový registr je vybrán podle použitého bázového registru tak, že segmentový registr:

DS se použije pro adresaci báze v EAX, EBX, ECX, EDX, ESI a EDI,

SS se použije pro adresaci báze v EBP a ESP.

Obr. 4.29 shrnuje použitelné adresovací techniky v jednotlivých režimech: 16bitový adresovací režim je nastaven D=0 (nebo D=1 s prefixem změny velikosti adresy), 32bitový adresovací režim je nastaven D=1 (nebo D=0 s prefixem změny velikosti adresy). V reálném režimu (kde je vždy implicitní 16bitový režim) lze po použití prefixu změny velikosti adresy používat nové adresovací techniky, nelze však nastavit offset větší než FFFFh.

Adresace	16bitová	32bitová
Bázové registry	BX, BP	EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI
Indexové registry	SI, DI	EAX, EBX, ECX, EDX, EBP, ESI, EDI
Měřítko	*žádné*	1, 2, 4, 8

Obr. 4.29. 16 a 32bitové adresovací techniky

4.16 Rozšíření instrukcí 80386 oproti 80286

Většina rozšíření instrukcí 80386 oproti předchozím typům procesorů spočívá v umožnění práce s 32bitovými operandy. Pro takové operandy jsme dosud nedefinovali potřebné symboly:

rel32 Relativní 32bitová adresa.
ptr16:32 Absolutní přímá adresa složená ze segmentu (16b) a offsetu (32b).
r32 32bitový všeobecný registr.
imm32 32bitová přímá hodnota.
r/m32 Buď 32bitový registr, nebo offset 32bitového dvojslova v paměti.
m16:32 Nepřímá adresa umístěná v paměti, složená ze segmentu (16b) a offsetu (32b).

Operand v paměti typu *m16:32* pojmenujeme `UkSlovo`. Obsahuje 48bitový ukazatel složený ze segmentu (16b) a offsetu (32b). V asemblerech se zpravidla deklaruje pseudoinstrukcí DP (Define Pointer).

Velké množství instrukcí definovaných v předchozích typech procesorů je v 80386 rozšířeno o nové kombinace operandů. Jiné změny ve fungování instrukcí zpravidla nejsou. Proto je v následujícím přehledu uveden u každé instrukce pouze seznam nových typů operandů bez detailního vysvětlení významu, protože ten se nezměnil.

Instrukce MOV

	Instrukce	*Příklad*	*Komentář*
MOV	MOV *r/m32,r32*	`MOV DvojSlovo,EAX`	`; DvojSlovo := EAX`
	MOV *r32,r/m32*	`MOV EBX,DvojSlovo`	`; EBX := DvojSlovo`
	MOV *r/m32,imm32*	`MOV DvojSlovo,80000`	`; DvojSlovo := 80000`

Aritmetické instrukce

	Instrukce	*Příklad*	*Komentář*
ADC	ADC *r/m32,r32*	`ADC DvojSlovo,EDX`	
		`; DvojSlovo:=DvojSlovo+EDX+CF`	
	ADC *r32,r/m32*	`ADC EBX,DvojSlovo`	
		`; EBX:=EBX+DvojSlovo+CF`	
	ADC *r/m32,imm32*	`ADC EDX,80000`	`; EDX:=EDX+80000+CF`
	ADC *r/m32,imm8*	`ADC DvojSlovo,12`	
		`; DvojSlovo:=DvojSlovo+12+CF`	
ADD	*viz* ADC		
INC	INC *r/m32*	`INC DvojSlovo`	`; DvojSlovo:=DvojSlovo+1`
SBB	*viz* ADC		
SUB	*viz* ADC		
DEC	DEC *r/m32*	`DEC DvojSlovo`	`; DvojSlovo:=DvojSlovo-1`
IMUL	IMUL *r/m32*	`IMUL DvojSlovo`	`; EDX&EAX:=EAX⋆DvojSlovo`
	IMUL *r16,r/m16*	`IMUL BX,Slovo`	`; BX:=BX⋆Slovo`
	IMUL *r32,r/m32*	`IMUL EDX,DvojSlovo`	`; EDX:=EDX⋆DvojSlovo`
	IMUL *r32,r/m32,imm8*		
		`IMUL ECX,DvojSlovo,3`	
			`; ECX := DvojSlovo ⋆ 3`
	IMUL *r32,imm8*	`IMUL EBX,7`	`; EBX:=EBX⋆7`
	IMUL *r32,r/m32,imm32*		
		`IMUL ECX,DvojSlovo,80000`	
			`; ECX := DvojSlovo ⋆ 80000`
	IMUL *r16,imm16*	`IMUL BX,500`	`; BX:=BX⋆500`
	IMUL *r32,imm32*	`IMUL ECX,80000`	`; ECX:=ECX⋆80000`
MUL	MUL *r/m32*	`MUL DvojSlovo`	`; EDX&EAX := EAX⋆DvojSlovo`

IDIV	IDIV *r/m32*	`IDIV 80000`	`; EAX := EDX&EAX ÷ 80000` `; EDX := EDX&EAX` *mod* `80000`
DIV	*viz* IDIV		
NEG	NEG *r/m32*	`NEG DvojSlovo`	`; DvojSlovo := -DvojSlovo`
CMP	CMP *r/m32,r32*	`CMP DvojSlovo,ECX`	`; F := DvojSlovo - ECX`
	CMP *r32,r/m32*	`CMP EBX,DvojSlovo`	`; F := EBX - DvojSlovo`
	CMP *r/m32,imm32*	`CMP DvojSlovo,80000`	`; F := DvojSlovo - 80000`
	CMP *r/m32,imm8*	`CMP EBX,7`	`; F := EBX - 7`

Logické instrukce

	Instrukce	*Příklad*	*Komentář*
AND	AND *r/m32,r32*	`AND DvojSlovo,ECX` `; DvojSlovo := DvojSlovo ∧ ECX`	
	AND *r32,r/m32*	`AND EBX,DvojSlovo`	`; EBX := EBX ∧ DvojSlovo`
	AND *r/m32,imm32*	`AND EAX,1FFFFh`	`; EAX := EAX ∧ 1FFFFh`
	AND *r/m16,imm8*	`AND DX,7`	`; DX := DX ∧ 7`
	AND *r/m32,imm8*	`AND EBX,7`	`; EBX := EBX ∧ 7`
OR	*viz* AND		
XOR	*viz* AND		
NOT	NOT *r/m32*	`NOT DvojSlovo`	`; DvojSlovo :=` $\overline{\texttt{DvojSlovo}}$
TEST	TEST *r/m32,imm32*	`TEST DvojSlovo,0Fh`	`; F := DvojSlovo ∧ 0Fh`
	TEST *r/m32,r32*	`TEST DvojSlovo,EAX`	`; F := DvojSlovo ∧ EAX`

Rotace a posuvy

	Instrukce	*Příklad*	*Komentář*
RCL, RCR, ROL, ROR, SAL, SAR, SHR, SHL			
RCL	RCL *r/m32,*1	`RCL DvojSlovo,1`	`; Rotace DvojSlovo o 1 b.`
	RCL *r/m32,*CL	`RCL DvojSlovo,CL`	`; Rotace DvojSlovo o CL b.`
	RCL *r/m32,imm8*	`RCL DvojSlovo,5`	`; Rotace DvojSlovo o 5 b.`

Větvení programu

	Instrukce	*Příklad*	*Komentář*
JMP	JMP *rel32*	`JMP Navesti`	`; Přímý blízký skok`
			`;  EIP:=EIP+vzdál. Navesti`
	JMP *r/m32*	`JMP [DvojSlovo]`	`; Nepřímý blízký skok`
			`;  EIP := DvojSlovo`
	JMP *ptr16:32*	`JMP FAR PTR Navesti`	`; Přímý vzdálený skok`
			`;  CS:EIP:=seg:off Navesti`
	JMP *m16:32*	`JMP [UkSlovo]`	`; Nepřímý vzdálený skok`
			`;  CS:EIP := UkSlovo`
JA, JAE, JB, JBE, JC, JE, JZ, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS,			
JZ	JZ *rel16/32*	`JZ Navesti`	`; Blízký podmíněný skok`
		`; s 16 nebo 32bitovým offsetem`	
JECXZ	JECXZ *rel8*	`JECXZ Navesti`	`; Jako JCXZ, ale s ECX`
CALL	CALL *rel32*	`CALL Navesti`	`; Přímé blízké volání`
			`;  EIP:=EIP+vzdál. Navesti`
	CALL *r/m32*	`CALL [DvojSlovo]`	`; Nepřímé blízké volání`
			`;  EIP := DvojSlovo`
	CALL *ptr16:32*	`CALL FAR PTR Navesti`	`; Přímé vzdálené volání`
			`;  CS:EIP:=seg:off Navesti`
	CALL *m16:32*	`CALL [UkSlovo]`	`; Nepřímé vzdálené volání`
			`;  CS:EIP := UkSlovo`

Zásobník a příznakový registr

	Instrukce	*Příklad*	*Komentář*
PUSH	PUSH *r32*	`PUSH EAX`	`; Uložení EAX`
	PUSH *imm32*	`PUSH 0EEEEEh`	`; Uložení 0EEEEEh`
	PUSH FS	`PUSH FS`	`; Uložení FS`
	PUSH GS	`PUSH GS`	`; Uložení GS`
POP	POP *r32*	`POP EAX`	`; Výběr EAX`
	POP FS	`POP FS`	`; Výběr FS`
	POP GS	`POP GS`	`; Výběr GS`
PUSHFD	PUSHFD	`PUSHFD`	`; Uložení EFLAGS`
POPFD	POPFD	`POPFD`	`; Výběr EFLAGS`
PUSHAD	PUSHAD	`PUSHAD`	
		`; Uložení EAX, ECX, EDX, EBX, ESP, EBP, ESI a EDI`	
POPAD	POPAD	`POPAD`	
		`; Výběr EDI, ESI, EBP, ESP, EBX, EDX, ECX a EAX`	

Přerušovací systém

	Instrukce	*Příklad*	*Komentář*
IRETD	IRETD	`IRETD`	`; Návrat z přerušení`
			`; s 32bitovým zásobníkem`

Cykly

	Instrukce	*Příklad*	*Komentář*
LOOP	LOOP *rel8*	`LOOP Navesti`	`; Použije CX nebo ECX podle`
LOOP*cond*		`; implicitní velikosti operandu nebo prefixu změny velikosti`	

Ovládání V/V

	Instrukce	*Příklad*	*Komentář*
IN	IN EAX,*imm8*	`IN EAX,61h`	`; Přenos dvojslova`
	IN EAX,DX	`IN EAX,DX`	
OUT	OUT *imm8*,EAX	`OUT 20h,EAX`	`; Přenos dvojslova`
	OUT DX,EAX	`OUT DX,EAX`	

Přesuny dat

	Instrukce	*Příklad*	*Komentář*
XCHG	XCHG *r/m32,r32*	`XCHG DvojSlovo,EAX`	`; Záměna DvojSlovo a EAX`
	XCHG *r32,r/m32*	`XCHG EBX,DvojSlovo`	`; Záměna EBX a DvojSlovo`
LEA	LEA *r32,m*	`LEA EAX,DvojSlovo`	`; EAX := OFFSET DvojSlovo`
LDS	LDS *r32,m16:32*	`LDS EBX,UkSlovo`	`; DS:EBX := UkSlovo`
LES	LES *r32,m16:32*	`LES ESI,UkSlovo`	`; ES:ESI := UkSlovo`
LFS	LFS *r32,m16:32*	`LFS EDI,UkSlovo`	`; FS:EDI := UkSlovo`
LGS	LGS *r32,m16:32*	`LGS EBX,UkSlovo`	`; GS:EBX := UkSlovo`
LSS	LSS *r32,m16:32*	`LSS EBP,UkSlovo`	`; SS:EBP := UkSlovo`
LSL	LSL *r32,r/m32*	`LSL EAX,DvojSlovo`	`; Naplnění EAX limitem`
	Při použití 32bitového cílového registru je hodnota limitu vždy ve slabikách.		

LAR	LAR *r32,r/m32*	`LAR EAX,DvojSlovo`	`; EAX := přístupová práva` `;  ∧ 00FxFF00h`

Řetězcové instrukce

Instrukce	*Příklad*	*Komentář*

CMPSD, SCASD, MOVSD, LODSD, STOSD, INSD, OUTSD
jsou varianty řetězcových instrukcí pracujících s 32bitovými dvojslovy.

4.17 Nové instrukce procesoru 80386

BSF — *Bit Scan Forward*
BSR — *Bit Scan Reverse*

O	D	I	T	S	Z	A	P	C
					*			

POPIS:

Instrukce BSF a BSR prohlížejí zdrojový operand po bitech počínaje bitem 0 (BSF) nebo nejvyšším bitem (BSR) a hledají první výskyt nenulového bitu. Instrukce BSF prohlíží bity v objektu směrem nahoru a BSR směrem dolů.

Pokud jsou všechny bity zdrojového operandu nulové, je nastaven příznak ZF na nulu. V opačném případě je ZF=1 a cílový operand obsahuje číslo prvního nenulového bitu ve zdrojovém operandu.

SYNTAX: ⌜ BSF *cílový_operand,zdrojový_operand*
BSR *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
BSF *r16,r/m16*	`BSF BX,Slovo`	`; Prohledávání Slovo 0 --> 15`

```
BSF r32,r/m32 BSF EDX,DvojSlovo ; Prohledávání DvojSlovo 0 --> 31
BSR r16,r/m16 BSR AX,BX         ; Prohledávání BX 15 --> 0
BSR r32,r/m32 BSR ECX,EDX       ; Prohledávání EDX 31 --> 0
```

BT*x* — *Bit Test (and Complement/Reset/Set)*

O	D	I	T	S	Z	A	P	C
								*

POPIS:

Skupina BT*x* zahrnuje čtyři instrukce BT, BTC, BTR a BTS. Instrukce BT kopíruje jeden bit cílového operandu do příznaku CF. Číslo bitu je určeno zdrojovým operandem (nejnižší bit je označen 0).

Instrukce BTC pak navíc hodnotu bitu *cílového_operandu* invertuje, instrukce BTR vynuluje a BTS nastaví na 1.

SYNTAX: ⌜ BT*x* *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
BT *r/m16,r16*	`BT Slovo,AX`	`; CF := Slovo(bit číslo AX)`
BT *r/m32,r32*	`BT DvojSlovo,EAX`	`; CF := DvojSlovo(bit číslo EAX)`
BT *r/m16,imm8*	`BT AX,12`	`; CF := AX(bit číslo 12)`
BT *r/m32,imm8*	`BT EAX,25`	`; CF := EAX(bit číslo 25)`

CDQ — *Convert Doubleword to Quad-Word*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce CDQ převádí 32bitové číslo se znaménkem uložené v EAX na 64bitové číslo se znaménkem uložené do dvojice registrů EDX&EAX. Znaménkový bit registru EAX se při konverzi rozšíří do všech bitů registru EDX.

SYNTAX: ⌜ CDQ

Instrukce	*Příklad*	*Komentář*
CDQ	`CDQ`	`; Převede obsah EAX do EDX&EAX se zachováním znaménka`

CWDE — *Convert Word to Doubleword*

O	D	I	T	S	Z	A	P	C

POPIS:

Instrukce CWDE převádí 16bitové číslo se znaménkem uložené v registru AX na 32bitové číslo se znaménkem uložené v EAX. Převod se provede rozšířením znaménkového bitu registru AX do 16 horních bitů registru EAX.

SYNTAX: ⌜ CWDE

Instrukce	*Příklad*	*Komentář*
CWDE	`CWDE`	`; Převede obsah AX do EAX se zachováním znaménka`

MOV — *Move to/from Special Registers*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Tato varianta instrukce MOV slouží ke čtení nebo plnění speciálních registrů procesoru. Těmi jsou řídicí registry CR0, CR2, CR3, testovací registry TR6, TR7 a ladicí registry DR0, DR1, DR2, DR3, DR6 a DR7. V této variantě instrukce MOV je povoleno použít pouze 32bitové všeobecné registry.

SYNTAX: ⌜ MOV *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
MOV *r32,speciální_registr*	`MOV EAX,CR0`	`; EAX := CR0`
MOV *speciální_registr,r32*	`MOV DR7,EDX`	`; DR7 := EDX`

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 zdrojovém operandu jeho znaménkovým bitem (v instrukci MOVSZ nulou) na velikost cílového operandu a tuto hodnotu uloží do cílového operandu.

SYNTAX: ⌜ MOVSX *cílový_operand,zdrojový_operand*
MOVSZ *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
MOVSX *r16,r/m8*	`MOVSX AX,Slab`	`; Převede Slab do AX`
MOVSX *r32,r/m8*	`MOVSX EAX,Slab`	`; Převede Slab do EAX`
MOVSX *r32,r/m16*	`MOVSX ECX,Slovo`	`; Převede Slovo do ECX`

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 kódem instrukce SET*cond*, nastaví se operand na hodnotu 1. Nevyhovuje-li obsah registru příznaků podmínce, je operand nastaven na hodnotu 0.

SYNTAX: ⌜ SET*cond operand*

Podmínky, které lze uvést v instrukci SET*cond*, jsou totožné s podmínkami užívanými instrukcí J*cond*. Přesto je pro přesnost 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)

SHLD
SHRD

Double Precision Shift Left
Double Precision Shift Right

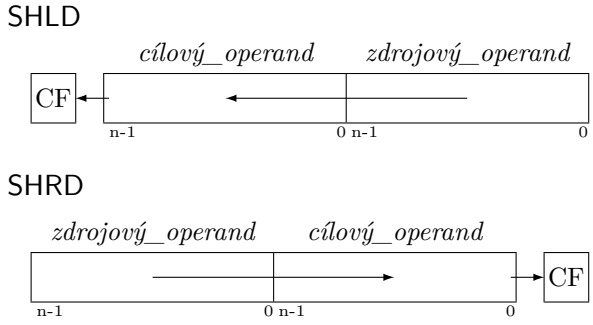
O	D	I	T	S	Z	A	P	C
?				*	*	?	*	*

POPIS:

Instrukce SHLD (SHRD) provádí operaci logického posuvu doleva (doprava) hodnoty, která vznikne spojením zdrojového a cílového operandu, o počet bitů určený operandem *počet*. V operandu *počet* je významných pouze nejnižších 5 bitů (maximální hodnota je tedy 31).

Algoritmus posuvu je patrný z obr. 4.30. Zdrojový operand zůstává po provedení instrukce beze změny. Výsledek je uložen v cílovém operandu. Od instrukcí SHL (SHR) se tyto liší tím, že do posouvaného objektu nevstupují nuly, ale bity zadané zdrojovým operandem.

Na místě operandu *počet* smí být přímá hodnota nebo registr CL.



Obr. 4.30. Algoritmus instrukcí SHLD a SHRD

SYNTAX: ⌜ SHLD *cílový_operand,zdrojový_operand,počet*
SHRD *cílový_operand,zdrojový_operand,počet*

Instrukce	*Příklad*	*Komentář*
SHLD *r/m16,r16,imm8*	`SHLD Slovo,AX,12`	`; Posuv Slovo o 12 bitů`
		`; vlevo s doplněním z AX`
SHLD *r/m32,r32,imm8*	`SHLD DvojSlovo,EAX,4`	`; Posuv DvojSlovo`

```
SHLD r/m16,r16,CL     SHLD Slovo,AX,CL        ; Posuv Slovo o CL bitů
SHLD r/m32,r32,CL     SHLD DvojSlovo,EAX,CL   ; Posuv DvojSlovo
```

4.18 Procesor Intel 80386SX

K procesoru 80386 byla vyvinuta jeho levnější varianta 80386SX, která je programově plně slučitelná s 80386. Tento fakt vyplývá i z toho, že programově se typ procesoru (jde-li o 80386 nebo 80386SX) detekuje jenom podle drobných chybiček v logice toho kterého typu. Jedna z možností je uvedena v příkladech zdrojových programů.

Procesor 80386SX má vnější 16bitovou datovou sběrnici a 24bitovou adresovou sběrnici. Srovnáváme-li výkon 80386 a 80386SX, musíme přihlédnout k tomu, jaké programové vybavení v procesorech provozujeme. Máme-li procesory 80386 a 80386SX pracující na stejné frekvenci a provozujeme-li 16bitové programy, budou výkony téměř stejné. Provozujeme-li 32bitové programy, bude výkon 80386SX nižší, protože 32bitová data musí procesor přenášet ve dvou krocích.

Adresová sběrnice šířky 24 bitů dovoluje připojit fyzickou paměť kapacity maximálně 16 MB (na rozdíl od 4 GB v 80386).

Po inicializaci procesoru 80386SX je registr DH naplněn hodnotou 23h (viz též str. 166).

K matematickému koprocesoru 80387 byla vyprojektována jeho vnější 16bitová varianta 80387SX. Tento koprocesor je určen ke spolupráci s procesorem 80386SX.

5 Intel 80486

Procesor 80486 má 32bitovou architekturu. Častěji než 80486 bývá označován ochrannou známkou **i486**. Na jednom čipu je spolu s procesorem integrována jednotka správy paměti, jednotka operací v pohyblivé řádové čárce (Floating-Point Unit) a jednotka rychlé vyrovnávací paměti (Cache). Jednotka operací v pohyblivé řádové čárce je vlastně to, co byl matematický koprocesor 80387 pro procesor 80386. Procesor má všechny rysy, které měl 80386, a je doplněn novými technologiemi pro zvýšení výkonu.

V roce 1991 byla uvedena na trh i levnější varianta procesoru 80486, nazývaná 80486SX nebo též **486SX**. Od 80486 se liší tím, že nemá integrovánu jednotku pohyblivé řádové čárky. K tomu slouží obvod nazývaný 487SX.

Všechny programy vytvořené pro 80386, i ty, které pracují s jednotkou správy paměti (používající segmentování a stránkování) a s koprocesorem 80387, budou v 80486 funkční bez jakýchkoliv zásahů.

Poněvadž se oproti 80386 nezměnila jednotka správy paměti, je v procesoru 80486 stejná kapacita fyzické paměti (4 GB) a virtuální paměti (64 TB). Nezměnily se pracovní režimy ani postupy a instrukce ovládající výpočty v pohyblivé řádové čárce. Proto se v dalším textu omezíme jen na výklad toho, co je v 80486 jiné než v 80386.

5.1 Vyrovnávací paměť

Pro zvýšení výkonu 80486 byly do procesoru implementovány některé rysy z architektury RISC (neprojevily se však na redukci instrukčního souboru), byla prodloužena fronta předvybraných instrukcí z 16 na 32 slabik a byla na čip spolu s procesorem integrována jednotka rychlé vyrovnávací paměti o kapacitě 8 KB, která redukuje opakující se přístupy k operační paměti.

Vyrovnávací paměť (dále též VP) je dvojího typu. **Interní vyrovnávací paměť** (též **IVP**) je integrována na čipu spolu s procesorem a **externí**

vyrovnávací paměť, která může být v počítači osazena, je tvořena dodatečnými obvody. Poněvadž jsou vnější vyrovnávací paměti různých typů a různě se ovládají, nebudeme se o nich zmiňovat.

Vyrovnávací paměť je zapojena mezi procesorem a fyzickou pamětí. Ve srovnání s fyzickou pamětí má nižší kapacitu a vyšší rychlost. Ve VP jsou uloženy obsahy tolika posledně zpřístupňovaných paměťových objektů, jaká je kapacita VP. Předpokládá se, že tyto objekty bude procesor v krátkém časovém intervalu opakovaně číst a tím, že se jejich obsah předá z VP, se zkrátí vybavovací doba.

Normální režim IVP je takový, že při čtení z fyzické paměti se údaj opíše do jedné z položek IVP, které se přiřadí adresa čteného objektu. Před každým čtením paměti se zkoumá adresa, neshoduje-li se s adresou přiřazenou některé položce IVP. Pokud ano, předá se obsah této položky. Pokud ne, přečte se obsah adresy z fyzické paměti, uloží se do jedné z položek IVP (položce se přiřadí adresa objektu, který obsahuje) a hodnota se předá procesoru. Zápis se vždy uskutečňuje do fyzické paměti s tím, že se zkoumá, je-li zapisovaný objekt uložen v jedné z položek IVP. Pokud ano, zapíše se i do IVP (aktualizuje se obsah položky). Pokud ne, zapíše se pouze do fyzické paměti.

5.2 Příznakový registr 80486

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	0	0	0	**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. 5.1. Příznakový registr EFLAGS procesoru 80486

Příznakový registr je doplněn o jeden bit umístěný do horních 16 bitů:

AC (Alignment Check) 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 přerušení INT 17. Stejně tak

bude generováno přerušení 17 při pokusu o čtení nebo zápis 32bitového dvojslova na adresu, která není dělitelná čtyřmi. Tabulka na obr. 5.2 shrnuje, kterými čísly musí být dělitelná adresa uložení objektu, aby nebylo při AC=1 generováno přerušení.

Objekt	*Adresa musí být dělitelná*
Slovo	2
Dvojslovo	4
Reálné číslo v jednoduché přesnosti	4
Reálné číslo v dvojnásobné přesnosti	8
Reálné číslo v rozšířené přesnosti	8
Selektor	2
48bitový ukazatel (16b segment, 32b offset)	4
32bitový offset	4
Segment v 32bitovém tvaru	2
48bitový pseudo-popisovač	4
Bitový řetězec	4

Obr. 5.2. Hranice zarovnání objektů v paměti

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 přerušení INT 17 není v žádném případě generováno.

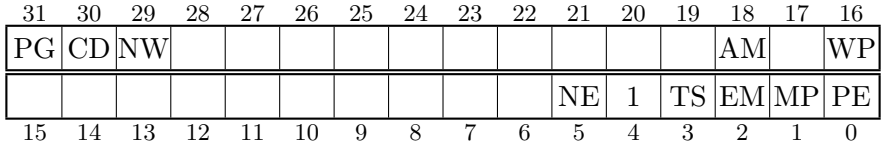
Přerušení INT 17 předává 32bitové chybové slovo s nulovým obsahem.

5.3 Řídicí registry CR*i* procesoru 80486

Řídicí registry jsou, stejně jako v 80386, definovány tři: CR0, CR2 a CR3.

V řídicím registru CR0 procesoru 80486 je pět nových bitů: CD, NW, AM, WP a NE. Bit, který byl v 80386 označen ET, má zde trvale hodnotu 1.

CD (Cache Disable) zapíná nebo vypíná interní vyrovnávací paměť. Je-li CD=1, je IVP vypnuta tak, že položky, které při čtení nebyly ve vy-



Obr. 5.3. Řídicí registr CR0 procesoru 80486

rovnávací 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.

NW (Not Write-Through) je-li nulový, potom se všechny zápisy do paměti, jejichž položka je v IVP, 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, má-li adresa zapisovaného objektu svoji položku v IVP. Údaj se ukládá pouze do fyzické paměti.

Po inicializaci procesoru signálem RESET je nastaveno NW=1.

AM (Alignment Mask) maskuje náhodné nastavení příznaku AC v příznakovém registru. 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 80386 by mohly do tohoto příznaku zavádět různé nedefinované hodnoty. Nastavením AC=0 zabráníme výskytům přerušení INT 17.

WP (Write Protect) je-li jedničkový, zakazuje zápis do stránek označených W=0 i procesům na úrovni oprávnění CPL<3. Procesor 80386, bylo-li W=0, zakazoval zápis pouze procesu s CPL=3 a procesy s CPL<3 měly zápis povolen. Kompatibilita 80486 s 80386 je zachována při WP=0. Je-li WP=1, nemůže do stránky označené W=0 zapisovat žádný proces.

NE (Numerics Exception) sděluje, jak se mají procesoru 80486 oznamovat chyby zjištěné v jednotce pohyblivé řádové čárky. Je-li NE=0, budou se

oznamovat vnějším přerušením, je-li NE=1, bude se generovat přerušení INT 16. Stav NE=0 je kompatibilní s předcházejícími typy koprocesorů a operačním systémem MS-DOS, který chyby přijímal přes vnější přerušení INT 13.

Všechna přerušení, která generuje jednotka operací v pohyblivé řádové čárce po detekování chyby při výpočtu, nastaví signál $\overline{\text{FERR}}$. Je-li NE=0 a signál $\overline{\text{IGNNE}}$ je aktivní, potom je chyba ignorována. Při NE=1 se signál $\overline{\text{IGNNE}}$ neuplatňuje.

Registr **CR2** má nezměněný význam jako v 80386. Do registru **CR3** byly přidány dva významné bity (viz obr. 5.4).

Obr. 5.4. Řídicí registr CR3 procesoru 80486

Bity **PWT** (Page Write-Through) a **PCD** (Page Cache Disable) slouží k řízení vyrovnávacích pamětí (viz odstavec „Stránkování“). 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.

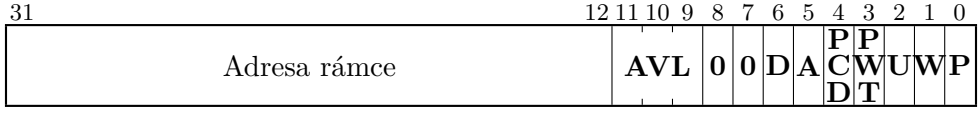
5.4 Stránkování

Mechanismus stránkování 80486 je rozšířen o několik rysů. Nová v procesoru 80486 je možnost zakázat, nastavením bitu WP v registru CR0, zápis do stránek označených W=0 i procesům s CPL<3. Pokus o provedení nepovoleného zápisu vyvolá přerušení INT 14. Je-li WP=0, mohou procesy na úrovni CPL<3 do těchto stránek zapisovat stejně jako v 80386. Vyhodnocení stránkové ochrany v procesoru 80486 je patrné z tabulky na obr. 5.5.

Specifikátor stránkového adresáře a stránkové tabulky obsahuje dva nové bity (viz obr. 5.6) PCD a PWT řídicí vyrovnávací paměť.

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. 5.5. Kombinace bitů stránkové ochrany v 80486



Obr. 5.6. Tvar specifikátoru stránkového adresáře a stránkové tabulky 80486

PWT (Page Write-Through) určuje způsob práce externí vyrovnávací paměti. 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 vyprázdňování VP.

Hodnota PWT se vztahuje pouze na externí VP. Interní VP používá techniky „Write-Through“ a její činnost nelze ovlivnit hodnotou bitu PWT.

PCD (Page Cache Disable) vypíná činnost interní VP. 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, přenáší 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.

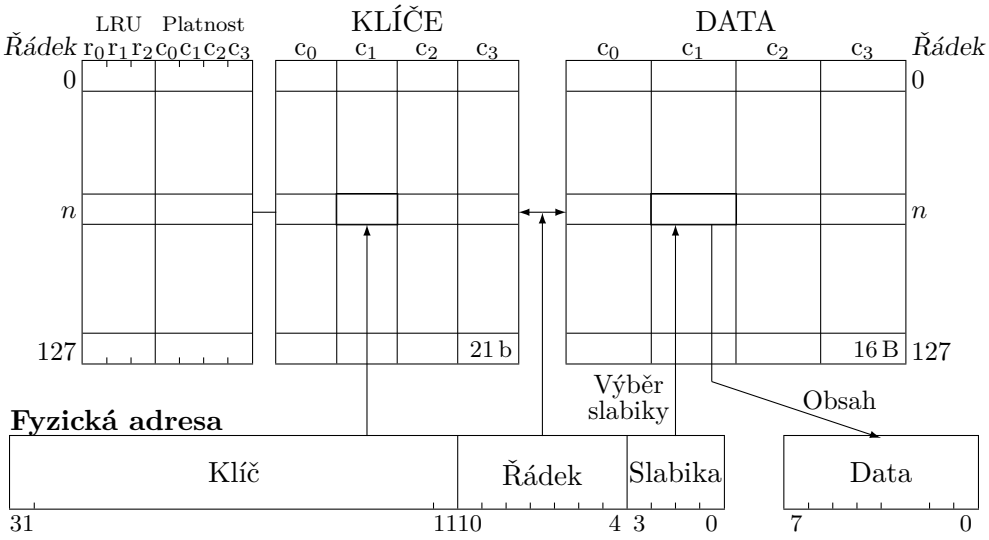
5.5 Interní vyrovnávací paměť

Jedním z prostředků, kterými se dociluje vysokého výkonu 80486, je 8 KB interní vyrovnávací paměť (IVP). Tato paměť je integrována spolu s procesorem do jednoho integrovaného obvodu. Její přítomnost nenarušuje programovou kompatibilitu nižších typů procesorů s 80486. Firma Intel prohlašuje, že IVP je vyprojektována tak, aby při aplikacích pod operačním systémem MS-DOS bylo 96 % úspěšných prohledávání IVP a pod operačními systémy Unix a OS/2 byla úspěšnost 92 %. Takové drastické snížení počtu čtecích operací na vnější datové sběrnici výrazně zvyšuje výkon procesoru.

5.5.1 Organizace IVP

IVP procesoru 80486 je společná pro data i pro instrukce. Objekty umístěné v IVP jsou označeny svými fyzickými adresami.

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. 5.7. 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.



Obr. 5.7. Struktura IVP 80486

5.5.2 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 INT 13 (chybové slovo 0). Režim bude možná využit vyššími typy procesorů k implementaci IVP typu „Write-Back“.

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. 5.8.

		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. 5.8. Kombinace bitů CD, NW a jejich funkce

5.5.3 Plnění IVP

Do IVP může být umístěna libovolná část fyzického adresovacího prostoru 80486. 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 16 slabik. Adresa první slabiky takového bloku je dělitelná 16.

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.

Každé fyzické adrese je bity 4 až 10 jednoznačně přiřazen řádek IVP (viz obr. 5.7). Má-li se IVP naplnit obsahem této fyzické adresy, testuje se, je-li některý z klíčů 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 alespoň 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. 5.9.

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. 5.10.

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.

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. 5.9. Nastavování rozhodovacích bitů pseudo-LRU

5.5.4 Vyprázdnění IVP

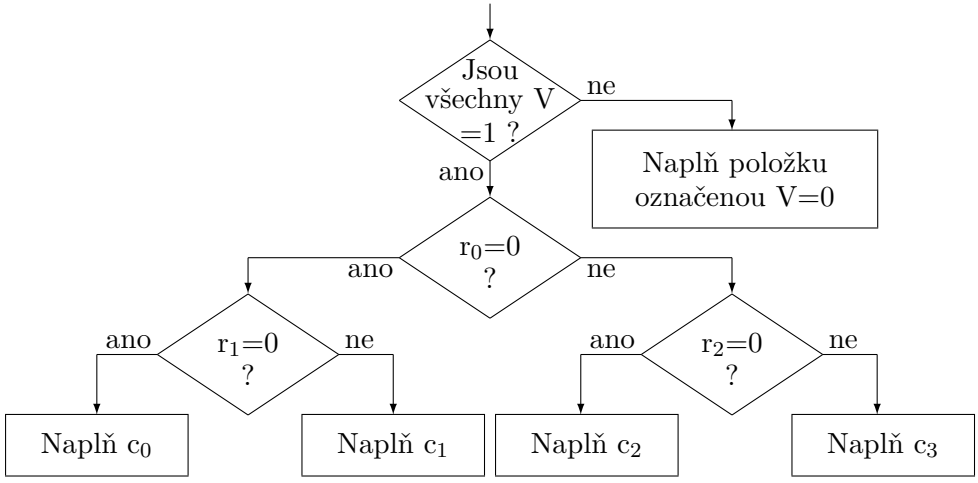
Vyprázdnění IVP je zneplatnění celého obsahu IVP. Protože IVP pracuje metodou „Write-Through“, není nutné při vyprazdňování opisovat obsah IVP do paměti. Vyprázdněním jsou vynulovány bity V všech klíčů IVP. Vnější technické vybavení má pro vyprázdnění IVP k dispozici signál $\overline{\text{FLUSH}}$.

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. Protože IVP pracuje metodou „Write-Through“, mají tyto instrukce, vzhledem k IVP v 80486, stejnou funkci. Ve vyšších typech procesorů tomu může být jinak.

5.5.5 Testování IVP

Pro testování IVP, podobně jako pro testování TLB, je v procesoru 80486 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. 5.11. 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 16 B 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



Obr. 5.10. Výběr nejdéle nepoužité položky algoritmem pseudo-LRU

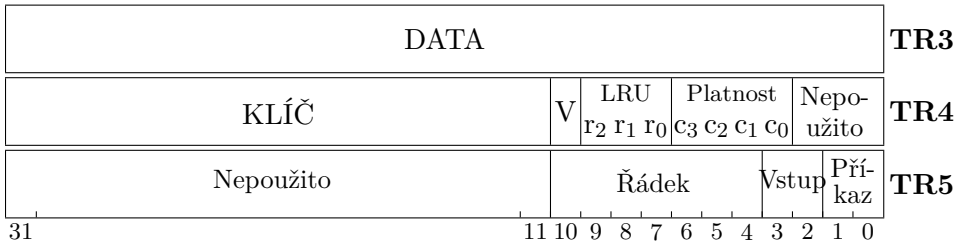
čtvrtina datové části je právě plněna nebo čtena z nebo do pomocného 16 B datového registru.

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.

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. 5.7), kam se má obsah pomocného datového registru zapsat nebo odkud se má přečíst.

Testovací zápis do IVP se provádí ve dvou krocích. Nejprve musí programátor naplnit 16 B 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



Obr. 5.11. Testovací registry TR3, TR4 a TR5

„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. 5.12. Řídicí bity testovacího registru TR5

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. 5.13.

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 16 B 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. 5.13.

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. 5.13. Příklad testovacího plnění a čtení IVP

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.

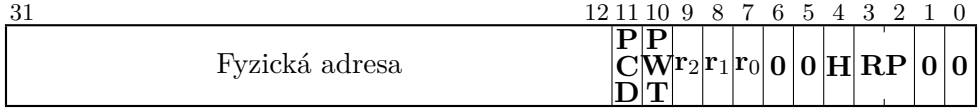
Vyprázdnění IVP příkazem 3 zapsaným do TR5 se liší od vyprázdnění IVP instrukcemi INVD a WBINVD tím, že operace vyprázdnění není signa-

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

5.6 TLB

Paměť obsahující stránkovací jednotkou posledně transformované fyzické adresy (TLB) byla v 80486 rozšířena ve dvou směrech. Za prvé, aby spolu s adresami a dalšími informacemi o stránce byly uloženy i bity PCD a PWT. Za druhé, aby byl 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 byl popsán na str. 208. Schéma TLB 80486 je shodné se schématem TLB 80386 (viz obr. 4.12 na str. 146) s tou výjimkou, že byly pro každý řádek přidány 3 rozhodovací bity r_0, r_1 a r_2 (podobně jako na obr. 5.7). TLB 80486 má stále 8 řádků a 4 cesty.

Oproti 80386 je změněn datový testovací registr **TR7**. Byly přidány bity PCD, PWT (popis viz str. 202) a 3 rozhodovací bity LRU označené r_0, r_1 a r_2 (viz obr. 5.14).



Obr. 5.14. Testovací registr TR7 procesoru 80486

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 80386 má bit význam i při zápisu (v 80386 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 80486 zavedena instrukce INVLPG (Invalidate TLB Entry), která zruší (nastaví

V=0) položku, jejíž klíč odpovídá operandu.

Na rozdíl od 80386 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.

5.7 Ladicí nástroje 80486

Procesor 80486, stejně jako 80386, disponuje čtyřmi registry pro uložení lineární adresy ladicího bodu DR0 až DR3, příkazovým registrem DR7 a stavovým registrem DR6.

Vnitřní mechanismus procesoru 80486 pro detekování ladicích bodů, jejichž lineární adresy jsou uloženy v ladicích registrech, je jiný než v procesoru 80386. Nová metoda rozpoznává ladicí body vždy bez ohledu na nastavení bitů GE a LE.

5.8 Rezervovaná přerušení 80486

Seznam rezervovaných přerušení byl oproti 80386 změněn na dvou místech:

1. nemůže nastat přerušení INT 9 (v 80286 a 80386 signalizovalo překročení segmentu koprocesorem),
2. je doplněn o přerušení INT 17 (Typ: Fault, vrací chybové slovo s nulovým obsahem).

INT 17 **Nezarovnaný přístup** (Unaligned Memory Access)

Přerušení je vyvoláno při současném splnění tří 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.

3. Procesor rozpoznal pokus o přístup k operandu v paměti, který nezačíná na adrese dělitelné délkou zpřístupňovaného operandu. Tabulka délek operandů je na obr. 5.2 na str. 199.

5.9 Jednotka operací v pohyblivé řádové čárce

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 80387 pro procesor 80386. Programové ovládání FPU je totožné s ovládáním 80387. Proto programy, které 80387 používaly, budou pracovat s procesorem 80486 bez jakýchkoliv změn.

5.9.1 Typy dat zpracovávaných FPU

FPU pracuje s osmi různými typy dat (viz též obr. 5.15). První dva typy jsou shodné s typy, které používá 80486 bez FPU.

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

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.

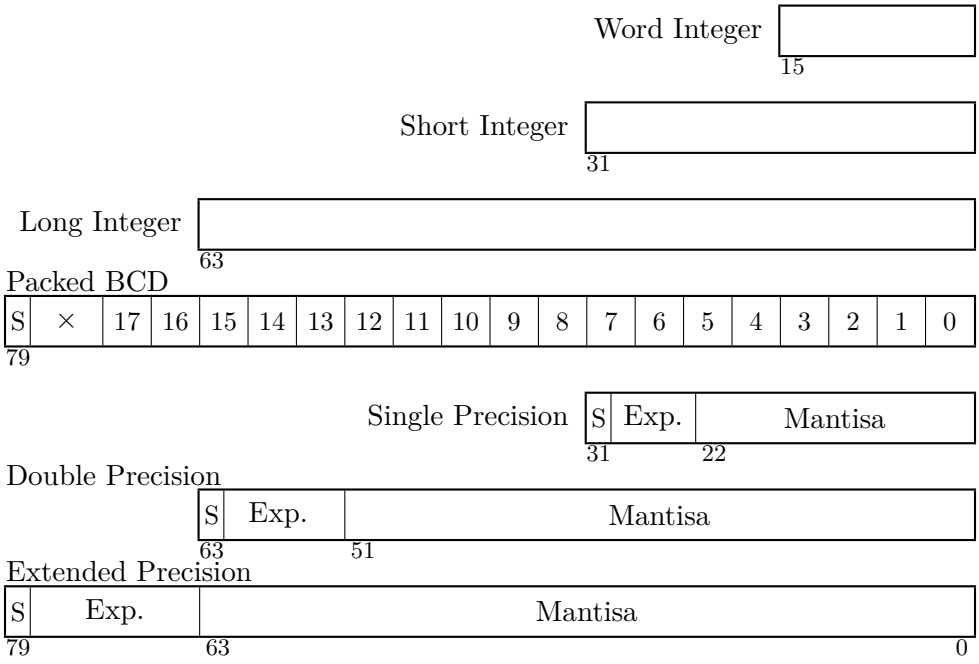
Zhuštěné BCD číslo (Packed BCD) je 18 BCD číslic (viz str. 69) se znaménkem v 80 bitech. Znaménko je uloženo v bitu nejvyššího řádu stejně jako u celých čísel.

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.



Obr. 5.15. Typy dat zpracovávaných FPU

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. 5.15 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.

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. Vezmemeli 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

5.9.2 Výsadní symboly

Nula (Zero) může být kladná nebo záporná. Znaménko u čísla nula však při výpočtu nemá význam a 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.

5.9.3 Přerušení FPU

Detekuje-li FPU chybu při výpočtu, oznámí ji nastavením aktivní úrovně na signálu $\overline{\text{FERR}}$. Procesor 80486 na tento stav odpoví přerušením INT 16. FPU umí detekovat několik různých chyb. Konkrétní druh chyby se oznamuje ve stavovém registru FPU. Možné chyby jsou následující:

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.

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.

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

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 ∞.

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.

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.

5.9.4 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.

Registr	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
řídicí					RC		PC				PM	UM	OM	ZM	DM	IM
stavový	B	C3	Vrchol		C2		C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE
doplňující	7		6		5		4		3		2		1		0	

Obr. 5.16. Registry FPU

Řídicí registr, jehož tvar je na obr. 5.16, 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. 5.16). Jsou čtyři možné kombinace:

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

5.9.5 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 80386, v 80486 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 80486 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.

5.10 Počáteční nastavení procesoru 80486

Po rozpoznání aktivní úrovně signálu RESET se zahájí test vnitřní logiky tehdy, byl-li současně se signálem RESET nastaven i signál AHOLD. Test trvá asi 42 ms. Bezchybné ukončení testu signalizuje opět nulový obsah EAX.

Procesor nastavuje jiný počáteční obsah těmto registrům:

DH = 4 (identifikace 80486),

CR0 = 6000 0000h (CD=1, NW=1).

Po inicializaci procesoru je vyprázdněna interní vyrovnávací paměť.

5.11 Nové instrukce procesoru 80486

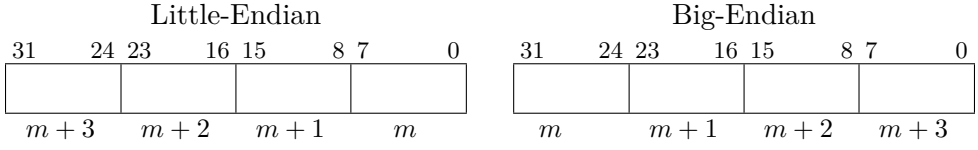
BSWAP *Byte Swap*

Popis:

O	D	I	T	S	Z	A	P	C

Procesor 80486, stejně jako předcházející typy, ukládá data delší než jedna slabika tak, že slabika nejnižšího řádu leží na nejnižší adrese (viz obr. na str. 133). 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. 5.17). 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.



Obr. 5.17. Srovnání tvaru Little-Endian a Big-Endian

Syntax: ⌜ BSWAP *operand*

Instrukce	*Příklad*	*Komentář*
BSWAP *r32*	`BSWAP EDX`	`; Zamění pořadí slabik v EDX`

XADD — *Exchange and Add*

Popis:

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Instrukce XADD naplní zdrojový operand obsahem cílového a naplní cílový operand součtem původního obsahu zdrojového operandu a cílového operandu.

Syntax: ⌜ XADD *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
XADD *r/m8,r8*	`XADD Slab,AL`	`; AL:=Slab a Slab:=původní AL+Slab`

```
XADD r/m16,r16   XADD Slovo,CX    ; CX:=Slovo a Slovo:=původní CX+Slovo
XADD r/m32,r32   XADD EAX,EBX     ; EBX:=EAX a EAX:=původní EBX+EAX
```

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 cílového operandu. Pokud jsou shodné, nastaví se příznak ZF na jedničku a cílový operand je naplněn obsahem zdrojového operandu. V opačném případě je vynulován příznak ZF a střádač (AL, AX nebo EAX) je naplněn cílovým operandem. Instrukce se smí použít s prefixem LOCK.

SYNTAX: ⌜ CMPXCHG *cílový_operand,zdrojový_operand*

Instrukce	*Příklad*	*Komentář*
CMPXCHG *r/m8,r8*	`CMPXCHG Slab,BL`	`; AL=Slab: ZF:=1 a Slab:=BL`
		`; Není-li: ZF:=0 a AL:=Slab`
CMPXCHG *r/m16,r16*	`CMPXCHG Slovo,CX`	`; AX≠Slovo: ZF:=1 a Slovo:=CX`
		`; Není-li: ZF:=0 a AX:=Slovo`
CMPXCHG *r/m32,r32*	`CMPXCHG EDX,EBX`	`; EAX=EDX: ZF:=1 a EDX:=EBX`
		`; Není-li: ZF:=0 a EAX:=EDX`

INVD — *Invalidate Cache*
WBINVD — *Write-Back and Invalidate Cache*

O	D	I	T	S	Z	A	P	C

POPIS: **CPL=0**

Obě instrukce vyprázdní celou interní vyrovnávací paměť procesoru 80486 a signalizují případné externí vyrovnávací paměti, že má udělat totéž. Instrukce WBINVD by před vyprázdněním měla opsat obsah IVP do paměti. V procesoru 80486 však není mezi WBINVD a INVD rozdíl, protože IVP pracuje metodou Write-Through (viz též str. 207). Instrukce nemusí být ve vyšších typech procesorů stejně implementována.

SYNTAX: ⌜ INVD
⌜ WBINVD

Instrukce	*Příklad*	*Komentář*
INVD	`INVD`	`; Vyprázdní celou IVP a externí VP`
WBINVD	`WBINVD`	`; Vyprázdní celou IVP a externí VP`

INVLPG *Invalidate TLB Entry*

O	D	I	T	S	Z	A	P	C

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 není v reálném ani ve V86 režimu rozpoznána a generuje se přerušení INT 6. Instrukce nemusí být ve vyšších typech procesorů stejně implementována.

SYNTAX: ⌜ INVLPG *operand*

Instrukce	*Příklad*	*Komentář*
INVLPG *m32*	`INVLPG DvojSlovo`	`; Zneplatnění položky TLB`

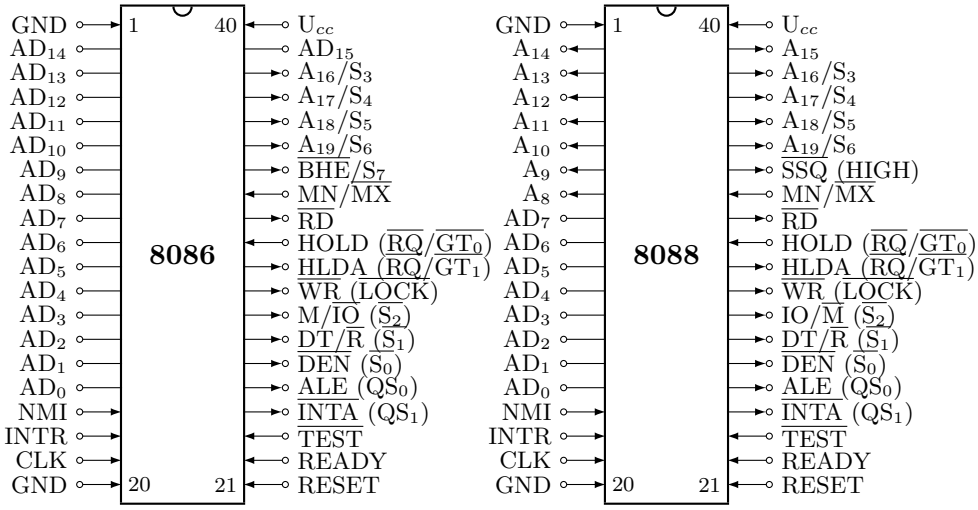
5.12 Procesor Intel 80486SX

Procesor 80486SX je levnější varianta procesoru 80486. Od plnohodnotného procesoru 80486 se liší pouze tím, že na čipu není FPU. Tento produkt je výsledkem konkurenčního boje firmy Intel s firmou AMD (Advanced Micro Devices), která uvádí na trh svoji variantu procesoru 80386, který je rychlejší než 80386 dostupný u firmy Intel. Při srovnávání tohoto rychlejšího 80386 (40 MHz) s 80486SX (20 MHz) dojdeme ke zjištění, že 80486SX je pomalejší pro 16bitové aplikace (MS-DOS), ale je rychlejší pro 32bitové aplikace (Unix, OS/2 verze 2.0, atd.).

A Popisy signálů

A.1 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.



Obr. A.1. Zapojení procesorů Intel 8086 a 8088

Význam následujících signálů je společný pro oba režimy:

AD$_0$÷AD$_{15}$ 16bitová datová a současně část 20bitové adresové sběrnice v procesoru 8086.

$AD_0 \div AD_7$ 8bitová datová a současně část 20bitové adresové sběrnice v procesoru 8088.

$A_8 \div A_{15}$ Druhá část 20bitové adresové sběrnice v procesoru 8088.

$A_{16}/S_3 \div A_{19}/S_6$ Nejvyšší bity 20bitové adresové sběrnice a část stavové informace.

$\overline{BHE}$ Je-li $\overline{BHE}$=L a A_0=L, přenáší se 16bitové slovo, je-li $\overline{BHE}$=L a A_0=H, přenáší se slabika z liché adresy (vyšší polovina slova) a je-li $\overline{BHE}$=H a A_0=L, přenáší se slabika ze sudé adresy.

$\overline{SSQ}$ Stavová informace v minimálním režimu (v max. vždy v úrovni H) v procesoru 8088.

$\overline{RD}$ Čtení z paměti nebo V/V zařízení.

READY Signál oznamující procesoru, že data na sběrnici jsou platná.

INTR Signál žádosti o maskovatelné přerušení.

$\overline{TEST}$ Signál testovatelný instrukcí WAIT. Při $\overline{TEST}$=L program pokračuje další instrukcí.

NMI Signál nemaskovatelného přerušení.

RESET Signál okamžitě ukončující aktivitu CPU a předávající řízení instrukci na adrese 0FFFF0h.

CLK Hodinový signál.

U_{cc} Napájecí napětí +5 V.

GND Společný vodič 0 V.

MN/$\overline{MX}$ Volba režimu procesoru maximální/minimální.

Další vývody mají odlišný význam pro jednotlivé režimy. V maximálním režimu platí:

$\overline{S_0} \div \overline{S_2}$ 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.

$\overline{\text{LOCK}}$ Uzamčení sběrnice pro procesor, který nastavil $\overline{\text{LOCK}}$=L instrukčním prefixem LOCK.

QS_0, QS_1 Signály oznamují stav vnitřní fronty procesoru.

V minimálním režimu platí:

M/$\overline{\text{IO}}$ Rozlišuje, zda adresa patří paměti nebo V/V v procesoru 8086.

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).

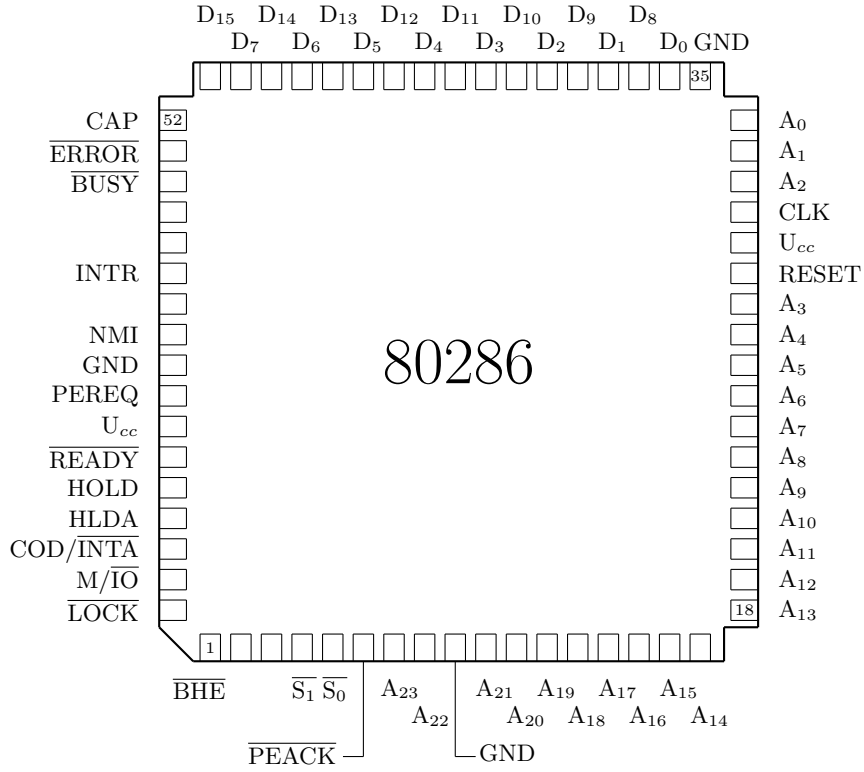
HOLD Žádost o přidělení lokální sběrnice od jiného zařízení.

HLDA Potvrzení odpojení procesoru od sběrnice.

A.2 Popis signálů procesoru Intel 80286

Procesor 80286 je umístěn ve čtvercovém integrovaném obvodu s 68 vývody. Vývody, které na obr. A.2 nemají přidělen název, jsou nezapojeny. Protože je význam některých signálů procesoru 80286 shodný s procesorem 8086, jsou v následujícím textu popsány pouze nové signály a signály se změněným významem.

CAP Mezi tento vývod a vývod GND musí být zapojen kondenzátor kapacity 0,047 μF $\pm$20 % 12 V vyhlazující nežádoucí napěťové zákmity.



Obr. A.2. Zapojení procesoru Intel 80286

$D_0 \div D_{15}$ 16bitová obousměrná datová sběrnice.

$A_0 \div A_{23}$ 24bitová adresová sběrnice. Při výstupu čísla V/V brány jsou využity pouze signály $A_0 \div A_{15}$.

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.

A.3 Popis signálů procesoru Intel 80386

Procesor je integrován do čtvercového keramického integrovaného obvodu, který má vývody na spodním povrchu (PGA – Pin Grid Array). Obvod má 132 vývodů (viz obr. A.3). Nepopsané vývody jsou nezapojeny. V dalším textu se zmíníme opět pouze o nových signálech a signálech se změněným významem.

$D_0 \div D_{31}$ 32bitová obousměrná datová sběrnice.

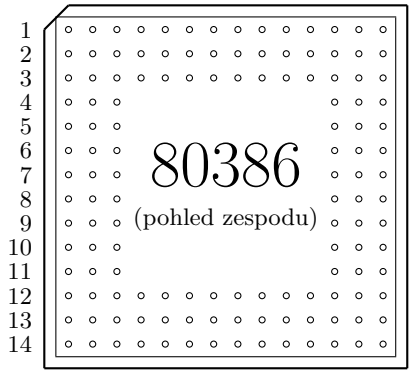
$A_2 \div A_{31}$ 32bitová adresová sběrnice adresující 32bitová dvojslova.

$\overline{\text{BE}_0} \div \overline{\text{BE}_3}$ Bližší určení přenášených slabik v rámci dvojslova.

$\overline{\text{BS16}}$ Volba 16bitového přenosu dat.

$\overline{\text{NA}}$ (Next Address) Slouží k zahájení výběru obsahu další adresy při proudovém zpracování.

D/$\overline{\text{C}}$, **$\overline{\text{ADS}}$**, **W/$\overline{\text{R}}$** jsou signály určené pro řízení sběrnice.



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. A.3. Zapojení procesoru Intel 80386

A.4 Popis signálů procesoru Intel 80486

Procesor je integrován do PGA se 168 vývody (viz obr. A.4).

$DP_0 \div DP_3$ Paritní bit pro každou slabiku přenášenou po sběrnici.

$\overline{\text{PCHK}}$ Chyba parity na sběrnici.

$\overline{\text{PLOCK}}$, **$\overline{\text{ADS}}$**, **$\overline{\text{RDY}}$**, **$\overline{\text{BRDY}}$**, **$\overline{\text{BLAST}}$**, **$\overline{\text{BOFF}}$** Signály pro řízení sběrnice.

AHOLD, **$\overline{\text{EADS}}$** Signály pro řízení vnitřní vyrovnávací paměti.

$\overline{\text{KEN}}$ Povoluje nebo zakazuje použití vnitřní vyrovnávací paměti.

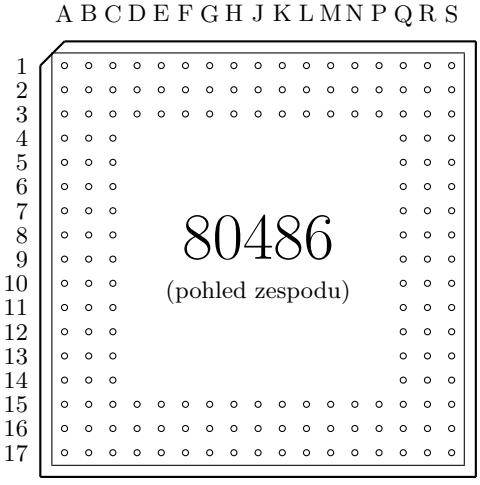
$\overline{\text{FLUSH}}$ Pokyn k vyprázdnění vnitřní vyrovnávací paměti.

PWT, **PCD** Signály přenášející hodnoty bitů PWT a PCD.

$\overline{\text{FERR}}$ Signál oznamuje chybu koprocesoru (podobně jako $\overline{\text{ERROR}}$).

$\overline{\text{IGNNE}}$ Ignorování chyb hlášených koprocesorem.

$\overline{\text{A20M}}$ Maskování adresy podle pravidel 8086.



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	$\overline{\text{D/C}}$	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. A.4. Zapojení procesoru Intel 80486

B Vzorový program

Vzorový program byl pro jednoduchost vytvořen v prostředí operačního systému MS-DOS a podle konvencí překladače MASM (Microsoft Macro Assembler). Odladěn byl překladačem Turbo Assembler (firmy Borland) verze 2.0. Jde o program typu `.EXE`, který částečně ukazuje, jak lze používat ladicí nástroje 80386. Je složen ze dvou částí: z rutiny obsluhující ladicí přerušení INT 1 a kódu, který rutinu instaluje a později vyzkouší. V 1. variantě program sleduje čtení nebo zápis dvojslova na adrese `LadBod` a ve 2. variantě sleduje výběr instrukce na adrese `LadInstr0`. V první variantě program vypisuje na obrazovku text např.:

```
b2313:0105 a2313:0109
i0FF1 2313:0109
```

Skupina `b2313:0105` oznamuje adresu začátku instrukce, která bude generovat INT 1 a skupina `a2313:0109` adresu instrukce následující. Další řádek je produkován obslužnou rutinou. Hodnota `0FF1` je obsah dolního slova registru DR6. Skupina `2313:0109` na druhém řádku je adresa, kterou ladicí nástroje uložily do zásobníku jako CS:IP. Podle čísel na prvním a druhém řádku lze ověřit, že sledování čtení nebo zápisu dat vyvolá přerušení typu „Trap“ (1. varianta) a sledování výběru instrukce vyvolá přerušení typu „Fault“ (2. varianta).

```
; Program demonstruje část ladicích možností procesoru 80386.
; Program pracuje pouze v reálném režimu pod operačním systémem MS-DOS.
.386p   ; Program bude používat instrukční repertoár procesoru 80386.
DATA    SEGMENT PUBLIC USE16 'DATA' ; Segment podle 16bitových pravidel.
OrgInt1 DD    0         ; Původní obsah přerušovacího vektoru číslo 1.
        ALIGN 4         ; Zarovnání na hranici dvojslova.
LadBod  DD    0         ; Cvičné ladicí místo. Je to dvojslovo a musí
DATA    ENDS            ; ležet na hranici dvojslova (adresa dělitelná 4).
;
STKSEG  SEGMENT STACK USE16 'STACK' ; Zásobník podle 16bitových pravidel.
        DB    1024 DUP(?)  ; Hloubka zásobníku je 1 KB.
```

```
STKSEG  ENDS
;
CODE    SEGMENT PUBLIC USE16 'CODE' ; Vlastní program v segmentu opět
        ASSUME   CS:CODE,SS:STKSEG,DS:DATA  ; podle 16bitových pravidel.
;
Vypis   MACRO znak      ; Makro na obrazovku vypíše ASCII znak.
        mov   AL,&znak  ; Parametr makra vlož do registru AL.
        mov   AH,14     ; Použijeme službu ROM-BIOSu pro zobrazování,
        int   16        ; která obsah AL vypíše na místo, kde je kurzor,
        ENDM            ; a kurzor posune o jeden sloupec doprava.
;
Int1    PROC    FAR     ; Rutina obsluhující přerušení INT 1.
        sub     SP,6    ; Vymez prostor pro simulaci 32bitového přeruš.
        pushad          ; Ulož všech 8 registrů 32bitově do zásobníku.
        mov     BP,SP   ; Poznamenej současnou hodnotu SP do BP.
        add     BP,32   ; Nyní BP ukazuje do zás. nad uložené registry.
; Provedeme simulaci 32bitového přerušení proto, aby při návratu byl
; nastaven příznak RF v registru EFLAGS na jedničku. Pokud by byl RF=0
; a sledoval by se výběr instrukce, došlo by po návratu k zacyklení.
        movzx   EAX,word ptr [BP+6]  ; Vezmi 16bitové IP a 32bitové
        mov     dword ptr [BP],EAX   ;   je ulož do zásobníku.
        movzx   EAX,word ptr [BP+8]  ; Vezmi 16bitové CS a 32bitové
        mov     dword ptr [BP+4],EAX ;   je ulož do zásobníku.
        movzx   EAX,word ptr [BP+10] ; Vezmi 16bitové FLAGS,
        or      EAX,00010000h        ;   horních 16 bitů doplň o RF=1
        mov     dword ptr [BP+8],EAX ;   a 32bitové ulož do zásobníku.
; Analýza obsahu stavového registru DR6 - zde výpis na obrazovku DR6
        Vypis   'i'     ; ve tvaru "iXXXX", kde XXXX je dolní slovo DR6.
        mov     EAX,DR6 ; Vezmi obsah DR6 (...,BT,BS,BD,...,B3,B2,B1,B0).
        call    tAX     ; Bezprostředně za "i" vypiš obsah dolního slova
        Vypis   ' '     ; DR6 (BT,BS,BD,...,B3,B2,B1,B0) a mezeru.
        mov     AX,word ptr [BP+4] ; Ze zásobníku vezmi 16bitové CS,
        call    tAX     ; vypiš je ve tvaru YYYY a
        Vypis   ':'     ; zakonči dvojtečkou.
        mov     AX,word ptr [BP]   ; Ze zásobníku vezmi 16bitové IP a
        call    tAX     ; vypiš je ve tvaru ZZZZ.
; Po dokončení této rutiny bylo na obrazovku vypsáno: "iXXXX YYYY:ZZZZ".
        xor     EAX,EAX ; Vynuluj obsah DR6, protože to procesor sám
        mov     DR6,EAX ; nedělá a protože nechceme informace kumulovat.
        popad           ; Obnov všech 8 32bitových registrů ze zásobníku.
        iretd           ; Proveď návrat do přerušeného procesu 32bitově.
Int1    ENDP            ; Při návratu je nastaveno RF=1.
;
```

```
tAX     PROC    ; Podprogram pro výpis obsahu AX na obrazovku ve formě
        push    AX      ; čtyř šestnáctkových (hex) číslic.
        mov     AL,AH   ; Nejprve 2 číslice vyšších řádů.
        call    tAL     ; Vypiš 2 číslice.
        pop     AX      ; Potom 2 číslice nižších řádů.
        call    tAL     ; Vypiš 2 číslice.
        ret             ; Návrat z podprogramu.
tAX     ENDP
tAL     PROC    ; Podprogram pro výpis obsahu AL na obrazovku ve formě
        push    AX      ; dvou šetstnáctkových (hex) číslic.
        shr     AL,4    ; Nejprve 1 číslice vyššího řádu.
        call    hex     ; Vypiš 1 číslici.
        pop     AX      ; Potom 1 číslice nižšího řádu.
        and     AL,0fh
        call    hex     ; Vypiš 1 číslici.
        ret             ; Návrat z podprogramu.
tAL     ENDP
hex     PROC    ; Převeď hex číslici v AL na ASCII a vypiš na obrazovku.
        or      AL,30h  ; Uprav číslici podle pravidel ASCII kódu.
        cmp     AL,3Ah  ; Je to ještě číslice?
        jl      Cislce  ; Ano.
        add     AL,7    ; Není, proveď korekci na písmeno.
Cislce: mov     AH,14   ; Vypiš ASCII znak z AL na obrazovku pomocí
        int     16      ; služby ROM-BIOSu (viz též makro Vypis).
        ret             ; Návrat z podprogramu.
hex     ENDP
;*************************************************************************
PROCES  PROC    ; Toto je vstupní bod programu. Sem se předá řízení.
        mov     AX,DATA ; Segmentový registr DS naplníme segmentovou
        mov     DS,AX   ; částí adresy segmentu pojmenovaného DATA.
; Nastavení adresy obslužné rutiny do přerušovacího vektoru INT 1.
        xor     AX,AX   ; Vynulujeme segmentový registr ES, který
        mov     ES,AX   ; použijeme pro odkaz do přerušovacího vektoru.
        mov     AX,CS   ; Do AX dej segm. část adresy segmentu CODE.
        cli             ; Budeme měnit přerušovací vektor - zakaž přer.
        mov     EBX,dword ptr ES:[4]   ; Schovej původní obsah INT 1.
        mov     word ptr ES:[4],offset Int1 ; [0:0004] := offset Int1
        mov     word ptr ES:[6],AX          ; [0:0006] := segment Int1
        sti             ; Konec změny přeruš. vektoru - povol přerušení.
        mov     OrgInt1,EBX ; Ulož původní hodnotu INT 1.
; Nastavení lineární adresy zájmového místa do DR0.
        xor     EAX,EAX ; Vynuluj EAX (zajímá nás horní slovo).
        mov     AX,DS   ; V 1. variantě budeme sledovat čtení/zápis dat.
```

```
;;VAR2;;mov     AX,CS   ; V 2. variantě můžeme sledovat výběr instrukce.
        rol     EAX,4   ; Segmentovou část vynásobíme 16.
        add     EAX,offset LadBod    ; Přičteme offset dat (1.varianta).
;;VAR2;;add     EAX,offset LadInstr0 ; Přičteme offset instrukce (2.var).
        mov     DR0,EAX ; Lineární adresu ladicího bodu vložíme do DR0.
        xor     EAX,EAX ; Vynulujeme DR6, abychom vymazali případné
        mov     DR6,EAX ;   předcházející indikace.
; Nastavíme obsah registru DR7  LNRW  G   GL      GL
; tak, aby aktivoval DR0  pro:   0 0  D   EE      00
        mov     EAX,00000000000011110000000000000001b ; č/z dvojslova (1)
;;VAR2;;mov     EAX,00000000000000000000000000000001b ; výběr instr. (2)
        mov     DR7,EAX ; Zapíšeme do DR7 - nyní může nastat INT 1.
; Demonstrační informace: Vypiš adresu před instrukcí ve tvaru:
        Vypis   'b'     ; "bXXXX:YYYY " (b jako Before).
        mov     AX,CS   ;
        call    tAX     ; Vypiš XXXX (CS).
        Vypis   ':'     ; Vypiš ":".
        mov     AX,offset LadInstr0
        call    tAX     ; Vypiš YYYY (offset instrukce).
        Vypis   ' '     ; Doplň mezeru.
; Demonstrační informace: Vypiš adresu za instrukcí ve tvaru:
        Vypis   'a'     ; "aXXXX:YYYY".
        mov     AX,CS
        call    tAX
        Vypis   ':'
        mov AX,offset LadInstr1
        call    tAX
        Vypis   10      ; Nový řádek na obrazovce.
        Vypis   13
; Proveď instrukci, která by měla vyvolat přerušení INT 1.
LadInstr0:  mov   EAX,LadBod     ; nebo:  mov AX,word ptr LadBod+2
LadInstr1:                       ; nebo:  mov AL,byte ptr LadBod+3
; Konec demonstrace.  Zruš hlídání podle DR7.
        mov     EAX,00000000000000000000000000000000b
        mov     DR7,EAX      ; Hlídání se zruší vynulováním DR7.
        mov     EBX,OrgInt1  ; Nastavíme původní obsah přerušovacího
        mov     dword ptr ES:[4],EBX ; vektoru INT 1.
        mov     AH,4ch       ; Konec programu (službou MS-DOSu) a
        int     21h          ; návrat do MS-DOSu.
PROCES  ENDP
;***********************************************
CODE    ENDS    ; Konec programu.
        END     PROCES
```

Literatura

[1] Vieillefond, C.: Programming the 80286, SYBEX Alameda, USA, 1987.

[2] Brumm, P.-Brumm D.: 80386 A Programming and Design Handbook, TAB BOOKS Inc., USA, 1987.

[3] Turley, J. L.: Advanced 80386 Programming Techniques, Osborne McGraw-Hill, Berkely, USA, 1988.

[4] i486 Microprocessor, Intel Corporation, USA, 1989.

[5] Turbo Assembler, Borland International Inc., USA, 1990.

[6] 386USERS, časopis vydávaný v sítích Internet a Bitnet, které jsou zpřístupněny díky Akademické iniciativě IBM v Československu.

Michal Brandejs

Mikroprocesory Intel 8086 – 80486

Původní vydání Grada, Praha 1991
Další elektronické vydání Michal Brandejs, Brno 1. 9. 2010
http://www.fi.muni.cz/usr/brandejs/Brandejs_Mikroprocesory_Intel_8086_80486_2010.pdf