Mikroprocesory II.

Letní semestr

Doc. Ing. Miroslav Čech, CSc.

míst. 234 – Trojanova, 2.NP

tel.: 778 532 036

E-mail: miroslav.cech@fjfi.cvut.cz

http://people.fjfi.cvut.cz/cechmiro/vyuka.html

Literatura

- V.Jirovský: Principy počítačů, Matfyz Press, Praha 2000
- J.Blatný, K.Kristoufek, Z.Pokorný, J.Kolenička: Číslicové počítače, SNTL, Praha 1982
- J.Pinker: Mikroprocesory a mikropočítače, Ben, Praha 2004
- J.Strelec, M.Liška: Architektury procesorů RISC, Grada, Praha 1992
- M.Brandejs: Mikroprocesory INTEL Pentium a spol., Grada, Praha, 1994
- H. Häberle: Průmyslová elektronika a informační technologie, Europa-Sobotáles cz., Praha 2003
- J.Pinker,M.Poupa:Číslicové systémy a jazyk VHDL, Ben, Praha 2006
- P. Tvrdík: Architektura superskalárních procesorů, PC World č.11/1994
- Firemní literatura Microchip
- Firemní literatura Intel
- www.wikipedia.com

Přehled mikroprocesorů Intel

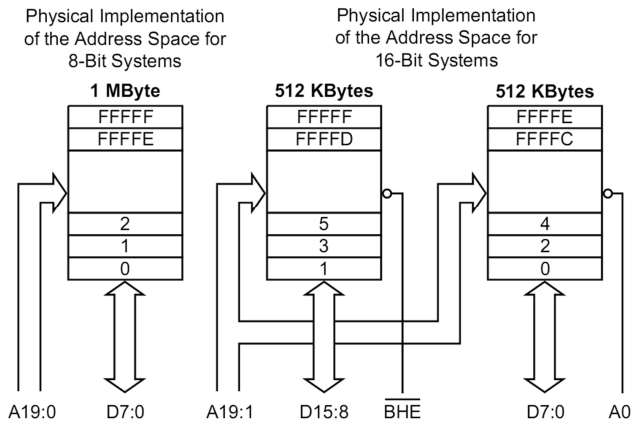
- 1968 založena firma Intel (Robert Noyce a Gordon Moore) - **Int**egrated **El**ectronics
- 1970 – mikroprocesor 4004 - 4bitová struktura, 2 250 tranzistorů MOS , rychlost 60 000 operací za sekundu. Adresoval 1 280 půlslabik dat a 4 KB instrukcí
- 1972 – první 8bitový mikroprocesor 8008 sestavený z 3 300 tranzistorů MOS a pracující rychlostí 30 000 operací za sekundu, paměť max. 16 KB.
- 1974 - mikroprocesor 8080 - 8bitový procesor, na čipu je 4 500 NMOS tranzistorů, rychlost 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.
- 1978 - první 16bitový mikroprocesor Intel 8086. Jde o první člen řady iAPX 86. 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.
- 1979 - mikroprocesor Intel 8088, což je modifikace 8086. Liší se tím, že má vnější 8bitovou strukturu (datová sběrnice), 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 (dodnes vyráběny a používány především v řídících aplikacích) 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.

Procesor 8086

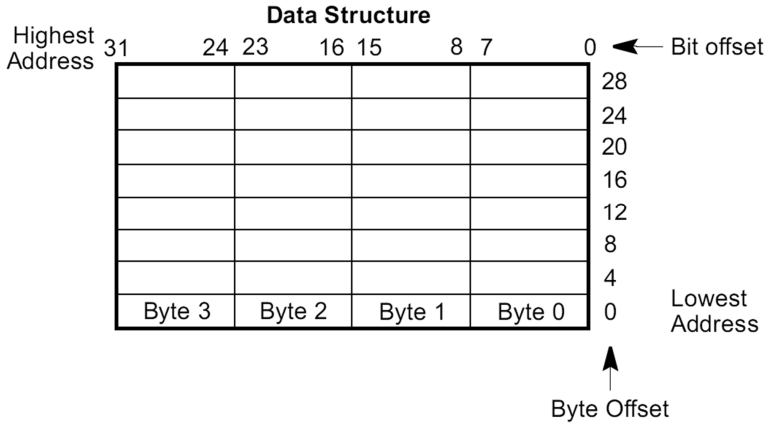
opakování ze zimního semestru

- Typická von Neumannova architektura – adresový prostor paměťový (společný pro program i data – velikost 1 MB) a adresový prostor vstupně-výstupních zařízení (pro adresaci portů – max. velikost 64k portů)
- Pracuje s daty šířky 16 nebo 8 bitů a vytváří 20bitovou adresu (tzv. fyzickou adresu) pro adresování 1 MB paměti. Virtuální paměť nepodporuje. Paměť je rozdělena na slabiky. Může být šíře 8 nebo 16 bitů.
- Pro uložení dat v paměti používá reprezentaci Little Endian (na nižší adrese méně významná slabika).
- Znaménková čísla jsou vyjádřena ve dvojkovém doplňku.
- Umí pracovat i s BCD čísly.

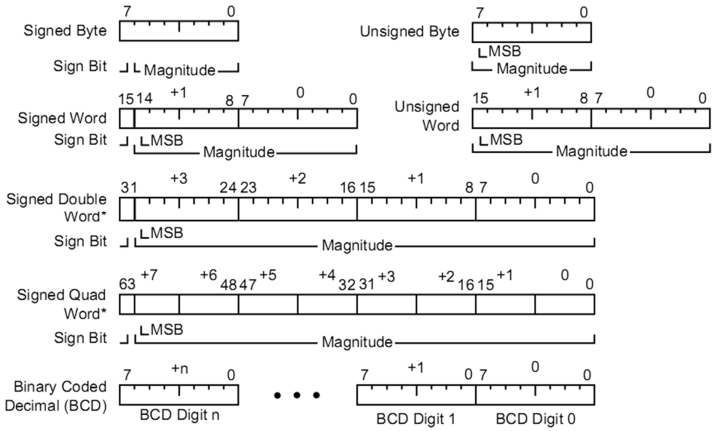
Organizace paměti šíře 8 nebo 16 bitů

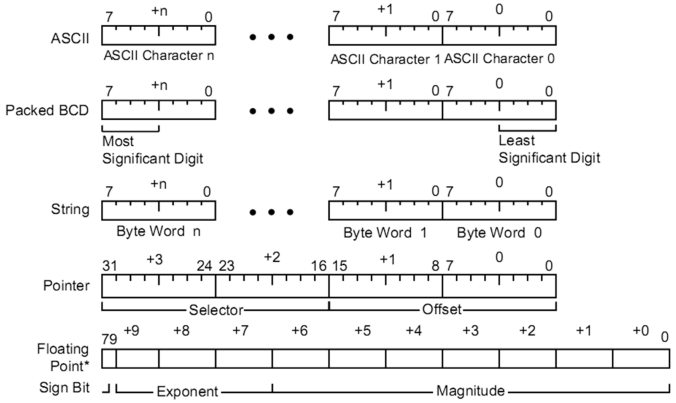


Adresace paměti při šíři 32 bitů



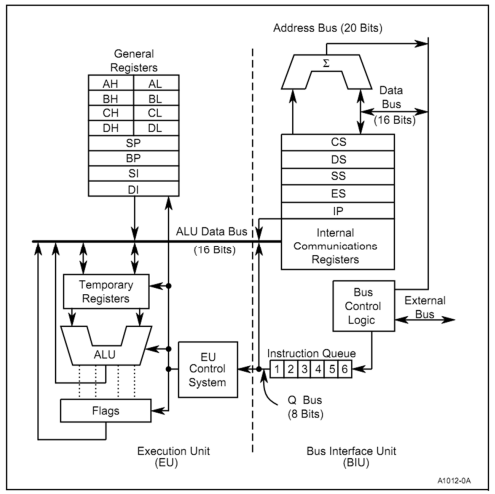
Formát dat v paměti





NOTE: *Directly supported if the system contains an 80C187.

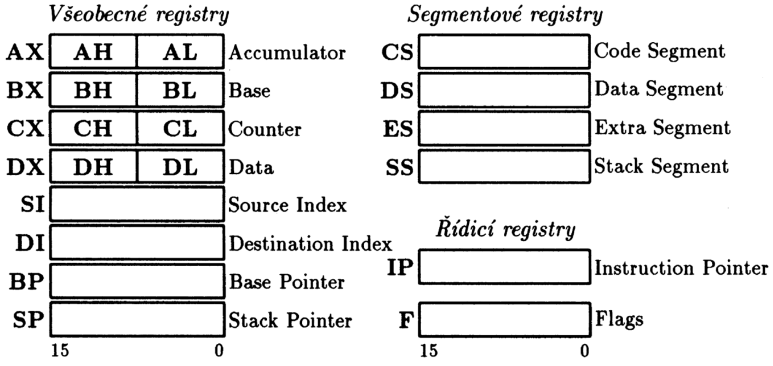
Struktura procesoru 8086



Dvě nezávislé jednotky EU a BIU

- Jednotka **Bus Interface Unit (BIU)** – **Sběrnicová jednotka** –vybírá instrukční kód z paměti a přenáší data po sběrnicích. Ukládá vybrané instrukční kódy do instrukční fronty.
- Jednotka **Execution Unit (EU)** – **Prováděcí jednotka** – vybírá instrukční kódy z instrukční fronty a postupně je zpracovává (náznak zřetězeného zpracování instrukcí v RISC procesorech). Problémy nastávají při instrukcích skoku. Instrukční fronta se musí vyprázdnit a BIU nahrát nové instrukční kódy.

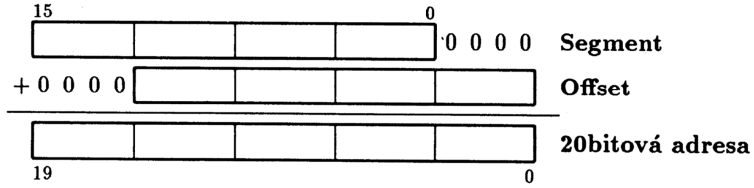
Vnitřní registry 8086



Logická a fyzická adresa

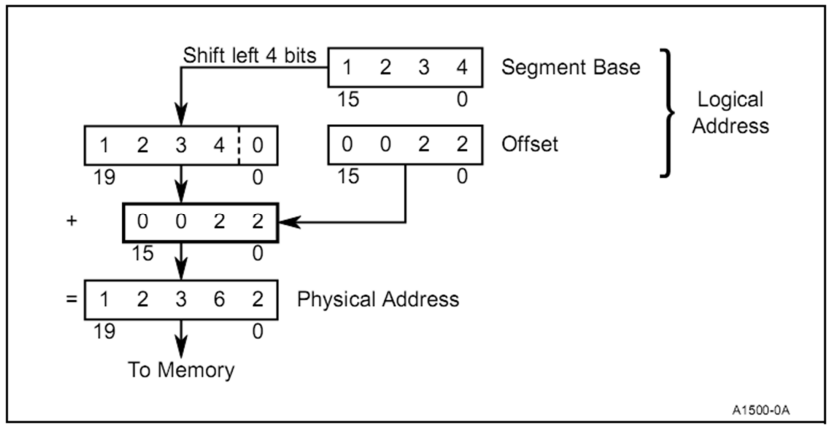
- **Logická adresa** – 32 bitové číslo skládající se ze dvou 16bitových čísel – segmentu a offsetu
- Zapisujeme **segment:offset** (např. 21AB:0030)
- Používá programátor
- **Fyzická adresa** – pro přístup do paměti – BIU jednotka převádí logickou adresu na fyzickou.
- Segmentová část se obyčejně vezme z tzv. segmentového registru (CS, SS, DS, ES)
- Převod z logické adresy na fyzickou - 16bitová segmentová část se vynásobí 16 (posune o 4 bity doleva) a přičte se offsetová část => 20bitové číslo

Převod logické adresy na fyzickou

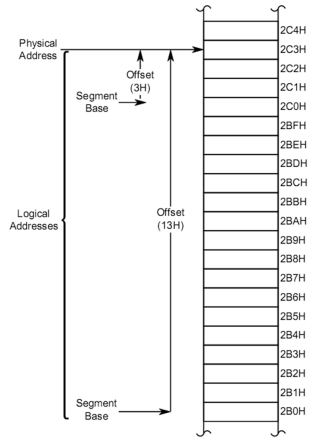


- Převod je jednoznačný.
- Převod z fyzické adresy na logickou je nejednoznačný – existuje 4096 možných vyjádření

Ukázka převodu logické adresy 1234:0022 na fyzickou



Převod fyzické adresy na logickou

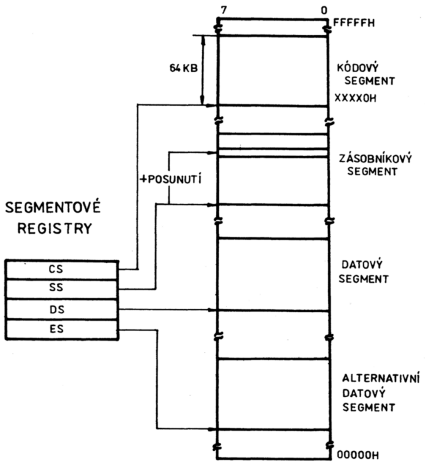


Ukázány 2 možnosti převody fyzické adresy 002C3H na logickou: 002B:0013H a 002C:0003H

Segmentové registry

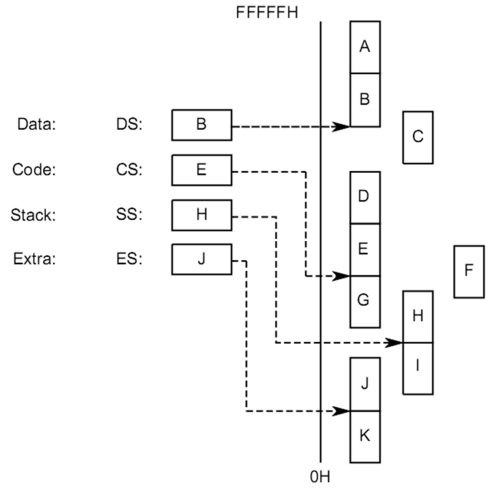
- CS (Code Segment) – je určen pro výpočet adresy instrukce (definuje instrukční segment). Logická adresa právě zpracovávané instrukce je CS:IP
- SS (Stack Segment) – definuje zásobníkový segment. Vrchol zásobníku je určen logickou adresou SS:SP
- DS (Data Segment) – definuje datový segment. Použije se při výpočtu adresy dat
- ES (Extra Segment) – obdoba DS registru, definuje pomocný datový segment nebo se používá při tzv. řetězcových instrukcích

Způsob adresace paměti 8086

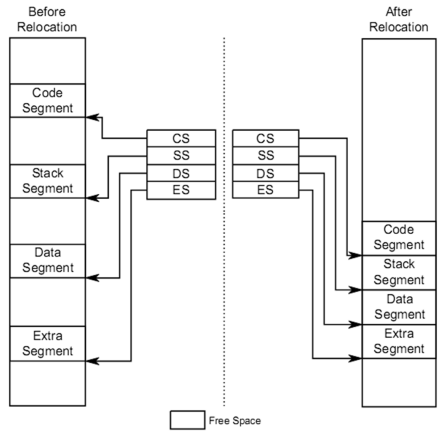


tzv. segmentování paměti pomocí segmentových registrů – okna do adresového prostoru

Rozmístění a zpřístupnění segmentů v paměti



Výhoda použití segmentování na tzv. relokaci paměti



Používá se často v operačních systémech na tzv. defragmentaci paměti

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

Příklady adresovacích technik si budeme uvádět na instrukci **MOV AH,operand**, která naplní registr AH zadanou hodnotou

- Přímý operand
  - Operand je uložen přímo v instrukci (za operačním kódem) jako konstanta.
  - Příklad: Instrukce MOV AH,50 naplní registr AH číslem 50.
- 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.
  - Např. instrukce MOV AH,BL přepíše obsah registru BL do AH.

Dalších šest technik zpřístupňuje operandy uložené ve fyzické paměti. Základní operace výpočtu 20bitové fyzické adresy je součástí každé z technik. Část adresy nazývaná segment je uložena v některém ze čtyř segmentových registrů. V instrukci se potom určí, ze kterého z registru 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 následující obrázek:

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.

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

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

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

1. přímé adresy (Displacement) uložné 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.

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.

Pozn.

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.

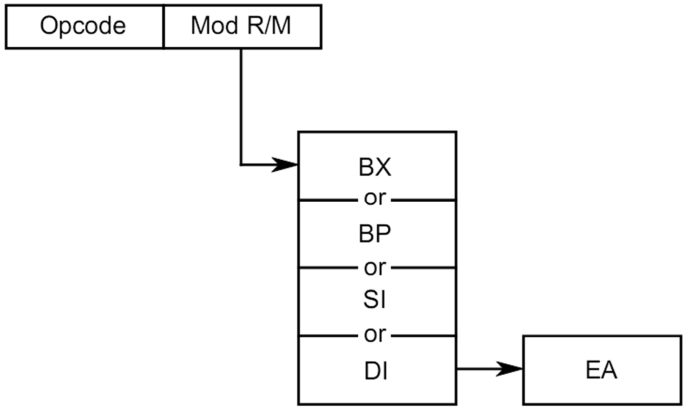
Přímá adresa

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 hranatých závorkách, znamená to, že se do AH neuloží jeho obsah, ale obsah ležící na adrese uvedené v zadaném registru.

Nepřímá adresa



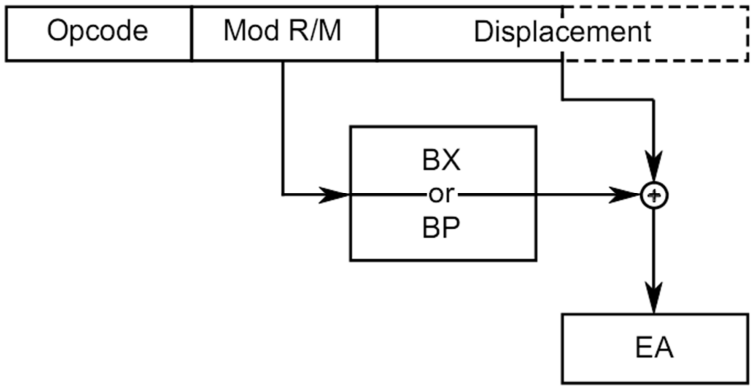
Bázovaná adresa

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.

Bázovaná adresa



Indexovaná adresa

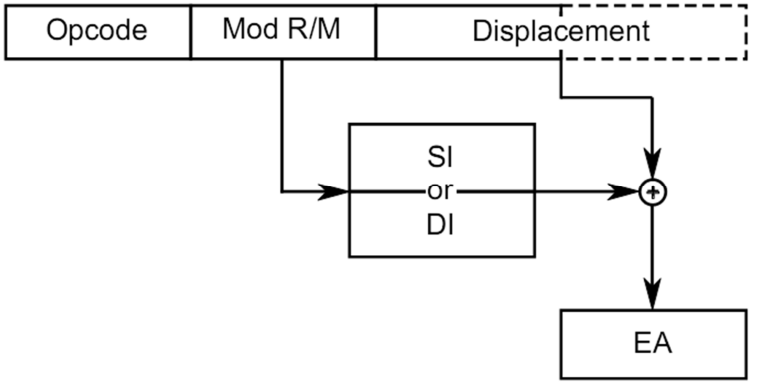
Offsetová část adresy se získá sečtením přímé adresy umístěné v instrukci a jednoho z indexových registrů (SI nebo DI).

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.

Indexovaná adresa



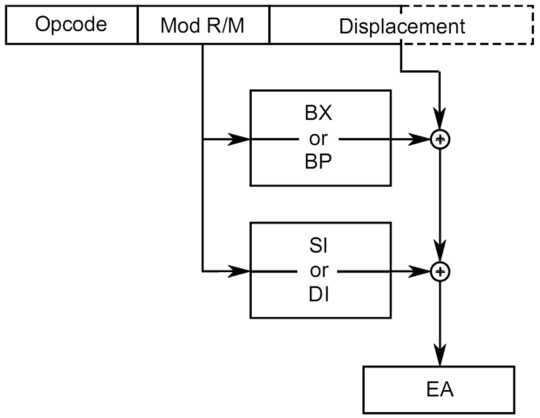
Kombinovaná adresa: báze+index

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.

Kombinovaná adresa



Kombinovaná adresa: přímá+báze+index

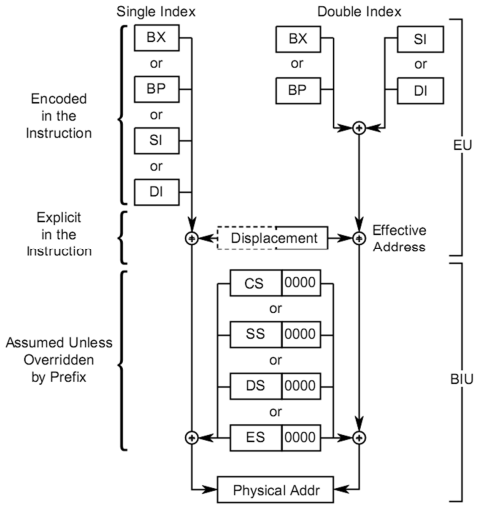
V tomto případě je offset vypočítán jako součet obsahu bázového registru (BX nebo BP), indexového registru (SI nebo DI) a přímé adresy uvedené v instrukci.

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.

Přehled všech adresovacích technik



Adresovací techniky a změna implicitního segmentového registru

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

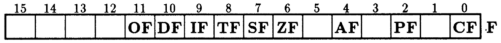
Segmentový registr	*Instrukční prefix*
DS (Data Segment)	3Eh
CS (Code Segment)	2Eh
SS (Stack Segment)	36h
ES (Extra Segment)	26h

Obr. 2.10. Prefixy měnící přiřazení segmentových registrů

Příklady změn implicitního segmentového registru

- MOV AH,CS:[BX] - Nepřímá adresa CS:BX (nikoli implicitně 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

Příznakový registr 8086



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. 67).

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).

Přerušení - Interrupt

- Přerušení dělíme do skupin podle toho, čím jsou generovány:
  - Technickými prostředky (vnější):
    - Nemaskovatelná (NMI)
    - Maskovatelná (INTR)
  - Programově (vnitřní):
    - Instrukcí INT x
    - Chybou při běhu programu
- Procesor rozlišuje 256 různých přerušení. Přerušovací řadič jako odpověď na signál INTA umístí na datovou sběrnici 8-bitové číslo 0-255
- Pro každé přerušení je v paměti uloženo 32 bitů adresy (tzv. přerušovací vektor) začátku obslužného přerušovacího podprogramu
- Adresy jsou zapsány v tabulce přerušovacích vektorů, která začíná na adrese 0000:0000 a má velikost 1kB (256x4 byty). Index do této tabulky je číslo 0-255 vrácené přerušovacím řadičem na signál INTA.

Tabulka přerušovacích vektorů

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

- Přerušení způsobená vnějšími V/V zařízeními lze maskovat vynulováním příznaku **IF** v příznakovém registru instrukcí **CLI**. Tento zákaz přerušení se nevztahuje na vnitřní přerušení a NMI
- Zakázané přerušení se povoluje instrukcí **STI**
- Předpokládejme, že nastalo přerušení číslo **n** nebo procesor dekódoval instrukci **INT n**. Potom se provedou činnosti v tomto pořadí:

Vyvolání přerušovacího podprogramu:

1. do zásobníku se uloží registr příznaků (F),
2. vynulují se příznaky IF a TF (zakáže se přerušení a krokování),
3. do zásobníku se uloží registr CS,
4. registr CS se naplní 16bitovým obsahem adresy n x 4 + 2,
5. do zásobníku se uloží registr IP ukazující na neprovedenou instrukci,
6. registr IP se naplní 16bitovým obsahem adresy n x 4.

Návrat z přerušovacího podprogramu instrukcí IRET

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

Vyhrazená přerušení procesoru 8086

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)

Počáteční nastavení procesoru

- Procesor je iniciován aktivní úrovní signálu RESET. V tom okamžiku:
  - vynuluje příznakový registr (tedy zakáže přerušení a krokování)
  - vynuluje registr IP
  - vynuluje segmentové registry SS, DS a ES a registr CS naplní hodnotou 0FFFFH
  - Zruší obsah instrukční fronty
- První instrukce, která bude provedena po signálu RESET nebo zapnutí napájení je CS:IP = FFFF:0000H = 0FFFF0H (16 bytů před koncem 1MB adresového prostoru)

IA-32

(INTEL Architecture 32)

- 32-bitová architektura
- Poprvé použita u procesoru 80386
- Zavádí tzv. chráněný režim (Protected Mode)
  - Chráněný režim byl sice použit i u procesoru 80286 (verze mezi procesorem 8086 a 80386). Byl to ale 16-bitový procesor.
- 80386 je zpětně binárně kompatibilní s 8086 (v tzv. reálném režimu) i s 80286 (v chráněném režimu).

Organizace paměti

- **Fyzická paměť** - paměť připojená na sběrnici procesoru. Je organizována jako posloupnost slabik. Každé slabice je přiřazena **fyzická adresa**, která je v intervalu 0 až 2^{32}-1 (4 GB).
- **Virtuální paměť** - technické prostředky správy paměti odstiňují programátora od fyzických adres a poskytují mu tzv. virtuální paměť. Správa paměti dává programátorovi k dispozici segmentování a stránkování paměti.
- **Segmentování paměti** je mechanismus poskytující násobný a nezávislý přístup k adresovému prostoru.
- **Stránkování paměti** podporuje vytváření rozsáhlejšího adresového prostoru, než je dostupná kapacita fyzické paměti s použitím vnější diskové paměti. Mechanismy se mohou používat oba, nebo jenom některý z nich.
- Odkazuje-li se program na paměť, použije tzv. **logickou adresu**. Tato se segmentováním překládá na nesegmentovou tzv. **lineární adresu** (32b). Stránkování může lineární adresu dále přeložit na **fyzickou adresu** (32b). Není-li stránkování zapnuto, je fyzická adresa totožná s lineární.

- Paměť můžeme používat buď jako nestrukturovaný adresový prostor, podobně jako bychom přímo používali fyzické adresy (tzv. **flat model**), nebo lze paměť rozdělit do vzájemně nezávislých prostorů nazývaných **segmenty**. Různé segmenty se vytvářejí pro uložení instrukcí prováděného programu, pro jejich data nebo zásobník. Jeden proces může mít až **16 383** segmentů různých velikostí a typů. Jeden z důležitých významů existence segmentů je zvýšení spolehlivosti operačního systému. Např. zásobníky se umísťují do speciálních segmentů. V případě, že by do zásobníku bylo uloženo více položek, než je kapacita segmentu, detekuje se pokus o překročení hranic segmentu a nemůže tedy dojít k přepisu instrukcí nebo dat jiného nebo i vlastního procesu.
- **Logická adresa** se skládá ze dvou složek, ze **selektoru** a **offsetu**. **Selektor** ukazuje na jeden z **popisovačů** v některé tabulce. Popisovač mj . obsahuje **bázi** (adresu začátku) segmentu a **limit** (velikost) segmentu. **Offset** je potom relativní adresa uvnitř segmentu (počítá se od začátku segmentu). Offset nesmí překročit limit. Logickou adresu, ve které se jednotlivé složky (tj. selektor a offset) oddělují dvojtečkou, zapisujeme ve tvaru:

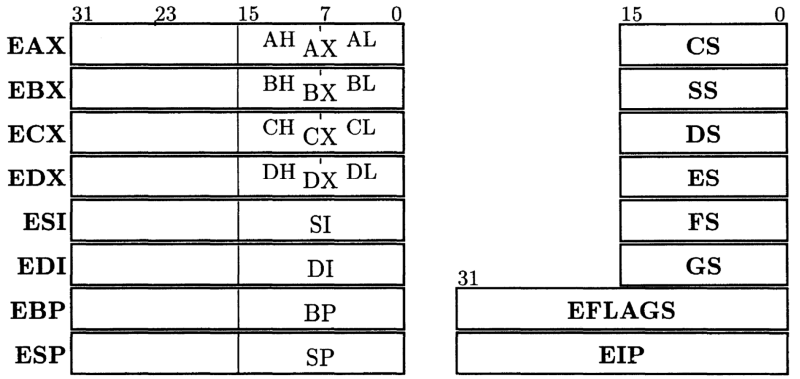
selektor:offset

Stránkování

- Stránkováním, je-li zapnuto, se lineární adresa rozděluje na **stránky** stejné velikosti. Stránka může potom být buď v paměti, nebo na disku. Fyzická paměť je totiž stránkováním rozdělena na **rámce** stejné velikosti jako stránky. Rámce potom operační systém „pronajímá“ stránkám, které jsou v daném okamžiku potřebné. Momentálně nepotřebné stránky se odkládají na disk. Odpovídající adresaci potom zajistí právě mechanismus stránkování. Není-li potřebná stránka v některém z rámců, generuje procesor tzv. výjimku, čímž se předá řízení programové rutině zajišťující přenos potřebné stránky do paměti.
- Stránkování je mechanismus určený operačnímu systému, aplikační programátor se problémy stránkování nemusí zabývat. Pro něj (ale ne jenom) je určena segmentace.

- **Nesegmentovaný model paměti (Flat model)** - procesor nemá sice možnost segmentaci vypnout, lze však všechny potřebné popisovače nastavit tak, aby ukazovaly na jeden a tentýž prostor lineárních adres. V tomto případě, kdy segmenty pokrývají celý dostupný adresový prostor, není možné (bez použití stránkování) kontrolovat procesy, zda přistupují jenom do svého rozsahu adres, těžko se hledají některé chyby v programech atd. Výjimka se generuje pouze při přístupu mimo dostupný adresový prostor.
- **Segmentovaný model paměti** - segmentovaný logický adresový prostor lze hypoteticky rozdělit až na 16 383 segmentů po 4 GB. Vzniklý virtuální adresový prostor se potom stránkováním mapuje do fyzické operační paměti. Výhodou segmentového modelu je, že se kontrolují přístupy k jednotlivým segmentům: kontroluje se překročení limitu, kontroluje se oprávnění přístupu k segmentu a kontroluje se typ operace prováděné nad segmentem.

Registry IA-32



Exx – 32bitové registry, registry FS a GS – rozšíření vzhledem k procesoru 8086

Příznakový registr EFLAGS

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

•Příznakové bity 0-11 totožné s procesorem 8086

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

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

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

- **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
- **AC (Alignment Check)** - spolu s bitem AM v registru CR0 zapíná generování přerušení při odkazu na paměť, který není „zarovnán“ na hranici odpovídající délce zpřístupňovaného objektu. Je-li AC=1 a je-li uskutečněn pokus o čtení nebo zápis např. 16bitového slova na lichou adresu, vyvolá se výjimka. Tato kontrola se provádí pouze na úrovni oprávnění 3 (tj. na uživatelské úrovni).
- **VIF (Virtual Interrupt Flag)** - je virtuální obraz příznaku IF pro daný V86 proces.
- **VIP (Virtual Interrupt Pending Flag) -** používá se spolu s příznakem VIF ve V86 procesech.
- **ID (Identification Flag)** - oznamuje, zda procesor má ve svém repertoáru instrukci CPUID. Procesor instrukci umí zpracovat tehdy, pokud má program možnost tento bit nastavovat a nulovat.

Výpočet efektivní adresy

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

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

Reálný režim

- Určen na spouštění programů napsaných pro procesor 8086 – 80186
- Architektura procesoru je redukována na architekturu 8086 – z Pentia se stane rychlá 8086
- Adresace paměti totožná s 8086. Využívá se pouze první 1MB paměti (přesně fyzické adresy 0 až 10FFEFh – důvod: je to adresa FFFF:FFFF vypočtená ve 32b aritmetice)

Přerušení a výjimky v reálném režimu

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

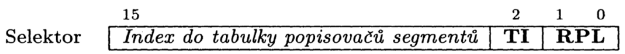
Správa paměti v chráněném režimu

- Je jiná než v reálném režimu
- Nabízí segmentaci a stránkování – segmentace se povinná, stránkování je volitelné.
- **Segmentace paměti**
  - Paměť je rozdělena na tzv. segmenty. Segment je souvislý prostor paměti velikosti až 4 GB, který může začínat na libovolné adrese. Každý segment je definován těmito parametry:
    1. **bází segmentu** (adresou začátku segmentu),
    2. **limitem segmentu** (délkou segmentu ve slabikách — 1),
    3. **přístupovými právy a typem segmentu**.
  - Uvnitř jednoho segmentu se pohybujeme pomocí zadání 32bitového offsetu, který přičtením k bázi určí lineární adresu v paměti.

- 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 umisťuje 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.

Logická adresa

Jako v reálném režimu je složena ze dvou složek – selektor segmentu a offset. Interpretace položky offset je stejná. Složka segment je v chráněném režimu nahrazena selektorem segmentu



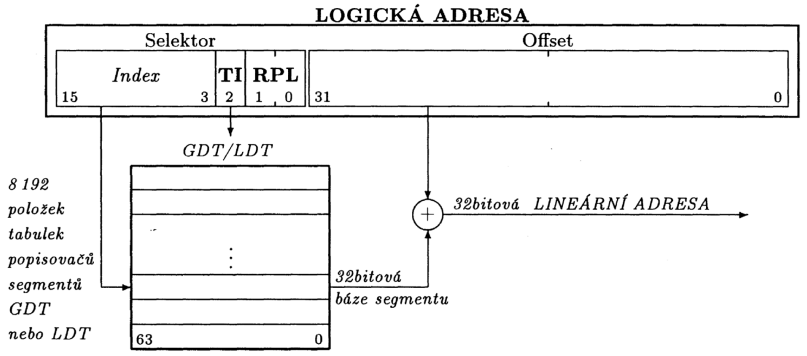
Obr. 5.1 Struktura selektoru

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

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

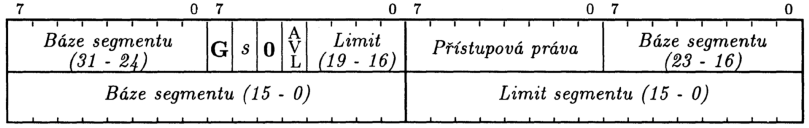
TI (Table Indicator) indikuje, ukazuje-li index do tabulky popisovačů segmentů lokálního adresovacího prostoru (TI=1) nebo globálního adresovacího prostoru (TI=0).

Převod logické adresy na lineární



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)**

Popisovač segmentu (Descriptor)



Báze segmentu definuje začátek segmentu v rámci 4 GB prostoru lineárních adres. Procesor si bázi složí ze tří polí (z důvodů kompatibility s procesorem 80286). Segment může začínat na libovolné adrese, žádoucí je však segmenty zahajovat na adresách dělitelných 16.

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

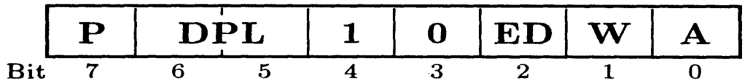
Hodnota položky *limit* se interpretuje podle obsahu bitu **G (Granularity**). Je- li **G=1**, může mít segment velikost až 4 GB. Je-li **G=0**, je maximální velikost 1 MB.

- **G (Granularity) -** nulová sděluje, že limit v popisovači je v jednotkách 1 B (potom segment může mít velikost max. 1 MB). Je-li G=1, jsou jednotkou limitu 4 KB (potom segment může mít velikost až 4 GB v násobcích 4 KB).
- **AVL (Available for Programmer Use)** je bit s nespecifikovaným významem, který může programové vybavení (operační systém) používat podle svých potřeb.
- **s** je bit, který má různý význam v závislosti na typu popisovače.
- Typ popisovaného segmentu je definován obsahem slabiky **přístupová práva.**

  Podle typu segmentu rozlišujeme tyto tři základní třídy popisovačů:

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

Popisovač datového segmentu
pole „Přístupová práva“



Ukazuje na segment v paměti obsahující data určité aplikace nebo zásobník.

Bity 4 a 3 určují, že jde o popisovač datového segmentu.

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

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

- **DPL (Descriptor Privilege Level)** určuje úroveň oprávnění přidělenou segmentu, který je adresován popisovačem.
- **W (Writable) -** nastaven na 1 , povoluje 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.
- Bit ***s*** znamená **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. 64KB a zaplňuje se od adresy max. FFFFH. Je-li B=1, může mít segment kapacitu až 4 GB (potom musí být G=1) a zaplňuje se od max. FFFFFFFFH k adrese *Limit*.
- Hodnota bitu B má ještě jeden význam: je-li B=0, je implicitní velikost položky v zásobníku (vybírané instrukcí POP a zapisované instrukcí PUSH) 16bitové slovo, a je-li B=1, je implicitní velikost 32bitové dvojslovo.

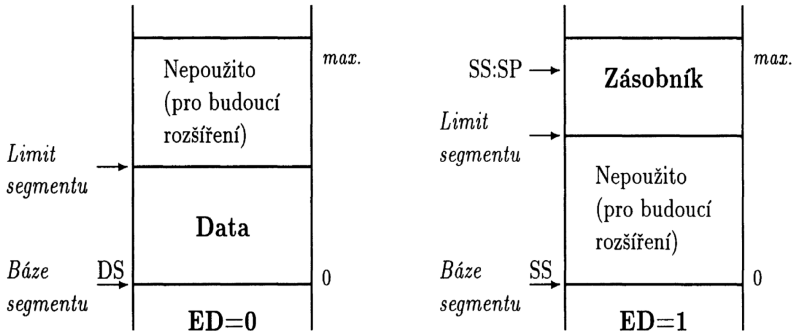
- **ED (Expand Down)** indikuje, kterým směrem se bude obsah segmentu rozšiřovat. Datové segmenty mohou obsahovat klasická data nebo zásobník. Při požadavku na zvětšení klasických dat se bude obsah rozšiřovat směrem k vyšším adresám. U zásobníku tomu bude naopak, protože ten se plní od vyšších adres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

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

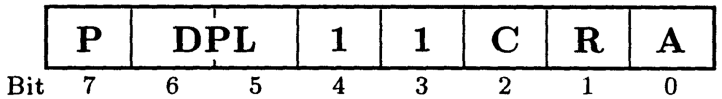
  Je-li nastaveno ED=1 (rostoucí dolů — zásobník), bude se obsah segmentu rozšiřovat směrem k nižším adresám. Položky zásobníku se ukládají od adresy 0FFFFFFFFH směrem k adrese 0. Při požadavku na zvětšení obsahu se musí zmenšit hodnota limitu segmentu (ten se totiž stále počítá od adresy 0). Správná hodnota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

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

Datový segment klasický (ED=0) a zásobník (ED=1)



Popisovač instrukčního segmentu pole „Přístupová práva“

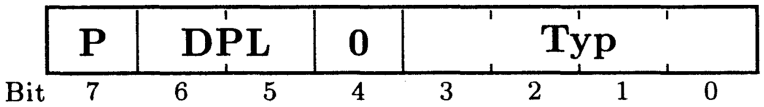


Instrukční segment obsahuje instrukce určené ke spuštění.

P a **DPL** - jako u popisovače 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)** - jako popisovač datového segmentu.
- Bit ***s*** je v popisovači instrukčního segmentu označován D a má tento význam:
- **D (Default)** určuje implicitní velikost efektivních adres a operandů používaných v tomto segmentu. Je-li D=0, je implicitní velikost efektivních adres a operandů 16 bitů (odpovídá Intel 286) 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.
- V reálném režimu není, ani po použití prefixu změny velikosti adresy, povoleno adresovat segmenty větší než 64 KB. Offset, který by překročil hodnotu FFFFH, způsobí výjimku 13.

Popisovač systémového segmentu pole „Přístupová práva“

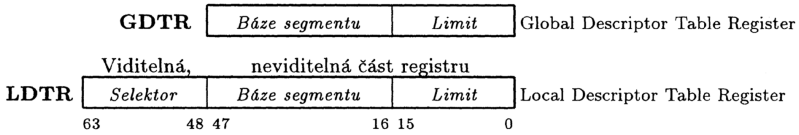


Popisovačů systémového segmentu je řada. Jsou to popisovače zvláštního určení. Jejich počet narostl zavedením 32bitových procesorů a udržováním kompatibility zdola. Bit ***s*** v popisovači systémového segmentu není použit.

Typ:
0 . . . nepovolená hodnota,
1 . . . TSS neaktivního procesu Intel286
2 . . . LDT všech procesorů
3 . . . TSS aktivního procesu Intel286
4 . . . brána pro předání řízení Intel286
5 . . . brána zpřístupňující TSS všech procesorů
6 . . . brána pro maskující přerušení Intel286
7 . . . brána pro nemaskující přerušení Intel286
8 . . . nepovolená hodnota
9 . . . TSS neaktivního procesu od Intel386
A . . . nepovolená hodnota
B . . . TSS aktivního procesu od Intel386
C . . . brána pro předání řízení od Intel386
D . . . nepovolená hodnota
E . . . brána pro maskující přerušení od Intel386
F . . . brána pro nemaskující přerušení od Intel386

- Do systémového segmentu smí zapisovat pouze procesor. Programátor (operační systém) si pro účely modifikace obsahu musí přes systémový segment překrýt datový
- Každá tabulka popisovačů segmentů je sama uložena v některém ze segmentů. Tabulek LDT může být v jednom okamžiku v paměti více. Tabulka popisovačů globálního adresového prostoru (GDT) je právě jedna. Pro uchování adresy jejího uložení je vyhrazen speciální registr **GDTR (Global Descriptor Table Register).**
- Každá LDT je specifikována jedním z popisovačů v GDT. Poněvadž s právě aktivním procesem je svázána jedna LDT, je pro snadnější přístup k adrese uložení této LDT vyhrazen rovněž jeden speciální registr nazvaný **LDTR (Local Descriptor Table Register).** Obsah registru LDTR se změní vždy, když je přepnuto zpracování na jiný proces tak, aby v každém okamžiku byla v LDTR adresa LDT právě aktivního procesu.

Registry GDTR a LDTR



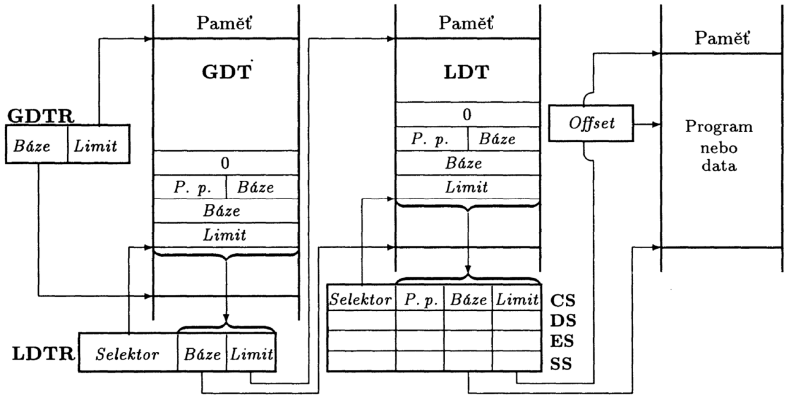
•Registr **GDTR (Global Descriptor Table Register)** uchovává adresu uložení tabulky popisovačů globálního adresového prostoru. GDTR obsahuje 4 slabiky báze segmentu s uloženou tabulkou a 2 slabiky limitu tohoto segmentu. Registr je přístupný pouze instrukcemi LGDT (Load GDT) a SGDT (Store GDT).

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

Segmentové registry

- Segmentové registry **CS, DS, ES, FS, GS a SS** jsou v chráněném režimu složeny ze dvou částí: část programátorovi „viditelná“ obsahuje 16bitový selektor a část programátorovi „neviditelná“ obsahuje slabiku přístupových práv , bázi segmentu, limit segmentu.
- Vždy při změně selektoru některého ze segmentových registrů použije procesor jeho hodnotu k výběru odpovídající položky z tabulky popisovačů a naplní neviditelnou část příslušnými hodnotami. Ta se potom použije při každém přístupu do paměti, protože se musí k bázi segmentu přičíst offset, zkontrolovat nepřekročení limitu a přístupová práva. Výhodné je mít tyto údaje umístěny v registrech proto, že doba přístupu do paměti (kde jsou tabulky popisovačů uloženy) je delší než do registru.
- Obsah selektoru registrů SS, DS, ES, FS a GS nastavujeme instrukcemi typu MOV, LES, LDS atd. Obsah selektoru registru CS se plní instrukcemi JMP, CALL a RETF ve variantě vzdáleného skoku a volání. Při každé změně selektoru se také kontroluje úroveň oprávnění.

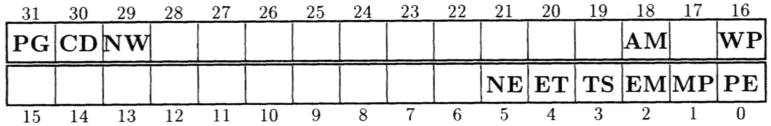
Použití GDTR a LDTR registru



Sdílení segmentů

1. Aplikační proces je uložen v instrukčním segmentu, který je v popisovači označen bitem R=0 (obsah segmentu se smí pouze spustit, nelze jej číst). Ladí-li programátor svůj aplikační program ladicím systémem, potřebuje tento systém instrukční segment číst i do něho zapisovat (označit ladicí body, atd.). Proto si ladicí systém vytvoří nový popisovač a segment označí jako datový s možností čtení i zápisu. Oba popisovače (instrukční pro aplikaci a datový pro ladicí systém) ukazují na stejnou oblast v lineární paměti.
2. 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.
3. Sdílení segmentu více popisovači používá také operační systém pro modifikaci svých systémových segmentů. Rovněž platí, že popisovače ukazující na stejné datové oblasti nemusejí mít stejné limity.

Řídící registr CR0



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

- **PE (Protected Mode Enable)** zapíná chráněný režim. Po inicializaci procesoru (signálem RESET) je zapnut reálný režim. Nastavením tohoto příznaku na 1 je procesor přepnut do chráněného režimu, vynulováním zpět do reálného režimu.
- **MP (Monitor Coprocessor**) indikuje fyzickou přítomnost koprocesoru (FPU) a řídí činnost instrukce WAIT.
- **EM (Emulate Coprocessor)** zapíná programovou emulaci koprocesoru tehdy, není-li koprocesor instalován - generuje výjimku 7
- **TS (Task Switch)** se nastavuje každým přepnutím procesu a testuje se při pokusu o provedení instrukce pro koprocesor. Je určen na vyvolání operace uložení kontextu koprocesoru při změně kontextu procesoru. Bit nuluje instrukce CLTS.

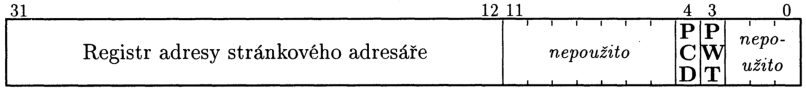
- **ET (Extension Type)** je v procesoru Pentium rezervován. Jinak sděluje typ instalovaného matematického koprocesoru. Bit nastavuje procesor během inicializace (po přijetí signálu RESET). Detekuje-li Intel287, nastaví ET=0, pro typ Intel387 a FPU Intel486 nastaví ET=1. Na základě bitu ET procesor používá buď 16bitový, nebo 32bitový protokol komunikace s koprocesorem. Bit lze nastavovat i programově.
- **NE (Numeric Error)** sděluje, jak se mají procesoru oznamovat chyby zjištěné v koprocesoru. Je-li NE=1, bude se generovat výjimka 16. Je-li NE=0 a vstupní signál IGNNE je aktivní, potom se chyba ignoruje. Je-li NE=0 a signál IGNNE je neaktivní, potom chyba zastaví procesor, který bude čekat na přerušení. O přerušení procesor současně žádá výstupním signálem FERR připojeným k řadiči přerušení (signál FERR emuluje signál ERROR koprocesorů Intel287 a Intel387). Stav NE=0 je kompatibilní s předcházejícími typy koprocesorů a operačním systémem MS-DOS, který chyby přijímal přes vnější přerušení.
- **WP (Write Protect)** je-li jedničkový, zakazuje zápis do stránek označených W=0 i procesům na úrovni oprávnění CPL<3. Procesor Intel386, bylo-li W=0, zakazoval zápis pouze procesu s CPL=3 a procesy s CPL<3 měly zápis povolen. Kompatibilita vyšších typů s Intel386 je zachována při WP=0. Je-li WP=1, nemůže do stránky označené W=0 zapisovat žádný proces.

- **AM (Alignment Mask)** maskuje náhodné nastavení příznaku AC v příznakovém registru . Je-li AM=0, není funkce AC zapnuta ani tehdy, je-li AC=1. Je-li AM=l, záleží na hodnotě AC. Maskování bitu AC je zavedeno proto, že programy přenesené z procesoru Inte1386 by mohly do tohoto příznaku zavádět různé nedefinované hodnoty. Nastavením AC=0 zabráníme výskytům výjimek 17.
- **NW (Not Write-Through)** je-li nulový, potom se všechny zápisy do paměti, jejichž položka je v interní vyrovnávací paměti (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.
- **CD (Cache Disable)** zapíná nebo vypíná IVP. Je-li CD=1, je IVP vypnuta tak, že položky, které při čtení nebyly ve vyrovnávací paměti nalezeny, se do ní nezapisují. IVP je zapnuta, jsou-li současně splněny tyto podmínky: CD=0, KEN=0 a PCD=0 (PCD je bit z CR3 nebo ze stránkové tabulky). Po inicializaci procesoru signálem RESET je nastaveno CD=1.
- **PG (Paging)** zapíná stránkovou jednotku určenou k transformaci lineárních na fyzické adresy.

Řídící registr CR2

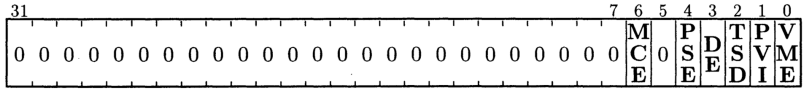
- Registr CR1 je rezervován pro využití budoucími typy procesorů.
- Registr CR2, je-li nastaven bit PG v CR0, obsahuje lineární adresu, která způsobila výpadek stránky detekovaný stránkovací jednotkou. Výpadek stránky má za následek generování výjimky 14. Registr je určen pouze ke čtení.
- Registry od CR2 jsou zavedeny od procesorů Intel386.

Řídící registr CR3



- Registr CR3 je rovněž využit pouze při zapnuté stránkovací jednotce. Obsahuje fyzickou adresu stránkového adresáře právě aktivního procesu (**Directory Base Address — DBA**). Dolních 12 bitů se při zápisu do tohoto registru ignoruje, protože stránkový adresář smí začínat pouze na hranici 4 KB stránky. Z dolních 12 bitů registru CR3 jsou využity bity 3 a 4
- Bity **PWT (Page Write-Through)** a **PCD (Page Cache Disable)** slouží k řízení vyrovnávacích pamětí. Hodnota těchto bitů se přenáší mimo procesor po stejnojmenných signálech tehdy, není-li zapnuto stránkování nebo se stránkování z nějaké příčiny obchází. V jiných případech se na výstup z procesoru předávají hodnoty bitů PWT a PCD ze stránkovacích tabulek odpovídající zpracovávané stránce.
- Bity PWT a PCD byly do registru CR3 zavedeny v procesoru Intel486.

Řídící registr CR4

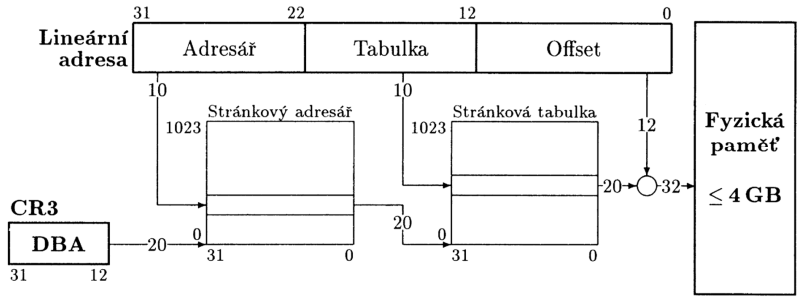


- **MCE (Machine Check Enable)** je-li nastaven na jedničku, je povoleno generování výjimky 18 (Machine Check Exception). Touto výjimkou se v procesoru Pentium oznamují chyby parity a jiné chyby technického vybavení.
- **PSE (Page Size Extensions)** zapíná používání stránek velikosti 4 MB při nastavení na 1.
- **DE (Debugging Extensions**) je-li nastaven na 1, jsou povoleny ladicí body sledující přístup k V/V branám
- **TSD (Time Stamp Disable)** nastavením tohoto příznaku se instrukce RDTSC (Read from Time Stamp Counter) stává privilegovanou. Instrukce je popsána ve zvláštních informačních dodatcích vydávaných firmou Intel.
- **PVI (Protected-Mode Virtual Interrupts)** je-li nastaven na 1, umožňuje používat virtuální IF v chráněném režimu Touto vlastností se některým programům určeným pro úroveň oprávnění 0 dovoluje běh i na úrovni oprávnění 3.
- **VME (Virtual-8086 Mode Extensions)** je-li nastaven na jedničku, je povoleno používat virtuální obraz příznaku IF v režimu V86. Tato vlastnost zvyšuje rychlost provádění V86 aplikací eliminací řady přerušení aktivujících monitor V86 procesu.

Stránkování

- Režim IA-32 poskytuje operačnímu systému dvě úrovně transformací adres. V první, vyšší úrovni je **logická adresa** mechanismem segmentační jednotky transformována na **lineární adresu**. Ve druhé úrovni je **lineární adresa** stránkováním transformována na **fyzickou adresu**. Fyzická adresa ukazuje již přímo do fyzické paměti.
- Stránkovací jednotka v procesoru realizuje mechanismus virtuální paměti, který dovoluje programům používat větší kapacitu paměti, než je skutečně instalovaná fyzická paměť. Stránkování rovněž dovoluje procesům mít vlastní adresový prostor. Celá kapacita virtuální paměti je uložena po **stránkách** v externím zařízení (např. na disku). Fyzická paměť se po **rámcích** propůjčuje na dočasné uložení stránek virtuální paměti. Kapacita stránky a rámce je stejná.
- Propůjčování rámců fyzické paměti stránkám paměti virtuální provádí stránkovací jednotka v kombinaci s programovou podporou. Stránkovací jednotka provádí přepočet lineární adresy na číslo rámce, kde je momentálně stránka ve fyzické paměti umístěna. Pokud požadovaná stránka není v žádném z rámců, nastává tzv. **výpadek stránky.** Při rozpoznání výpadku stránky se generuje výjimka „Page Fault“, která aktivuje programovou rutinu zajišťující uvolnění rámce a přenos požadované stránky do uvolněného rámce. Stránkovací jednotka dále zajišťuje stránkovou ochranu na základě přístupových práv.

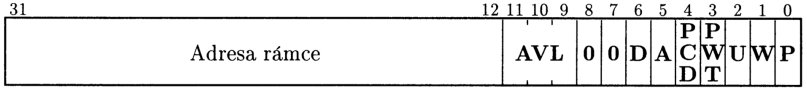
Mechanismus transformace lineární adresy na fyzickou



- Segmentování je povinnou částí zpracování adresy, naopak stránkování je volitelné. Stránkování je zapnuto jenom tehdy, je-li bit PG v CR0 nastaven na jedničku.
- Stránkování lze zapnout pouze v rámci chráněného režimu. Zapnuté musí být tehdy, potřebujeme-li realizovat virtuální paměť pomocí stránkování, spouštět více než jeden virtuální 8086 proces a provádět stránkově orientovanou ochranu paměti.
- Pro účely stránkování dělíme vstupní 32bitovou lineární adresu do tří částí. První část (10 bitů nejvyšších řádů) je **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 4k. 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 výjimku výpadku stránky.
- Vybraná položka stránkového adresáře ukazuje na začátek **stránkové tabulky** (**Page Table**), kterých může být až 1 024. Stránkové tabulky opět musí být uloženy od adres dělitelných 4 K. Položka stránkové tabulky, vybraná částí lineární adresy nazvanou tabulka, ukazuje na začátek 4 KB rámce ve fyzické paměti. Ukazatelem do vybraného rámce je poslední 12bitová část lineární adresy offset.
- Fyzická paměť je tedy rozdělena na 4 KB rámce, které začínají na adresách dělitelných 4 K a které se propůjčují pro uložení 4 KB stránek.
- Stránkový adresář a stránková tabulka jsou pole obsahující 1024 32bitových specifikátorů. Kapacita každé z těchto tabulek je právě stránka.

Specifikátor stránkového adresáře a tabulky



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

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

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

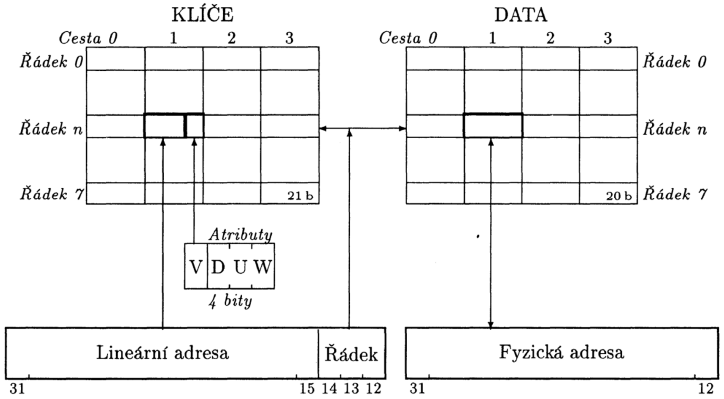
- **A (Accessed)** nastavuje procesor na jedničku před každým použití tohoto specifikátoru. Bit má stejný význam jako v popisovači segmentů
- **U (User Accesible)** je určen pro stránkovou ochranu paměti. Pracuje-li proces na úrovni oprávnění CPL=3, smí k této stránce přistupovat pouze tehdy, je-li U=1. 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)** 1 označuje, že obsah specifikátoru je platný a položku lze použít ke transformaci adresy. Je-li P=0, není obsah odpovídající stránky ve fyzické paměti. Potom se položka nepoužívá a s jejím obsahem (vyjma bitu P) může operační systém disponovat na jiné účely (např. uchovat do ní informaci o tom, kde na disku je stránka momentálně uložena). Pokus o přístup ke stránce, která má nastaveno P=0 (ve stránkovém adresáři nebo stránkové tabulce), je výpadek stránky a vyvolá výjimku 14.

- **PWT (Page Write-Through**) určuje způsob práce externí vyrovnávací paměti týkající se této stránky. Je-li PWT=1, provádí se zápis metodou „Write-Through“. Jde o způsob, ve kterém se zapisovaná slabika současně ukládá jak do vyrovnávací paměti, tak i do paměti. Je-li PWT=0, provádí se zápis metodou „WriteBack“, ve které se slabiky zapisují pouze do vyrovnávací paměti. Změněné položky se z vyrovnávací paměti přepíší do paměti až při jejím vyprazdňování.
- **PCD (Page Cache Disable**) vypíná činnost interní VP pro danou stránku. Je-li PCD=0,je splněna jedna z podmínek zapínajících IVP. Další podmínky tvoří signál KEN a bity CD a NW v registru CR0. Je-li PCD=1, je IVP vypnuta bez ohledu na ostatní podmínky.

TLB (Translation Look-Aside Buffer)

- Pro zrychlení převodu lineární adresy na fyzickou (nejde použít mechanismus neviditelné části v segmentovém registru při převodu logické adresy na lineární)
- Protože při transformaci lineární adresy na fyzickou se musí přistupovat ke dvěma tabulkám umístěným ve fyzické paměti (a tento přístup trvá poměrně dlouho), má procesor implementovánu TLB (Translation Look-Aside Buffer), která je vyrovnávací pamětí pro posledně transformované adresy včetně příznaků.
- V procesoru Pentium jsou integrovány dvě TLB. Jedna je součástí instrukční vyrovnávací paměti a druhá je uvnitř datové vyrovnávací paměti. TLB v datové VP je dvouportová. To jí dovoluje současně transformovat adresy ze dvou zdrojů. Instrukční TLB je jednoportová.
- Datová TLB je 4cestná asociativní paměť o 64 položkách pro 4KB stránky a 4cestná asociativní paměť o 8 položkách pro 1MB stránky. Instrukční TLB je 4cestná asociativní paměť o 32 položkách pro 4KB stránky a pro 1MB stránky ve 4KB násobcích. Četnost přístupů k jednotlivým položkám se sleduje 3bitovým algoritmem LRU implementovaným ve vyrovnávacích pamětech procesoru
- Obsahy TLB jsou chráněny paritou.

Struktura TLB v procesoru 80486



Virtuální paměť

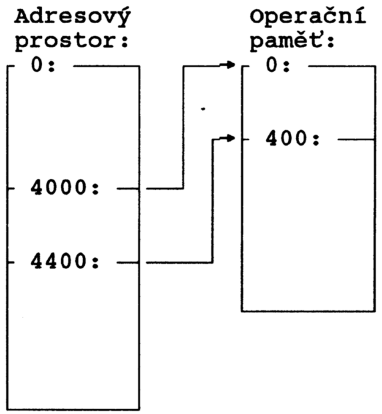
- Zopakování přístupu do paměti je základem virtualizace paměti. Pokusí-li se procesor pracovat s pamětí, která neexistuje (je virtuální), vyvolá MMU přerušení. Přerušovací podprogram neexistující paměť doplní a procesor zopakuje přístup do paměti. Protože paměť už nyní existuje, může činnost programu pokračovat nyní dále.

MMU by měla disponovat **ochranou paměti** a mechanismem **překladu adres.**

- **Ochrana paměti** – zajišťuje, aby proces nemohl přistupovat nekontrolovaně do paměti jiného procesu. MMU udržuje tabulky, ve kterých jsou informace, který úsek paměti patří kterému programu a jaká jsou přístupová práva pro procesy (jak vlastníka paměti, tak i cizí procesy). Pokud proces chce přistoupit k paměti, MMU ověří z těchto tabulek, zda má proces právo přístupu. Pokud ne, vyvolá se výjimka, v rámci které se zajistí vše potřebné (např. ukončení procesu, neboť chce použít paměť, kam nemá přístup).

- **Překlad adres** – umožňuje přiřadit libovolnému úseku v adresovém prostoru procesoru libovolné adresy ve fyzické operační paměti. Toto přiřazení se děje obyčejně pomocí tabulek, ve kterých jsou zaznamenány potřebné převody adres. Abychom odlišili obě adresy, používá se označení **logická adresa** pro adresu v adresovém prostoru procesoru a označení **fyzická adresa** pro adresu ve fyzické operační paměti. Teoreticky by překlad adres mohl fungovat pro libovolnou adresu (úsek paměti je jedna adresa), např. logická adresa 4000 by byla převedena na fyzickou adresu 0, 4001 na 1100, 4002 na 3856, atd. to by nebylo ale příliš praktické, převodní tabulky by byly příliš veliké. Zavádí se proto **stránkování**.

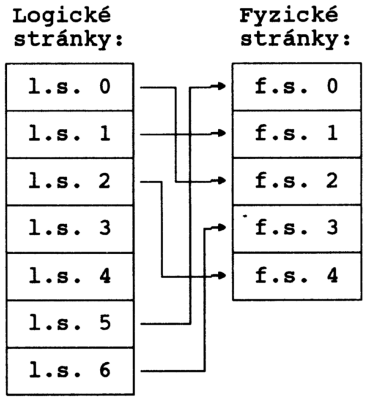
Překlad adres



Stránkování

- Požaduje, aby bloky, které se účastní překladu adres, byly složeny z tzv. stránek pevné velikosti (velmi často 4kB).
- Logický adresový prostor je tvořen stránkami (logickými), první z nich začíná na logické adrese 0, druhá na adrese <velikost_stránky>, třetí na adrese 2*<velikost_stránky>, atd.
- Stejně tak rozdělíme na jednotlivé stránky (fyzické - někdy tomu říkáme **rámce**, aby se to nepletlo) fyzickou operační paměť. První z nich začíná na fyzické adrese 0, druhá na adrese <velikost_stránky>, třetí na adrese 2*<velikost_stránky>, atd.
- Tabulky pro převod budou mít jednoduchou strukturu – pro převod jedné stránky na rámec bude v tabulce jedna položka.

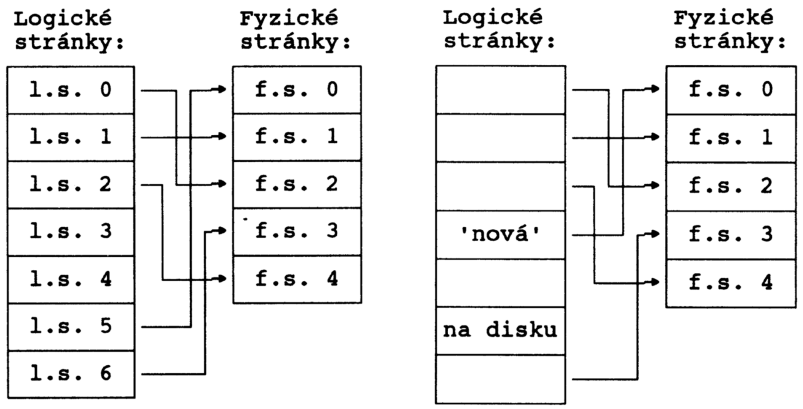
Stránkování – překlad adres



Princip virtuální paměti

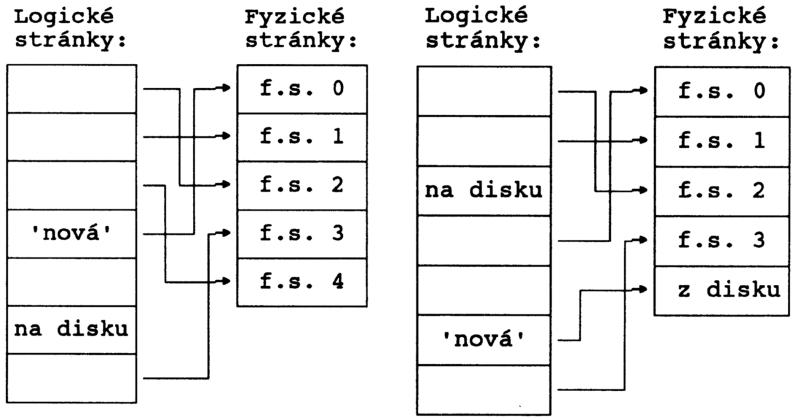
- Správce paměti má k dispozici odkládací prostor na disku.
- Pokud správce potřebuje někomu přidělit paměť a přitom není volný rámec, odebere rámec některému z procesů, uloží je do odkládacího prostoru a ve stránkových tabulkách vyznačí, že odpovídající stránka nemá přiřazen rámec a kde je odložen na disku.
- Přidělí rámec žádajícímu procesu - upraví stránkovací tabulku. Překlad adres zajistí správné adresování.
- Příklad: předpokládejme, že logická stránka číslo 3 je dosud volná a nějaký proces požaduje přidělení paměti; pro jednoduchost řekněme právě v rozsahu jedné stránky. Správce paměti přidělí logickou stránku číslo 3 a hledá odpovídající fyzickou stránku (rámec); zjistí však, že jsou všechny obsazeny. Zapíše proto obsah rámce číslo 0 do odkládacího prostoru a pozmění potřebným způsobem stránkové tabulky.

Odebrání stránky



- Pokud se původní vlastník pokusí pracovat se svou pamětí (kterou už nemá, neboť byla z fyzické paměti odložena na disk), detekuje MMU, že požadovaná adresa neexistuje. Vyvolá výjimku, tzv. **výpadek stránky**, tuto obslouží správce paměti.
- Najde odložený rámec na disku, a umístí jej někam do fyzické paměti (pokud je plná, musí uvolnit nějaký rámec – viz předcházející případ).
- Upraví stránkovací tabulky a procesor zopakuje přístup do paměti. Nyní už paměť existuje a proces může pokračovat v práci.
- Poznámka – na začátku stránka 5 měla přidělen rámec 0, nyní má přidělen rámec 4. Překlad adres ale zajistí, že vše funguje OK.

Přidělení nové stránky



Vybrání stránky pro odstranění

- Pokud potřebuje správce paměti uvolnit obsazený rámec ve fyzické paměti, nastává otázka, který má uvolnit.
- Měl by uvolnit ten rámec, který už nebude potřeba, nebo bude potřeba co nejpozději. Problém je, jak tuto informaci zjistit.
- Ukazuje se však, že je relativně rozumné vybrat rámec, který byl nejdéle nepoužitý – pokud nebyl dlouho použitý, je velká pravděpodobnost, že nebude použitý i v budoucnu – algoritmus **LRU** (**Least Recently Used**).
- Bohužel zjišťovat, který rámec byl nejdéle nepoužitý je také problém. Používá se proto jednodušší **pseudo-LRU** algoritmus, jehož výsledky se blíží výsledkům LRU.

Pseudo-LRU algoritmus

- Součástí stránkových tabulek je tzv. **Access bit** - speciální bit, který jednotka řízení paměti nastaví na jedničku při každém přístupu k odpovídající logické stránce (a tedy také k jí přiřazené stránce fyzické). To lze snadno zajistit bez jakékoli degradace výpočetního výkonu
- Operační systém pravidelně v určitém intervalu prochází stránkové tabulky a nuluje všechny „access“ bity. Nalezne-li již některý „access“bit nulový, znamená to, že tato stránka nebyla po celý interval použita (jinak by jednotka řízení paměti její „access“ bit nastavila), a je tedy dobrým kandidátem na odebrání.

Realizace virtuální paměti

- Na začátku práce má program k dispozici celý adresový prostor, ale žádnou fyzickou paměť (všechny logické stránky tedy mají nastaven přepínač, určující přiřazení fyzické stránky, na ‘neexistenci‘).
- Kdykoli se pokusí program použít nějakou adresu v logické stránce, která nemá a ani dosud neměla svůj ekvivalent v operační paměti (ze začátku tedy pokusí-li se program použít jakoukoli adresu), dojde k výpadku stránky a jednotka řízení paměti samozřejmě vyvolá přerušení. V něm se operační systém pokusí vyhledat nějakou dosud nevyužitou fyzickou stránku a přidělit ji programu. Pokud se mu to podaří, ukončí ihned přerušení a umožní tak programu pokračovat v přerušené práci - požadovaná adresa již ovšem má svůj ekvivalent v operační paměti.

- Ve chvíli, kdy operační systém nemá žádnou volnou fyzickou stránku, kterou by mohl programu přidělit, musí nějakou stránku uvolnit. K výběru fyzické stránky, která má být uvolněna, nejspíše využije strategii LRU. Původní obsah této fyzické stránky je zapotřebí zapsat do odkládací oblasti; obsah fyzické stránky se uloží na disk (tam, kde je zrovna místo), stránka se ‘staré‘ logické stránce odebere a ke ‘staré‘ logické stránce se v tabulce stránek zaznamená, kde přesně na disku je obsah stránky uložen. Pak již nic nebrání tornu, aby byla uvolněná fyzická stránka přidělena ‘nové‘ logické stránce.
- Jestliže potřebuje program použít data v logické stránce, která již dříve byla v operační paměti, ale fyzická stránka jí byla během času odebrána, proběhnou v podstatě opět předchozí dva body (nebo jen první z nich, je-li k dispozici nějaká volná fyzická stránka); jedinou změnou je to, že obsah přidělené stránky se před návratem do přerušeného programu inicializuje daty z disku, jejichž adresa byla uložena v tabulce stránek, když se fyzická stránka logické stránce odebírala podle minulého bodu.

Poznámky k virtuální paměti

- I při realizaci správce paměti pomocí virtuální paměti dochází k fragmentaci paměti. Fragmentuje se ale logický adresový prostor, který je obrovský (u současných procesorů Intel 64 TB), takže prakticky vždy budeme moci najít dostatečně velký blok pro přidělení procesu.
- Volné bloky paměti v logickém adresovém prostoru nemají svůj obraz ve fyzickém adresovém prostoru, takže nezabírají žádné místo v paměti.
- Při programování je třeba dát pozor na práci s velkými poli dat např. `double a[1000,1000]`. Pokud zpracováváme pole po řádcích a v paměti je pole uloženo po sloupcích, nevejde se do fyzické paměti z důvodu stránkování celý řádek. Při každém přístupu k prvku pole musí proběhnout výměna stránky -> velmi pomalé. Stačí zaměnit směr zpracování matice po sloupcích (v paměti budou prvky sloupce) a zpracování se výrazným způsobem zrychlí.

Systém ochran

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

1. Úrovněmi oprávnění
2. Privilegovanými instrukcemi

Úrovně oprávnění

- **Úroveň oprávnění (Privilege Level)** vyjadřuje kvantitu „důvěry“ poskytnuté určitému procesu. Např. procesy s nižší úrovní oprávnění nesmějí mít možnost zasahovat do procesů nebo dat s vyšší úrovní oprávnění.
- Procesor poskytuje **4** úrovně oprávnění. Nejvyšší úroveň oprávnění (tj. nejvíce „důvěry“) má úroveň číslo 0. Nejmenší úroveň oprávnění má úroveň číslo 3.
- Rozdělení těchto čtyř úrovní mezi jednotlivé vrstvy operačního systému může být následující:

úroveň 0 . . . jádro operačního systému (řízení procesoru, V/V operací),
úroveň 1 . . . služby poskytované operačním systémem (plánování procesů, organizace V/V, přidělování prostředků),
úroveň 2 . . . systémové programy a podprogramy z knihoven (systém obsluhy souborů, správa knihoven),
úroveň 3 . . . uživatelské aplikace.

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

- **DPL (Descriptor Privilege Level)** je uložen ve dvou bitech slabiky *přístupová práva* popisovače segmentu. Obsahuje úroveň oprávnění přidělenou obsahu segmentu.
- **CPL (Current Privilege Level)** je zapsán ve dvou nejnižších bitech selektoru CS (tj. v poli označeném RPL). Představuje momentální úroveň oprávnění přidělenou právě prováděnému procesu.
- **RPL (Requestor Privilege Level)** je uložen v bitech 0 a 1 selektoru segmentové ho registru a obsahuje úroveň oprávnění, kterou proces nabízí při přístupu k segmentu, jehož popisovač je adresován právě tímto selektorem. Procesor použije **numerické maximum** z čísel **RPL, CPL**. Z toho plyne, že proces může nabídnout pouze nižší úroveň oprávnění, než je jeho uložená v CPL.
- **EPL (Effective Privilege Level)** je numerické maximum CPL a RPL (tedy hodnota nižší úrovně oprávnění).

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 **CPL ≤ DPL**. Dále musí být úroveň oprávnění v RPL větší nebo rovna úrovni zpřístupňovaného segmentu (**RPL ≤ DPL**). Obě podmínky spojíme v jednu

  Max(CPL,RPL) ≤DPL neboli **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.

Předání řízení do instrukčního segmentu

- Proces smí předat řízení pouze do segmentu se stejnou úrovní oprávnění, musí tedy platit CPL = DPL, tj úroveň oprávnění běžícího procesu (CPL) se rovná úrovni oprávnění segmentu, do kterého se předává řízení (DPL).
- Zvláštní případ je předávání řízení do segmentu, který je přizpůsobitelný (ve slabice přístupových práv má nastaven bit C na jedničku). Tento segment musí mít stejnou nebo vyšší úroveň oprávnění, než má běžící proces, tj CPL ≥ DPL. Přizpůsobitelný segment se potom provádí na úrovni oprávnění volajícího procesu, tj. CPL. Hodnota CPL v registru CS zůstane zachována, i když je DPL menší.
- Bez použití dalších prostředků (volání podprogramů na jiné úrovni oprávnění pomocí brány) lze instrukcemi CALL, JMP a RETF předávat řízení přímo pouze do:
  - segmentu se stejnou úrovní oprávnění, jako je CPL
  - přizpůsobitelného (C=1) segmentu na stejné nebo vyšší úrovni oprávnění, tj. DPL ≤ CPL

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). Je to z důvodu bezpečnosti (aby např. jádro operačního systému nemohlo zavolat podprogram v uživatelském procesu)

Možnosti předání řízení

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

Brána (Gate)

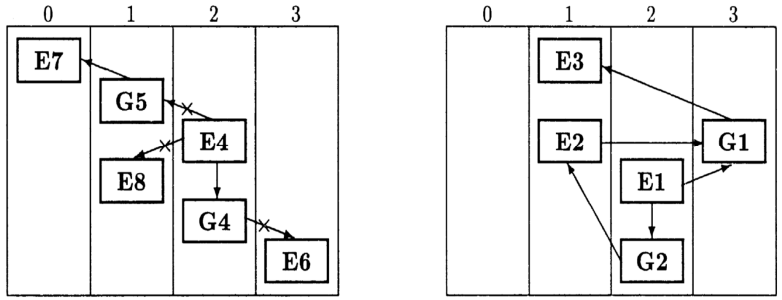
Brána (Gate) je popisovač uložený v tabulce popisovačů segmentů čtyř možných významů:

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)
4. brána zpřístupňující segment stavu procesu (Task Gate)

Brána pro předání řízení

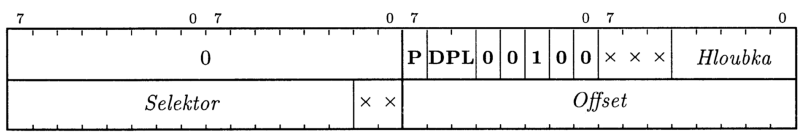
- Brána pro předání řízení do segmentu vyšší (nebo stejné) úrovně oprávnění (Call Gate) může být buď v GDT, nebo LDT. Jakmile se proces na tuto bránu odkáže, procesor zkontroluje, má-li proces dostatečnou úroveň oprávnění pro volání této brány. Při odkazu procesu na bránu platí stejná pravidla jako při odkazech procesu na data, tj. proces musí mít vyšší nebo stejnou úroveň oprávnění jako brána, tedy CPL ≤ DPL brány. Zároveň pole RPL popisovače odkazujícího se na bránu musí indikovat vyšší úroveň oprávnění než volaný segment, tj. RPL ≤ DPL brány. Tedy musí platit EPL ≤ DPL brány
- Musí ale také platit, že úroveň oprávnění volaného podprogramu musí být stejná nebo vyšší než volajícího, tj. CPL ≥ DPL podprogramu.
- Instrukcí JMP lze bránou předat řízení pouze do segmentu na stejné úrovni oprávnění nebo do přizpůsobitelného segmentu na vyšší úrovni oprávnění.
- Instrukcí CALL lze bránou předat řízení do libovolného segmentu na vyšší úrovni oprávnění.
- U obou instrukcí musí být navíc splněna podmínka MAX(CPL,RPL) ≤ DPL brány.

Příklady předání řízení



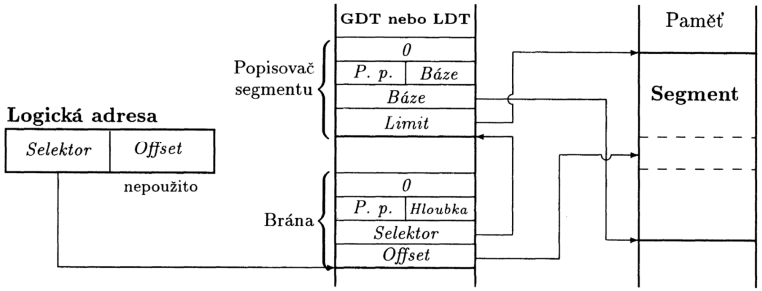
Přeškrtnuté šipky jsou nepovolená předání řízení
Při provádění instrukce RETF (vzdálený návrat do podprogramu) se kontroluje, je-li návrat veden do segmentu se stejnou nebo nižší úrovní oprávnění.

Formát popisovače brány pro předání řízení



- Při volání brány (rozuměj popisovače brány v tabulce popisovačů LDT nebo GDT) se z logické adresy uplatní pouze část selektor. Offset je ignorován.
- Z popisovače brány 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
- Brána je tedy jednoznačný vstupní bod do podprogramu, neboť offset není definován v uživatelském programu, ale v systémové tabulce v popisovači brány => zvýšená ochrana

Mechanismus předání řízení pomocí brány



Popis činnosti brány při předávání řízení

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

Přepínání zásobníků

- Pro zajištění vzájemné izolace procesů nabízí chráněný režim všem úrovním oprávnění každého procesu vlastní zásobníky. SS:ESP obsahuje vždy ukazatel do zásobníku úrovně oprávnění, na které proces právě pracuje. Systém limitů v datových segmentech hlídá případné přeplnění obsahu zásobníku. V okamžiku přeplnění by totiž došlo k zničení dat uložených bezprostředně „pod" zásobníkem.
- Zásobník je používán i pro předávání parametrů mezi volajícím modulem a volaným podprogramem tak, že volající před provedením instrukce CALL do zásobníku předávané parametry vloží např. tuto posloupností instrukcí:

  PUSH Par1
  PUSH Par2
  CALL Podprogram
- Brána pro předávání řízení zkopíruje parametrem, **hloubka (WC — Word Count**) zadaný počet 32bitových dvojslov ze zásobníku úrovně oprávnění volajícího modulu do zásobníku úrovně oprávnění volaného modulu. Maximální hodnota parametru hloubka je 31. Potřebujeme-li předat více než 31 parametrů, může být jeden z nich ukazatel na datovou strukturu obsahující další parametry.
-

Shrnutí pravidel pro předávání řízení

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

Privilegované instrukce

Jsou **dva typy** privilegovaných instrukcí.

1. Instrukce, které lze provést pouze s úrovní oprávnění 0 (na nejvyšší úrovni oprávnění) – tzv. **Privileged Instruction**
2. Instrukce, které lze provádět pouze od jisté úrovně oprávnění – tzv. **Trusted Instruction.** Jsou to 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**.

Privileged Instruction

- LGDT naplnění registru GDTR
- LIDT naplnění registru IDTR
- LLDT naplnění registru LDTR
- LTR naplnění registru TR
- MOV naplnění registru CRi nebo DRi
- CLTS nulování bitu TS (Task Switch) v registru CR0
- INVD vyprázdnění IVP bez uložení
- WBINVD vyprázdnění IVP s uložením
- INVLPG vyprázdnění TLB
- RSM návrat z SMM,
- HLT zastavení procesoru

Pokud je použita některá tato instrukce na jiné úrovni oprávnění než 0, generuje se výjimka 13

Trusted Instruction

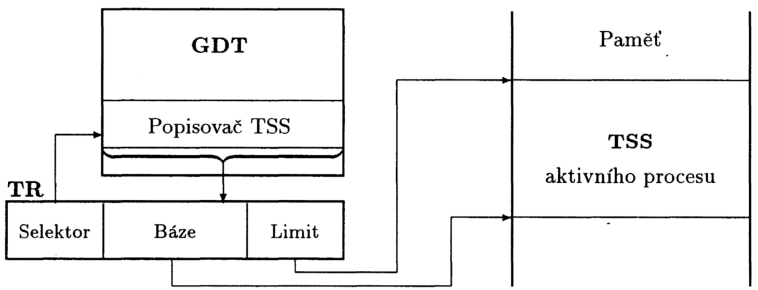
- IN, INS čtení ze V/V brány
- OUT, OUTS zápis na V/V bránu
- STI, CLI změna příznaku IF

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

Přepínání procesů

- V multitaskingových operačních systémech je důležitý pojem přepínání procesů. Architektura IA-32 podporuje přepínání procesů technickými prostředky mikroprocesoru
- O každém procesu (aktivním i neaktivním) jsou v paměti ve speciálním segmentu uloženy všechny potřebné informace (tzv. **kontext** neboli **stav procesu**). 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 (obdoba činnosti přerušovacího podprogramu).
- 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 **TR (Task Register)**. Na každý TSS ukazuje právě jeden popisovač TSS

Zpřístupnění TSS aktivního procesu pomocí registru TR



TR obsahuje viditelnou a neviditelnou část (obdoba segmentových registrů)

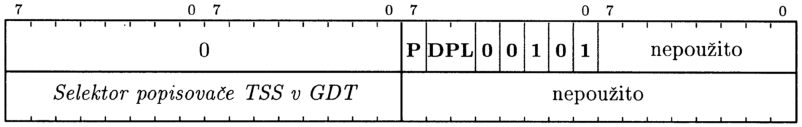
Obsah TSS segmentu

31	0	
Offset mapy V/V bran	0000000000000000 T	100
0000000000000000	Selektor LDT	96
0000000000000000	Selektor GS	92
0000000000000000	Selektor FS	88
0000000000000000	Selektor DS	84
0000000000000000	Selektor SS	80
0000000000000000	Selektor CS	76
0000000000000000	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
0000000000000000	SS pro úroveň 2	24
ESP pro úroveň 2		20
0000000000000000	SS pro úroveň 1	16
ESP pro úroveň 1		12
0000000000000000	SS pro úroveň 0	8
ESP pro úroveň 0		4
0000000000000000	Zpětný ukazatel	0

Velikost segmentu (limit) není obecně stanovena. Maximální délka TSS je 4 GB a minimální je 104 slabik.

- **Zpětný ukazatel** obsahuje selektor TSS přerušeného procesu. Ukazatel se použije při obnovení přerušeného procesu instrukcí RET.
- **Ukazatele zásobníků** zahrnují momentální stavy ukazatelů zásobníků pro úrovně oprávnění 0, 1 a 2.
- **Řídicí registry** vztahující se k procesu - registr CR3 s bází stránkového adresáře, registr EIP čítače instrukcí a příznakový registr EFLAGS. Nejsou zde registry vztahující se k procesoru (CR0 apod.).
- **Všeobecné registry procesoru**.
- **Segmentové registry procesoru**.
- **Selektor LDT** obsahuje selektor položky GDT, který specifikuje LDT náležející procesu.
- **T (Trap)** je bit, který, je-li nastaven na jedničku, způsobí v okamžiku přepnutí na tento proces ladicí výjimku 1.
- **Offset mapy přístupných V/V bran** ukazuje na začátek bitové mapy uvnitř tohoto segmentu stavu procesu. Báze segmentu ukazuje na položku Zpětný ukazatel a offset musí být minimálně 104 a maximálně 0DFFFh.

Formát popisovače brány zpřístupňující TSS v GDT, LDT nebo IDT



- Brána zpřístupňující TSS (Task Gate) může být umístěna v GDT, LDT nebo IDT (tabulka pro přerušovací systém). Protože popisovač TSS smí být umístěn pouze v GDT, zprostředkovává brána přepínání procesů i pomocí LDT

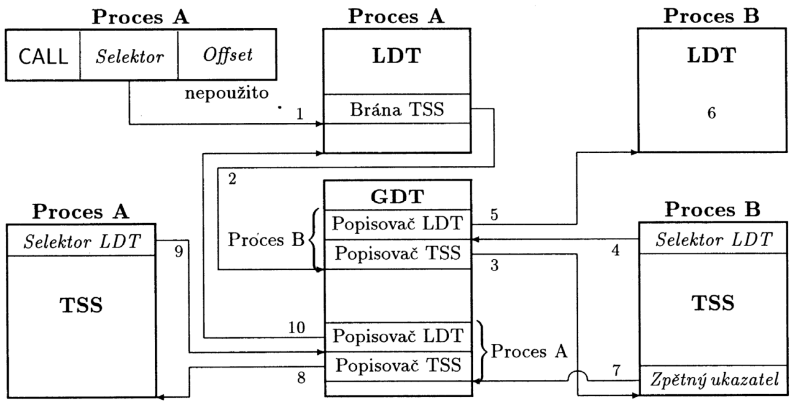
Operace přepnutí procesu sestává z těchto elementárních akcí:

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:

1. vzdáleným JMP nebo CALL, jehož selektor ukazuje na popisovač TSS nového procesu v GDT. Offsetová část adresy se ignoruje.
2. vzdáleným JMP nebo CALL, jehož selektor ukazuje na bránu (zpřístupňující TSS nového procesu) v GDT, LDT nebo IDT. Offsetová část adresy se ignoruje.
3. přerušením, jehož přerušovací vektor ukazuje na bránu (zpřístupňující TSS nového procesu) v IDT.
4. 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.

Přepnutí procesů vyvolané instrukcí CALL pomocí brány

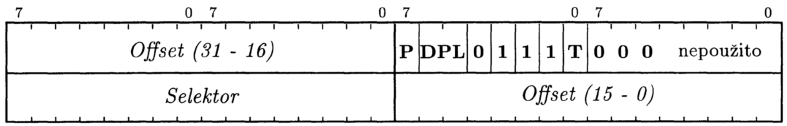


Výjimky a přerušení

- Výjimky a přerušení jsou v chráněném režimu obsluhovány diametrálně odlišně než v reálném režimu. Zásadní změnou je jiný formát tabulky přerušovacích vektorů a zavedení nové struktury IDT, podobající se GDT.
- Tabulka popisovačů segmentů obsluhy přerušení **IDT (Interrupt Descriptor Table)** obsahuje až 256 popisovačů. Adresa IDT je uložena ve speciálním registru **IDTR (Interrupt Descriptor Table Register)**. Registr obsahuje 32bitovou bázi a 16bitový limit IDT. Plní se privilegovanou instrukcí LIDTR. V IDT se vyskytují pouze tyto tři typy popisovačů:
  - brána zpřístupňující TSS
  - brána pro maskující přerušení **(Interrupt Gate)**
  - brána pro nemaskující přerušení **(Trap Gate)**

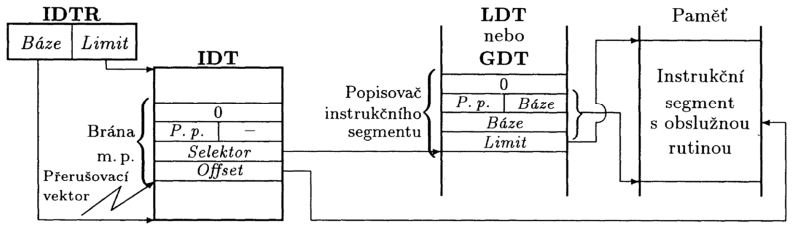
- Odkaz na bránu zpřístupňující TSS způsobí přepnutí procesů, zatímco obsluha dalších dvou typů je v rámci stejného procesu bez přepnutí.
- V okamžiku přerušení je vybrán jeden z 256 popisovačů IDT podle obsahu přerušovacího vektoru (v intervalu 0 až 255). V IDT nemusí být využito všech 256 položek. Nepoužité položky mají ve slabice přístupová práva nulu (P=0) nebo limit IDT je menší než 256 položek.

Formát popisovače brány přerušení v IDT



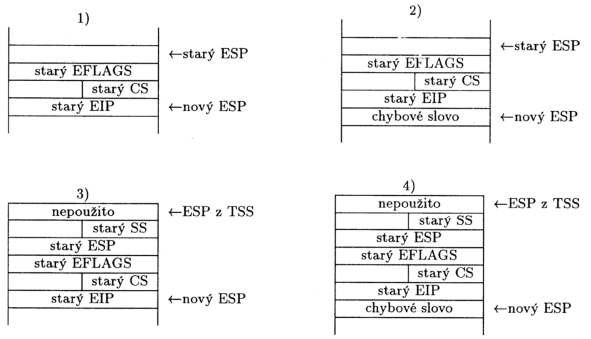
- Je-li bit T=1, jde o bránu pro nemaskující přerušení **(Trap Gate),** je-li T=0, jde o bránu pro maskující přerušení **(Interrupt Gate).** Položka *Selektor* ukazuje na popisovač instrukčního segmentu v GDT nebo LDT. V tomto instrukčním segmentu je od relativní adresy *Offset* uložen podprogram pro obsluhu právě vyvolaného přerušení
- Při povolování přístupu k podprogramu obsluhující přerušení platí stejná pravidla jako pro zpřístupňování instrukčního segmentu branou předávající řízení: CPL ≤ DPL brány přerušení a zároveň CPL ≥ DPL rutiny obsluhující přerušení. Přerušením tedy nelze předat řízení na nižší úrovni oprávnění.

Obsluha výjimky a přerušení branou v IDT



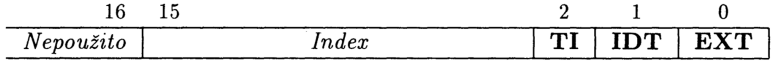
Návrat z obsluhy přerušení provádí instrukce IRET. Na rozdíl od instrukce RET ze zásobníku vyzvedne ještě obraz registru EFLAGS. Pole IOPL v registru EFLAGS obrazem přepíše pouze tehdy, je-li CPL nulové. Příznak IF přepíše pouze při CPL ≤ IOPL.

Možné obsahy zásobníku při aktivaci obsluhy přerušení



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

Chybové slovo



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

- **IDT** indikuje, že index ukazuje do IDT (nikoli do GDT nebo LDT podle TI).
- **EXT (External)** je nastaven tehdy, bylo-li přerušení způsobeno vnější událostí bez zavinění procesu (např. přerušení 10: vnější přerušení přes bránu zpřístupňující TSS vyvolalo pokus o přepnutí na proces, který má TSS s chybným obsahem).

Rezervované výjimky

Výjimky generované procesorem dělíme do tří kategorií:

- **Fault** do zásobníku uloží CS:EIP ukazující **na instrukci**, která způsobila přerušení.
- **Trap** do zásobníku uloží CS:EIP ukazující **za instrukci** (na následující instrukci), která přerušení způsobila. Pokud instrukce, která způsobila výjimku, je instrukcí předávání řízení, potom CS:EIP uložené v zásobníku ukazuje na cílové místo (nikoli na následující instrukci za instrukcí způsobující výjimku).
- **Abort** způsobí, že v procesu nelze pokračovat a musí být násilně ukončen. Výjimka tohoto typu nastává při chybách systémových tabulek, či při chybách technického vybavení.

Výjimky generované procesorem

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

- **INT 0 Chyba dělení (Divide Error)** - výjimka 0 nastane při dělení nulou v instrukcích DIV a DIV.
- **INT 1 Ladicí výjimky (Debug Exceptions)** - ladicí výjimku generuje ladicí systém procesoru. Existuje pět příčin, pro které procesor tuto výjimku generuje:
  - při čtení/zápisu z/do paměti byl detekován ladicí bod (Trap),
  - při výběru instrukce byl detekován ladicí bod (Fault),
  - po provedení instrukce v krokovacím režimu (Trap),
  - při přepnutí na proces mající v TSS T=1 (Trap),
  - nedovoleným přístupem k ladicím registrům při G=1 (Fault).
- **INT 3 Ladicí bod (Breakpoint)** - výjimka 3 se používá společně s výjimkou 1 v ladicích programech. Výjimka číslo 3 se vygeneruje po dekódování speciální jednoslabikové instrukce INT3 (s operačním kódem 0CCh). Tuto instrukci ladicí program vloží na to místo, na kterém se má běžící program zastavit. Výjimka uloží do zásobníku obsah CS:EIP ukazující na slabiku bezprostředně za touto instrukcí.

- **INT 4 Přeplnění (Overflow)** - je-li nastaven příznak OF=1 (Overflow Flag) a potom je provedena instrukce INTO (Interrupt on Overflow), generuje se výjimka 4. Příznak OF je nastaven během některých instrukcí ze skupiny aritmetických operací při zjištění přeplnění (výsledek je mimo rozsah zobrazení).
- **INT 5 Kontrola mezí (Bound Check)** - výjimku vyvolá instrukce BOUND, je-li její operand mimo zadané meze
- **INT 6 Chybný operační kód (Invalid Opcode)** - tuto výjimku generuje prováděcí jednotka procesoru, nedokázala-li rozpoznat operační znak instrukce, nalezla-li chybnou kombinaci operačního znaku a operandu (např. použití registru v instrukci vzdáleného skoku), nebo při použití prefixu LOCK před instrukcí, se kterou se prefix použít nesmí. Chybný operační kód se pozná až při provádění instrukce, nikoli už při jejím výběru. CS:EIP uložené do zásobníku ukazuje na instrukci, která přerušení způsobila.

- **INT 7 Nedostupnost koprocesoru (Processor Extension Not Available)** - výjimku mohou vyvolat instrukce obsluhující koprocesor v závislosti na indikátorech zaznamenaných v registru CR0
- **INT 8 Dvojnásobný výpadek segmentu (Double Fault)** - výjimka se generuje tehdy, vyskytne-li se výjimka v okamžiku předávání řízení obslužné rutině předchozího přerušení (např. je-li instrukční segment obsluhující přerušení označen P=0, tak se při pokusu o zpřístupnění generuje výjimka 11, a je-li segment s rutinou obsluhující výjimku 11 označen rovněž P=0, generuje se výjimka 8. Vyskytne-li se výjimka i během provádění obsluhy výjimky 8, procesor se zastaví (z tohoto stavu ho vyvede bud' RESET, nebo NMI). Výjimka ukládá do zásobníku nulové chybové slovo.
- **INT 9 - Rezervováno**

- **INT 10 Chybný TSS (Invalid Task State Segment)** - výjimka nastává při pokusu o přepnutí na proces mající chybný TSS.
- **INT 11 Výpadek segmentu (Segment Not Present)** - výjimka nastane tehdy, je-li detekováno P=0 v těchto okamžicích:
  - při plnění registrů CS, DS, ES, FS nebo GS,
  - při plnění registru LDTR instrukcí LLDT,
  - při přístupu k bráně.

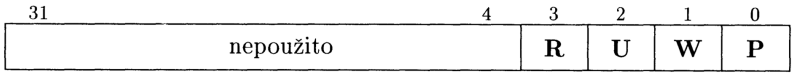
  V chybovém slově je selektor segmentu, který vyvolal přerušení.
- **INT 12 Výpadek segmentu se zásobníkem (Stack Fault)** - výjimka je generována v těchto případech:
  - pokud některá z instrukcí, odkazujících se na registr SS (POP, PUSH, ENTER, LEAVE, CALL atd.), způsobí překročení limitu segmentu se zásobníkem
  - pokud při plnění registru SS je selektor popisovače označen P=0.

- **INT 13 Obecná chyba ochrany (General Protection Fault)** - výjimka se vyvolá při chybě detekované systémem ochran v těch případech, ve kterých nepatří pod žádnou z dosud diskutovaných výjimek. Může nastat při těchto pokusech:
  - překročení limitu segmentu,
  - předání řízení (skok) do datového segmentu,
  - zápis do segmentu určeného pouze ke čtení nebo ke spuštění,
  - čtení segmentu určeného pouze ke spuštění,
  - použití neplatného selektoru,
  - přístup k popisovači, který je za limitem příslušné tabulky,
  - přepnutí na aktivní úlohu,
  - provedení privilegované instrukce při CPL>O,
  - překročení limitu délky instrukce, který je 15 slabik.
  - zápis jedničky do vyhrazených bitů CR4,
  - jakékoli jiné porušení pravidel systému ochrany.

- **INT 14 Výpadek stránky (Page Fault)** - výjimku generuje stránkovací jednotka (je-li zapnuta nastavením bitu PG=1 v CR0), pokud při transformaci lineární na fyzickou adresu nastala jedna z těchto událostí:
  - právě aktivní proces neměl dostatečnou úroveň oprávnění pro přístup k požadované stránce,
  - položka jedné z transformačních tabulek, použitá pro přepočet adresy, signalizovala, že požadovaná stránka není momentálně v paměti (má nastaveno P=0).

  Nastala-li jedna z těchto událostí, je v CR2 uložena lineární adresa, která vyvolala přerušení, a v zásobníku je uloženo chybové slovo. Na rozdíl od výjimek 10 až 13 má zde chybové slovo jiný tvar. Indikuje, vznikla-li výjimka na základě výpadku stránky nebo porušením přístupových práv, zda chybná operace byla čtení nebo zápis a zda šlo o uživatelský nebo privilegovaný přístup.

Chybové slovo výjimky 14



- **P (Present)** je výsledkem logického součinu bitů P obou transformačních tabulek, použitých k výpočtu této fyzické adresy. Je-li nulový, měla jedna z položek P=0.
- **W (Write)** indikuje, o jakou operaci byl procesor požádán. V okamžiku vzniku přerušení se prováděl zápis (W=1) nebo čtení (W=0).
- **U (User Level)** je-li nastaven, bylo požádáno o přístup ke stránce s úrovní oprávnění CPL=3. Je-li bit U nulový, bylo požádáno o přístup ke stránce s CPL<3.
- **R (Reserved)** je-li nastaven, byla výjimka generována detekcí jedničky v některém z rezervovaných bitů položky stránkového adresáře nebo stránkové tabulky, která je označena jako přítomná.

- **INT 16 Chyba koprocesoru (Processor Extension Error)**
- **INT 17 Kontrola zarovnání (Alignment Check)** - výjimka je vyvolána při současném splnění čtyř podmínek:
  - Přerušení musí být povoleno nastavením bitu AM=1 v registru CR0.
  - Přerušení musí být povoleno nastavením příznaku AC1 v příznakovém registru EFLAGS
  - Proces běží na nejnižší úrovni oprávnění (CPL=3).
  - Procesor rozpoznal pokus o přístup k operandu v paměti, který nezačíná na adrese dělitelné délkou zpřístupňovaného operandu (viz tabulka)

  Kontrola při AC1 se provádí pouze pro ty procesy, které pracují na úrovni oprávnění CPL=3. Má-li proces CPL<3, je nastavení AC ignorováno a výjimka 17 není v žádném případě generována. Výjimka 17 předává 32bitové chybové slovo s nulovým obsahem.
- **INT 18 Chyba v procesoru (Machine Check Exception)** - výjimka je dostupná pouze na procesorech Pentium a nemusí být v novějších typech. Oznamují se jí chyby parity a jiné chybové stavy technického vybavení.

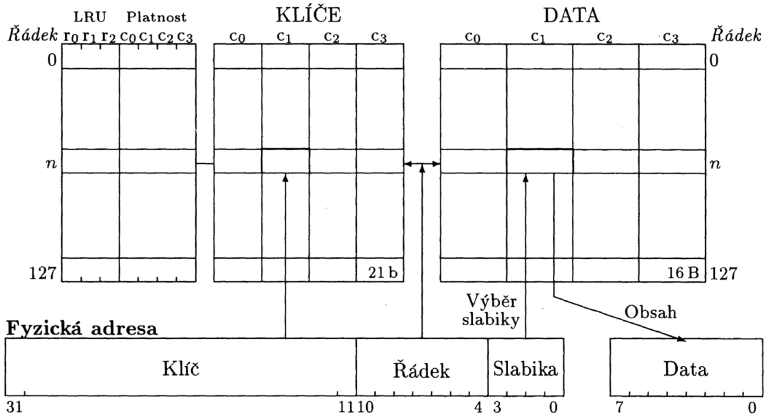
Tabulka hranic zarovnání objektů v paměti

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

Interní vyrovnávací paměť - IVP

- Oddělené IVP pro data a instrukce
- Do datové IVP lze i zapisovat
- Do instrukční zapisovat nelze, lze pouze číst
- Zvyšuje výkon systému – procesor nemusí čekat na pomalou operační paměť
- Zápis do datové paměti může probíhat dvojím způsobem:
- **Write-Through** - zápis se provádí souběžně do položky IVP a operační paměti. Tato metoda je užitečná např. při implementaci grafické video-paměti, kde se musí průběžně aktualizovat její obsah proto, aby i obraz byl aktuální.
- **Write-Back** - zápis se provádí pouze do vyrovnávací paměti. V tomto režimu se snižuje zatížení sběrnice eliminací řady ne nezbytných zápisů do paměti. Obsah paměti se aktualizuje až později při vyprazdňování IVP operací Write-Back.

Struktura IVP 80486



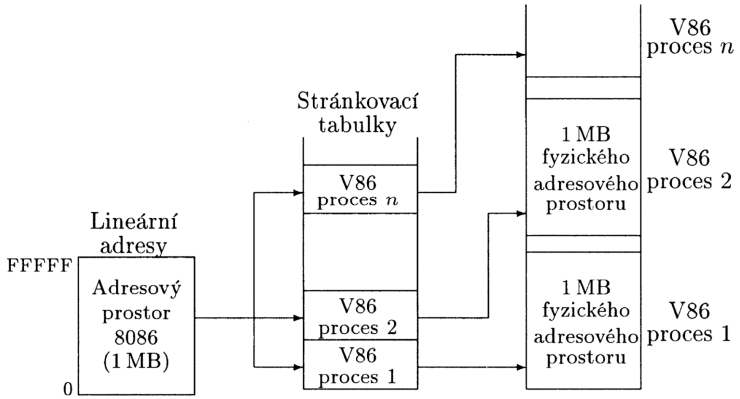
Režim Virtual 8086 - V86

- Režim virtuální 8086 umožňuje v rámci chráněného režimu spouštět programy určené pro procesory 8086 a 8088 a pro reálné režimy ostatních procesorů Intel, aniž by bylo nutné modifikovat jejich kód.
- Režim V86 se zapíná pro konkrétní úlohu a nevylučuje se tím možnost víceúlohového zpracování.
- Z pohledu uživatele umožňuje režim virtuální 8086 provozovat jeden nebo více virtuálních procesorů 8086 uvnitř jednoho.
- Virtuální 8086 proces má vlastní TSS, je mu přiřazen první 1 MB lineárního adresového prostoru, přístup procesu k V/V branám je kontrolován mapou přístupných V/V bran atd.
- Proces V86 uvnitř operačního systému je tvořen dvěma podprocesy: vlastním V86 procesem a řídicím procesem, běžícím v chráněném režimu. Řídicí proces se aktivuje výjimkami a přerušeními, které vznikly činností V86 procesu, příp. směřují zvnějšku do V86 procesu.

- V procesoru 8086 ochrany nejsou implementovány. Libovolně lze přecházet z jednoho segmentu do druhého, přistupovat ke všem částem paměti, ke všem V/V branám atd.
- Poněvadž v chráněném režimu může být v paměti umístěno více procesů, musí být od sebe izolovány. Tedy je nutno také izolovat proces virtuální 8086 od procesů ostatních. Dovnitř procesu, podle zvyklostí 8086, prostředky ochrany nezasahují, hlídají pouze ty akce, které by mohly ovlivnit ostatní procesy.
- Fakt, že proces pracuje v režimu virtuální 8086, sděluje, že má přidělenu úroveň oprávnění CPL=3. Z toho plyne, že v procesu virtuální 8086 nelze používat ty privilegované instrukce, které lze provádět pouze na úrovni oprávnění CPL=0 (většinu z nich stejně procesor 8086 nezná).

- V okamžiku přerušení procesu V86 není předáno řízení podle zvyklostí 8086 na tabulku přerušovacích vektorů, uloženou od adresy 0000:0000, ale přepne se do chráněného režimu a provede se obsluha podle příslušného popisovače v IDT. Je-li popisovačem některá z bran pro přerušení, musí ukazovat na proces s CPL=0. Zpět do režimu V86 se řízení vrací až instrukcí RET.
- Stránkování v režimu V86
  - 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.. Faktu, že více procesů může přistupovat k jedné stránce, lze naopak využít ke sdílení části paměti (programy v pamětech ROM nebo re-entrantní části procesů) těmito procesy.

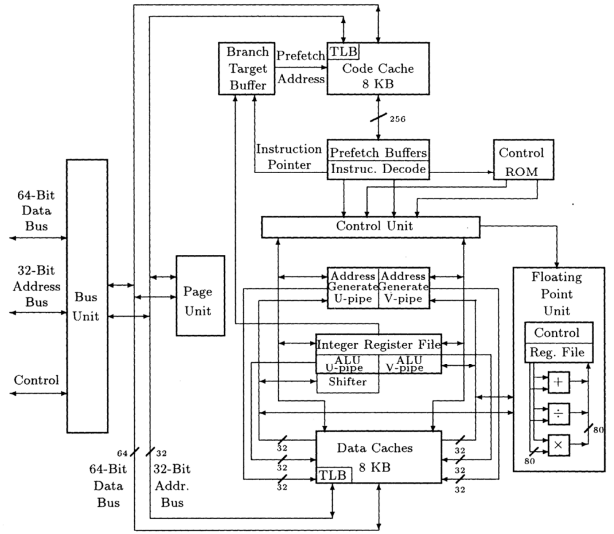
Stránkování v režimu V86 –více procesů V86



Architektura Pentium – rozšíření vůči 80486

- superskalární architektura
- dynamické předvídání skoků
- zřetězená FPU
- zkrácení doby provádění instrukcí
- oddělené datové a instrukční vnitřní vyrovnávací paměti
- 64bitová datová sběrnice
- zřetězování cyklů sběrnice
- adresová parita
- vnitřní kontrola parity
- kontrola správné funkce znásobením čipů s procesorem
- sledování provádění
- monitorování výkonnosti

Blokový diagram Pentia



Zřetězené zpracování instrukcí

Fáze zpracování:

PF - Prefetch - Výběr instrukce

D1 - Instruction Decode - Dekódování instrukce

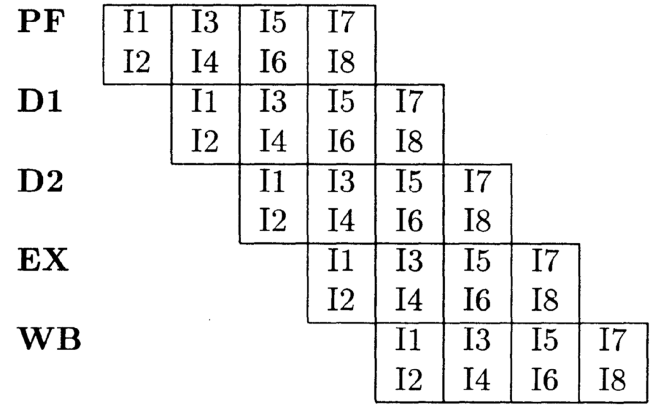
D2 - Address Generate - Generování adresy

EX - Execute - Provedení instrukce

WB - Write Back - Dokončení instrukce

Tyto dvě zřetězené fronty se nazývají „u“ a „v“. Proces souběžného zpracovává ní instrukcí se nazývá „párování“. Ve zřetězené frontě „u“ lze provádět libovolnou instrukci, zatímco ve frontě „v“ lze provádět pouze jednoduché instrukce, popsané v pravidlech pro párování instrukcí. Jednoduše řečeno — párovat lze instrukce podobné RISCovým. Instrukce vložená do fronty „v“ je vždy instrukcí následující za instrukcí ve frontě „u“.

Zřetězené zpracování instrukcí



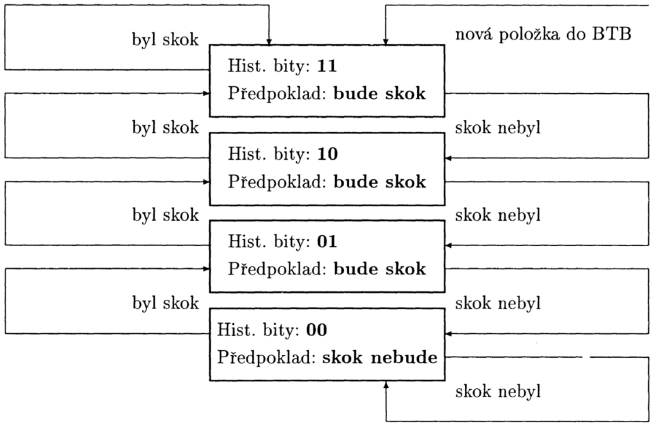
Párování instrukcí

- Obě instrukce v páru musí být „jednoduché“
- Mezi instrukcemi v páru nesmí být vztah „Čtení až po zápisu“ nebo „zápis až po čtení“
- Žádná z instrukcí nesmí mít výpočet adresy složen ze dvou částí: z přímé hodnoty a zároveň z přírůstku
- Instrukce s prefixy (vyjma OF před podmíněným skokem) lze provádět pouze ve frontě „u“
- „Jednoduché“ instrukce jsou ty, které nevyžadují mikrokód (jsou realizovány zapojením logických obvodů) a provedou se během jednoho hodinového cyklu. Výjimkou jsou instrukce aritmeticko-logické jednotky (ALU) *mem,reg* a *reg,mem*, které se provádějí ve dvou nebo třech taktech a jsou považovány za jednoduché.

Předvídání skoků

- Procesor Pentium má v sobě implementován mechanismus předvídání výsledku podmíněných skokových instrukcí.
- Algoritmus je založen na následujícím faktu: vždy po provedení těla cyklu se podmíněnou skokovou instrukcí rozhoduje o jeho opětném provedení. S významně větší pravděpodobností nastává situace, že při zopakování téže podmíněné skokové instrukce bude její výsledek stejný (tj. tělo cyklu se bude opakovat vícekrát a jenom jednou se cyklus opustí).
- Návrháři do procesoru zahrnuli paměť adres skoků (**Branch Target Buffer - BTB**). Do této paměti procesor ukládá cílové adresy ze skokových instrukcí (získaných fází DI), které provedly skok, spolu s 2bitovým záznamem historie skoků.
- Při výběru instrukce se testuje obsah BTB na shodu s adresou vybírané instrukce. Pokud se adresa v BTB najde, zkoumá se obsah bitů historie. Předpokládá-li se skok, pokračuje se výběrem následujících instrukcí počínaje instrukcí, na kterou ukazuje operand skokové instrukce. Pokud se ve fázi provedení skokové instrukce zjistí, že předpověď byla špatná, musí se fronta předvybraných instrukcí vyprázdnit a výběr se zahajuje znovu ze správné adresy.
- Historické bity představují automat. Vždy po provedení skoku se bity aktualizují. Pouze v kombinaci 00 se předpokládá, že skok nenastane.

Paměť adres skoků - BTB

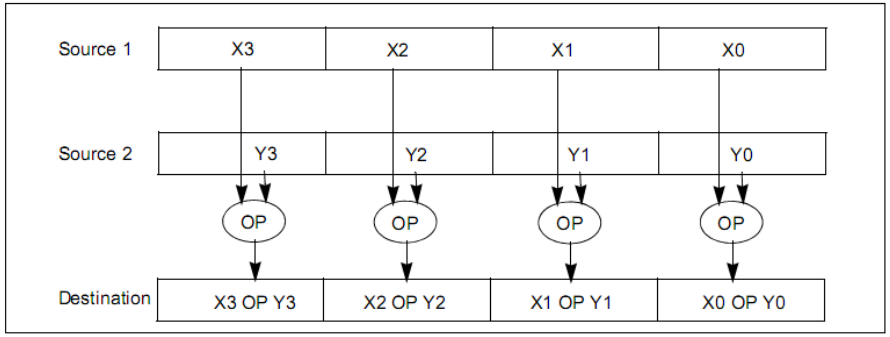


Rozšíření instrukčního souboru MMX, SSEx

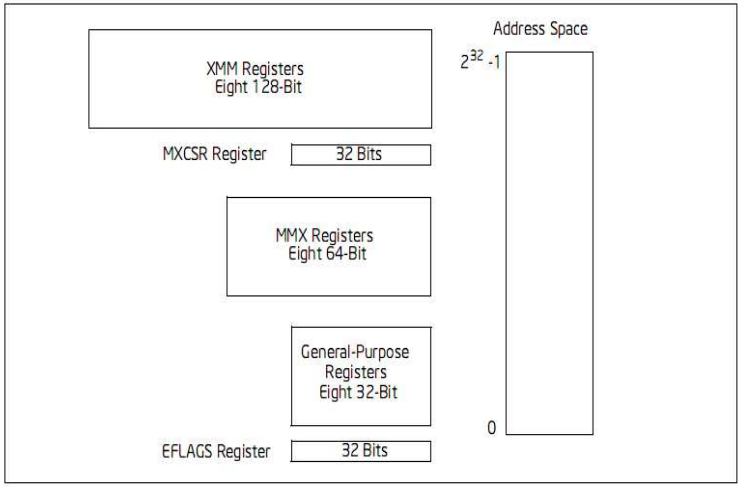
- Jde o rozšíření instrukčního souboru procesoru o instrukce typu **SIMD – Single Instruction Multiple Data.** Každá z těchto rozšíření poskytuje skupinu instrukcí, které provádějí SIMD operaci na zhuštěných (packed) celočíselných datech nebo datech s plovoucí desetinnou tečkou, které jsou umístěny v 64b registrech MMX nebo 128b registrech XMM.
- MMX – byly poprvé zavedeny u Pentia MMX a Pentia II. Jsou užitečné pro aplikace, které operují nad celočíselnými poli a proudy celočíselných dat – multimediální aplikace
- SSE – poprvé zavedeny u Pentia III. Pracují nad zhuštěnými FP čísly v XMM registrech a celočíselnými daty v MMX registrech. Použití především ve 3D grafice, rendering, dekódování a enkódování videa.

- SSE2 – zavedeno u Pentia 4 a Xeon, jako SSE, le pracuje i nad celočíselnými daty v XMM registrech a zhuštěnými daty ve FP v XMM registrech
- SSE3 – 13 instrukcí - zavedena u Pentií 4 s tzv. hyper-threading Technology (chová se jako nepravý vícejádrový procesor).
- SSE4 – 54 nových instrukcí, 47 označeno jako SSE4.1, 7 jako SSE4.2

Model provádění SIMD instrukcí



Registry pro SIMD instrukce

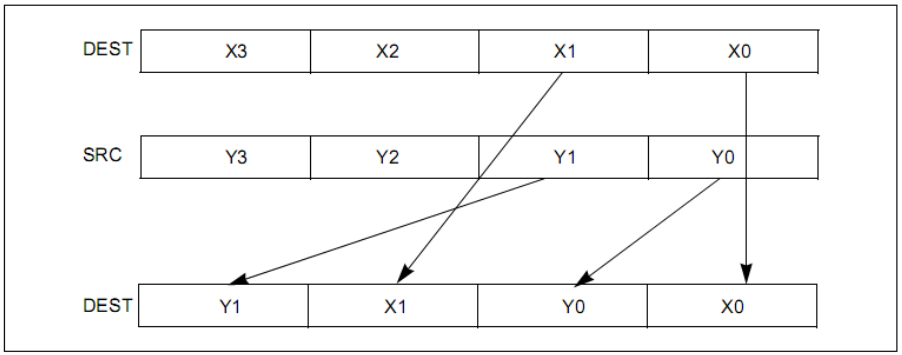


MXCSR – 32b stavový a řídící registr pro SIMD FP instrukce

Příklad MMX instrukcí

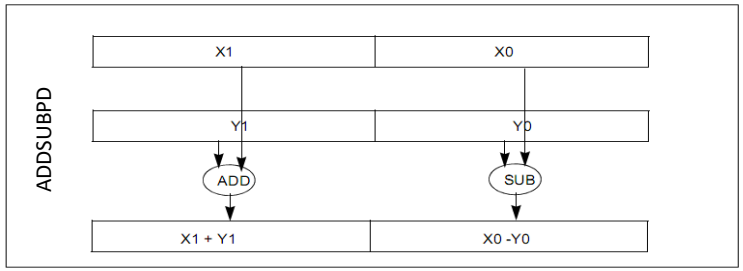
- Instrukce pro přesun dat v MMX reg., instrukce konverzní, aritmetické, porovnání, logické, posunu a rotace.
- MMX Data Transfer Instructions – přesun dat mezi MMX registry
  - MOVD Move doubleword
  - MOVQ Move quadword
- Conversion Instructions - provádí převody mezi zhuštěným a nezhuštěným tvarem dat
  - PACKSSWB Pack words into bytes with signed saturation
  - PACKSSDW Pack doublewords into words with signed saturation
  - PACKUSWB Pack words into bytes with unsigned saturation.
  - PUNPCKHBW Unpack high-order bytes
  - PUNPCKHWD Unpack high-order words
  - PUNPCKHDQ Unpack high-order doublewords
  - PUNPCKLBW Unpack low-order bytes
  - PUNPCKLWD Unpack low-order words
  - PUNPCKLDQ Unpack low-order doublewords

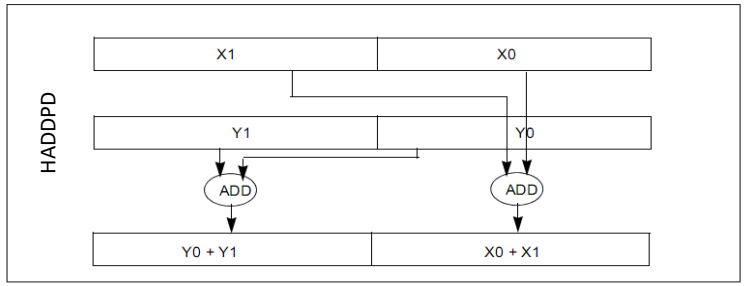
Příklad instrukce UNPCKLPS



SSE instrukce UNPCKLPS (unpack and interleave low packed single-precision floating-point values) – provádí prokládané rozbalení dolních FP dat v jednoduché přesnosti

Příklady SSE3 instrukcí ADDSUBPD a HADDPD





64-bitové rozšíření IA-32

- **x86-64 (též AMD64, EM64T, IA-32E)** je v informatice označení generace 64bitových procesorů pro počítače IBM PC kompatibilní.
- Procesor je zpětně kompatibilní s 32bitovou (viz IA-32) a 16bitovou architekturou (viz x86), a proto se na IBM PC prosadil.
- Hlavní výhodou nastupujícího 64bitového režimu je odstranění limitu přímé dostupnosti paměti nad 4 GiB operační paměti RAM.

- Architektura AMD64 byla vyvinuta společností AMD jako alternativa k radikálně odlišné architektuře IA-64, kterou se snažila prosadit dvojice Intel a Hewlett-Packard.
- AMD64 byl evoluční krok CISC architektury podobný nástupu 32bitové architektury IA-32 na rozdíl od architektury IA-64 (procesor Itanium), která byla pokusem o prosazení nové zpětně nekompatibilní 64bitové architektury typu RISC.
- Jako první bylo na novou architekturu adaptováno jádro Linuxu v roce 2001 (ještě před fyzickou dostupností procesorů. Jako první byl s podporou x86-64 v roce 2003 na trh uveden procesor Opteron. Původní označení x86-64 bylo společností AMD změněno na AMD64, zatímco Intel používá u svých procesorů označení EM64T a IA-32E.
- Microsoft technologii nazývá „64-bit extended systems“ nebo x64.

- Kvůli zpětné kompatibilitě je rozšíření realizováno jako další módy procesoru. K **reálnému**, **chráněnému** a **V86 módu** i386, nyní zvanými 'Legacy' (zděděné) módy, přibyly dva 'Long' (dlouhé) módy: '64bitový' a 'kompatibilní'.

- Procesor je možné provozovat buď s 32bitovým jádrem operačního systému (kterým může být i systém určený pro i386) v Legacy módech, nebo s 64bitovým jádrem v Long módech - jádro potom běží v 64bitovém módu a aplikace v 64bitovém nebo v kompatibilním.

- Většina vylepšení architektury se týká pouze 64bitového módu, menšina i kompatibilního. Legacy módy nemají žádné vylepšení (na rozdíl od i386, kde byl vylepšen i starý reálný mód).

Vylepšení architektury

- Plná podpora 64bitových celých čísel - veškeré aritmetické i logické operace se provádí v 64bitech.
- Rozšíření registrů - registry byly rozšířeny na 64bitů (stále je přístupná 32bitová, 16bitová a 8bitová část).
- Rozšíření počtu registrů - k původní sadě 8 'general-purpose' registrů přibylo dalších 8. To umožňuje držet více lokálních proměnných v registrech a tedy významně zrychluje aplikace. 16 registrů je ovšem stále málo v porovnání s RISCovými stroji. Zdvojnásoben z 8 na 16 byl i počet XMM registrů.
- Rozšíření virtuálního adresového prostoru - současné implementace AMD64 mohou adresovat 256 terabyte (2^{48}), v budoucnu bude možné to rozšířit na 16 exabyte (2^{64}). Ukazatelová aritmetika běží v 64bitech, omezení je dáno metodou překladu virtuálních adres na fyzické.

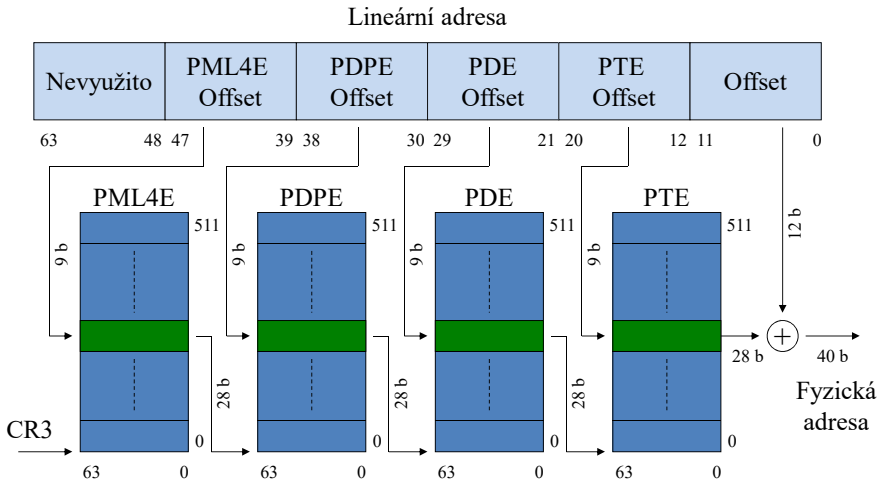
- Rozšíření fyzického adresového prostoru - současné implementace AMD64 mohou adresovat 1 terabyte (2^{40}) RAM, architektura povoluje rozšíření na 4 petabyte (2^{52}). V legacy módech je podporováno **PAE** – **Physical Address Extension** (rozšíření fyzických adres), stejně jako na moderních procesorech architektury i386, umožňující přístup k 64 gigabyte.
- Adresace relativní k ukazateli instrukce - adresace relativní k RIP zvyšuje efektivitu kódu nezávislého na pozici používaného ve sdílených knihovnách.
- SSE instrukce - součástí architektury je povinná implementace rozšíření procesorů i386 SSE a SSE2 pro výpočty v pohyblivé řádové čárce. Podpora SSE3 byla přidána dodatečně.
- Bit No-eXecute (nespustitelné) - stránku paměti je bitem NX možné označit jako obsahující pouze data a zabránit tak spuštění kódu z dané stránky. Tato vlastnost umožňuje chránit systém před většinou buffer overrun (přetečení bufferu) chyb, které se často zneužívají k útoku.
- Odstranění starších vlastností - v Long módech procesor nepodporuje některé méně používané vlastnosti i386, jako je segmentace (částečně stále fungují registry FS a GS), TSS nebo v86.

- používá tzv. **flat model**:
  - segmentace je obecně vypnuta, tzn. že bázová adresa daná registry CS, DS, ES a SS je brána jako rovna nule ⇒ lineární adresa je rovna adrese efektivní
  - výjimku tvoří bázové adresy dané registry FS a GS, jejichž hodnoty lze použít jako další báze při výpočtu lineární adresy

- stránkování je umožněno pomocí 4 tabulek:
  - PML4E – Page Map Level 4 Table Entry
  - PDPE – Page Directory Pointer Table Entry
  - PDE – Page Directory Table Entry
  - PTE – Page Table Entry

- v rámci tohoto režimu jsou podporovány dva stránkovací režimy s velikostí stránky:
  - 4 kB
  - 2 MB

Stránkovací režim se stránkou o velikosti 4 kB:



Režim procesoru		Potřebný operační systém	Nutno znovu přeložit stávající programy	Výchozí velikost adresy	Výchozí velikost operandu	Je k dispozici 64bitové rozšíření registrů	Typická šířka registru
Nové režimy	64bitový Long	64bitový OS	ano	64	32	ano	64
	Režim kompatibility		ne	32	32	ne	32
				16	16		16
Dosavadní režimy	Chráněný režim	Klasický 16 nebo 32bitový OS	ne	32	32	ne	32
				16	16		16
	Virtuální 8086 režim			16	16		16
	Reálný režim	Klasický 16bitový OS					

Physical Address Extension