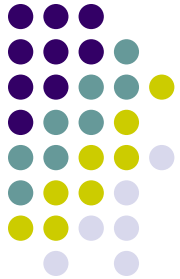
32-bitové procesory



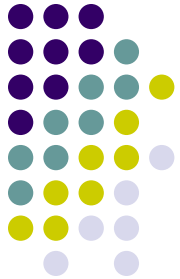
- Prvním plně 32-bitovým mikroprocesorem byl Intel iAPX432 (1981), ale ten byl brzy stažen z výroby pro svou značnou složitost

- Velmi úspěšným 32-bitovým procesorem byla **Motorola 68000**
  - Na trh uveden v roce 1980 a vyráběl se až do roku 2001
  - Obsahuje 70000 tranzistorů
  - Měl pouze 24 bitů pro adresaci (max. 16 MB)
  - Nemá střadačovou architekturu
  - 8 univerzálních datových registrů D0-D7
  - Registry jsou 32-bitové, ale s okolím komunikuje jen přes 16-bitovou sběrnici (výhodou je pak jen menší počet vývodů)
  - Adresace byla jednoduchá, bez segmentů, offsetů, selektorů...
  - Byl nasazen do počítačů Amiga a Apple
  - Mikroprocesor 68000 byl často používaný jako řídící řadič v tiskárnách
  - Nástupcem procesoru byly chipy 68010 a 68020 (v roce 1985 umožňoval i 32-bitovou adresaci paměti)
  - Mluvíme o rodině procesorů 68K

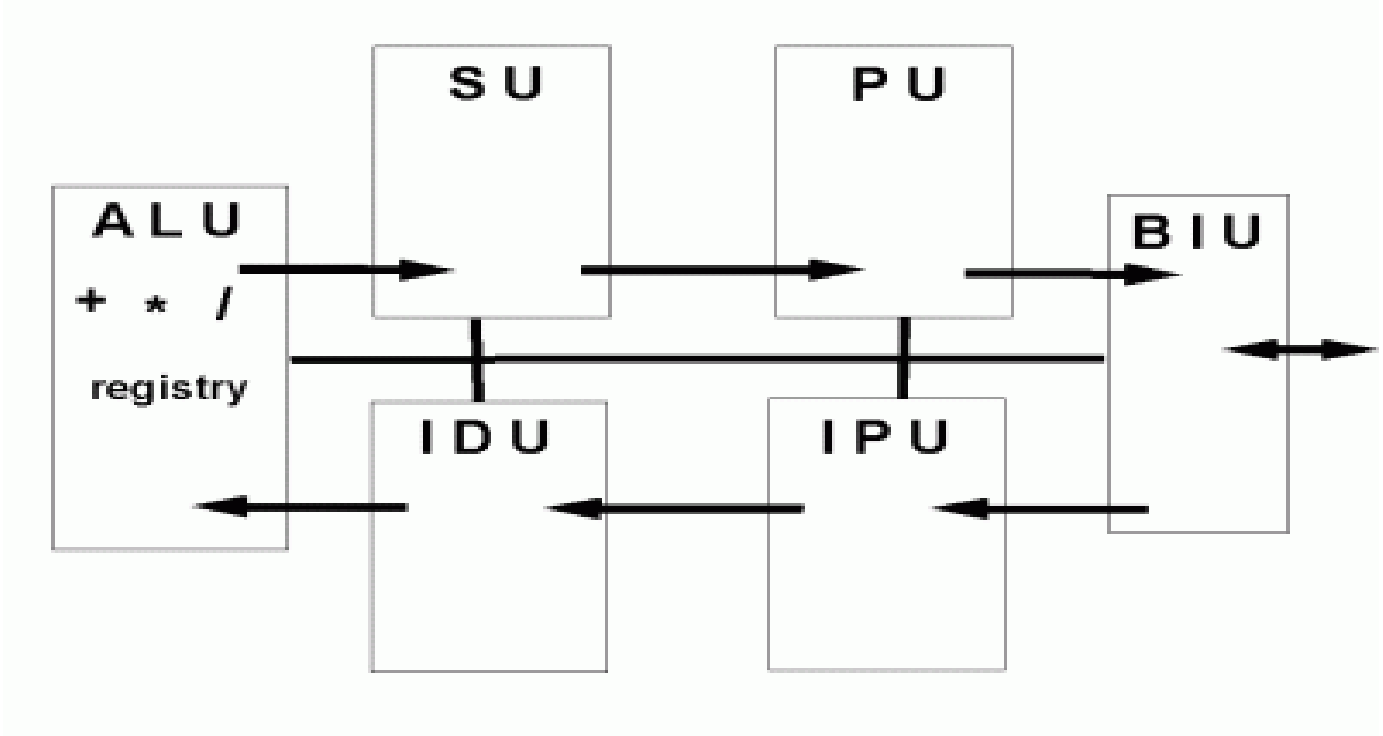
i80386

- Představen v roce 1985
- 275000 tranzistorů
- Příkon 1,8W
- Zhruba 2,5x vyšší výkon než 80286
- Patice PGA - 132 vývodů
- Zpracovává 32-bitové adresy i data
- 32-bitová adresa umožňuje adresovat až **4 GB** paměti (2^{32} B)
- Vnitřní architektura - 6 jednotek (BIU, IPU, IDU, EU, SU, PU)
- 3 režimy činnosti
  - reálný režim
  - chráněný režim (rozšířen tak, aby dovoloval jednoduché přepnutí mezi sebou a režimem reálným)
  - virtuální 86 (umožňuj spouštět programy psané pro DOS z prostředí chráněného režimu)

80386

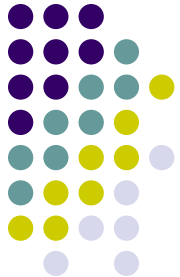


- 80386 nebyla dlouho po svém uvedení použita v žádném počítačovém systému
- **Compaq** byl první významnou firmou, která představila počítač sestavený s procesorem 80386 a tím na trhu porazila IBM
- Varianta **386SX** - ven vyvedena pouze 16-bitová datová sběrnice a 24-bitová adresní sběrnice - tedy navenek cosi jako 80286 a adresovat šlo jen 16MB paměti v chráněném režimu
- Varianta 386SX byla asi o 40% levnější než varianta DX, ale také byla podstatně méně výkonná (o polovinu nižší paměťová propustnost)
- Kopie procesoru 386 vyráběli ve velkém i další výrobci - AMD a Cyrix
- V chipsetu (chipová sada na základní desce kolem mikroprocesoru) se začíná objevovat paměť Cache (probereme později)
- 80386 je prvním mikroprocesorem, který je rychlejší než operační paměť (mikroprocesor dokáže vykonat více operací MOV, než kolik se dá skutečně provést kvůli pomalé paměti)
- K dispozici jsou numerické koprocesory 80387DX a 80387SX pro výpočty s reálnými čísly



- **BIU** - zabezpečuje veškerou komunikaci procesoru s okolím
- **IPU - Instruction Prefetch Unit** - fronta na 16 bajtů strojového kódu (průměrná délka instrukce 32b, tedy asi na 4 instrukce)
- **IDU - Instuction Decode Unit** - vyzvedne z fronty IPU instrukci, dekóduje ji a instrukci pak umístí do fronty dekódovaných instrukcí (3 instrukce)
- **EU - výkonná jednotka** - její součástí je ALU (obsahující nejen sčítačku, ale i násobičku a děličku) a soubor univerzálních registrů
- **SU - Segmentation Unit** - odpovídá svou funkcí AU u 286. Převádí virtuální adresu na lineární
- **PU - Paging Unit** - stránkovací jednotka, pracuje pouze pokud je aktivní stránkování a slouží k převodu lineární adresy na fyzickou

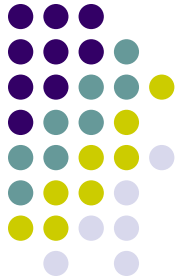
Registry



- Systém registrů je převzat z 8086
- Šířka většiny registrů se zvětšila na **32 bitů** a označují se EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP
- EAX = extended AX….
- Nadále lze používat 16-bitové registry AX, BX, CX, DX a jejich osmibitové půlky AH, AL, BH, BL, CH, CL, DH a DL
- AX, BX, CX, DX, SI, DI, BP, SP tvoří při tom vždy spodních 16 bitů registrů EAX, EBX, ECX, EDX, ESI, EDI, EBP a ESP

- Segmentové registry (selektory) CS, DS, ES a SS zůstaly **16-bitové**
- Jejich rozšíření na více bitů nemá význam – v reálném režimu lze adresovat metodou segment offset stejně jen 1 MB paměti a v chráněném režimu fungují jako selektory a v tomto případě 16 bitů stačí
- K dispozici jsou další dva nové selektory FS a GS
- Příznakový registr FLAGS se rozšířil na 32-bitový EFLAGS

Registry



31	15	7	0
EAX	AX (AH)	AX (AL)	
EBX	BX (BH)	BX (BL)	
ECX	CX (CH)	CX (CL)	
EDX	DX (DH)	DX (DL)	
ESI	SI		
EDI	DI		
EBP	BP		
ESP	SP		

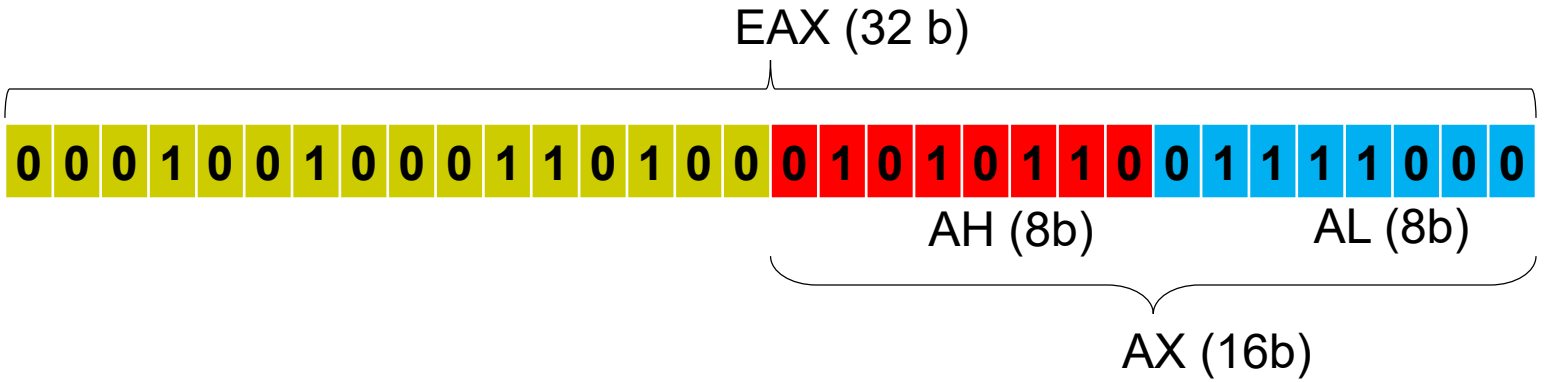
EFLAGS	FLAGS

EIP	IP

15	0
CS	
DS	
SS	
ES	
FS	
GS	

32-bitové registry

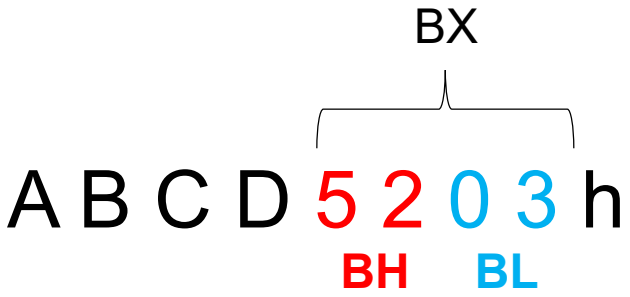
- Příklad
- EAX=12345678h
- Určete stav AX, AH, AL



AX = 0101011001111000 = **5678h**
AH = 01010110 = **56h**
AL = 01111000 = **78h**

32-bitové registry

- Příklad
- EBX=ABCD5203h
- Určete stav BX, BH, BL

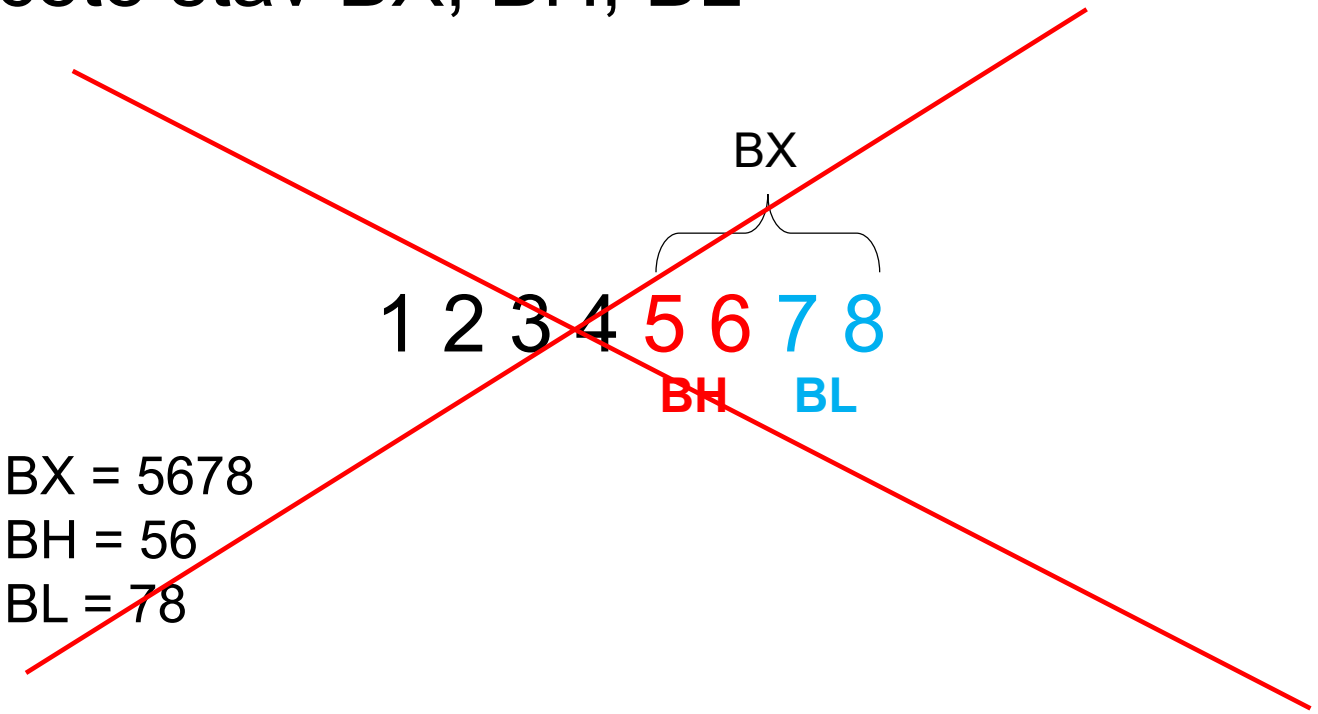


BX = 5203 h
BH = 52 h
BL = 03 h

32-bitové registry

- Příklad
- EBX=12345678
- Určete stav BX, BH, BL

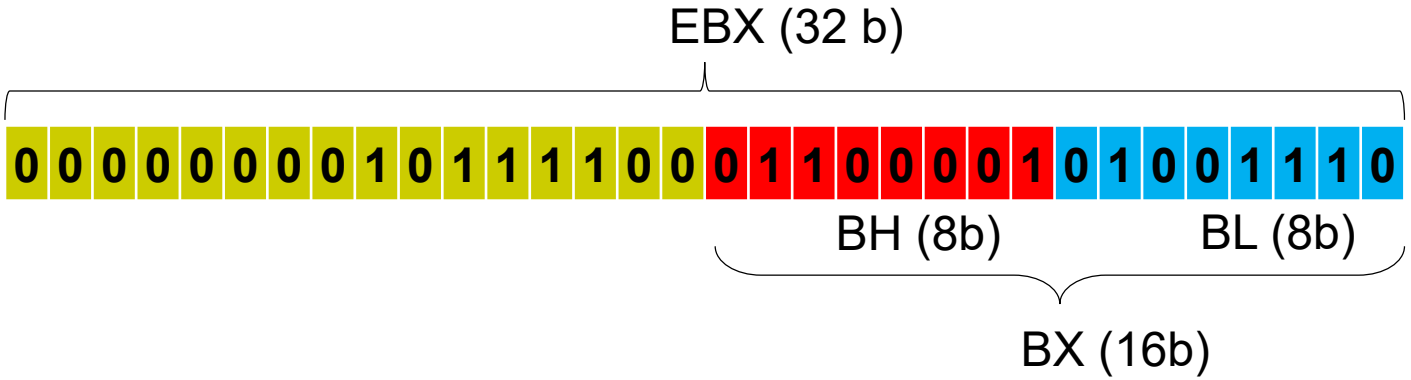
Pozor !!!
Pokud je hodnota registru zapsána v desítkové soustavě, neplatí, že spodní 4 cifry odpovídají spodním 16-bitům (to funguje pouze v šestnáctkové soustavě)



BX = 5678
BH = 56
BL = 78

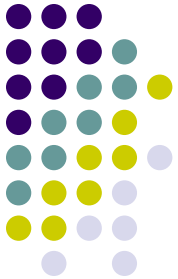
32-bitové registry

- Příklad
- EBX=12345678 (desítková soustava !)
- Určete stav BX, BH, BL



BX = 0110000101001110 = **614Eh = 29490**
BH = 01100001 = **61h = 97**
BL = 01001110 = **4Eh = 78**

Chráněný režim

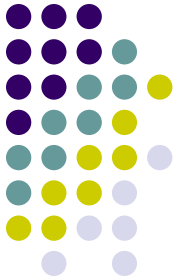


- Adresovat lze **4 GB** fyzické paměti (**32-bitová fyzická adresa**)

- **Selektor** je stejný jako v 80286 (16-bitový registr CS, DS, ES, SS, FS nebo GS)
- Horních 13 bitů selektoru (**index**) vybírá jednu z položek tabulky deskriptorů
- Tabulka deskriptorů může obsahovat popis až 8192 segmentů
- Segmenty mohou mít velikost **až 4 GB**
- **Offset** má šířku **32 bitů**
- **Offset** vybírá pozici uvnitř segmentu a může nabývat hodnot <0; FFFFFFFFh>, to je <0; 4 294 967 295>

- **Logická adresa** je složena z 16-bitového selektoru a 32-bitového offsetu
- Segmentační jednotka má za úkol ze selektoru a offsetu vygenerovat tzv. **lineární adresu**
- Další postup závisí na tom, jestli je nebo není zapnuto stránkování
- Je-li stránkování vypnuto, pak lze lineární adresu považovat za fyzickou
- Je-li stránkování zapnuto, stránkovací jednotka dále transformuje lineární adresu pomocí tabulky stránek na fyzickou adresu

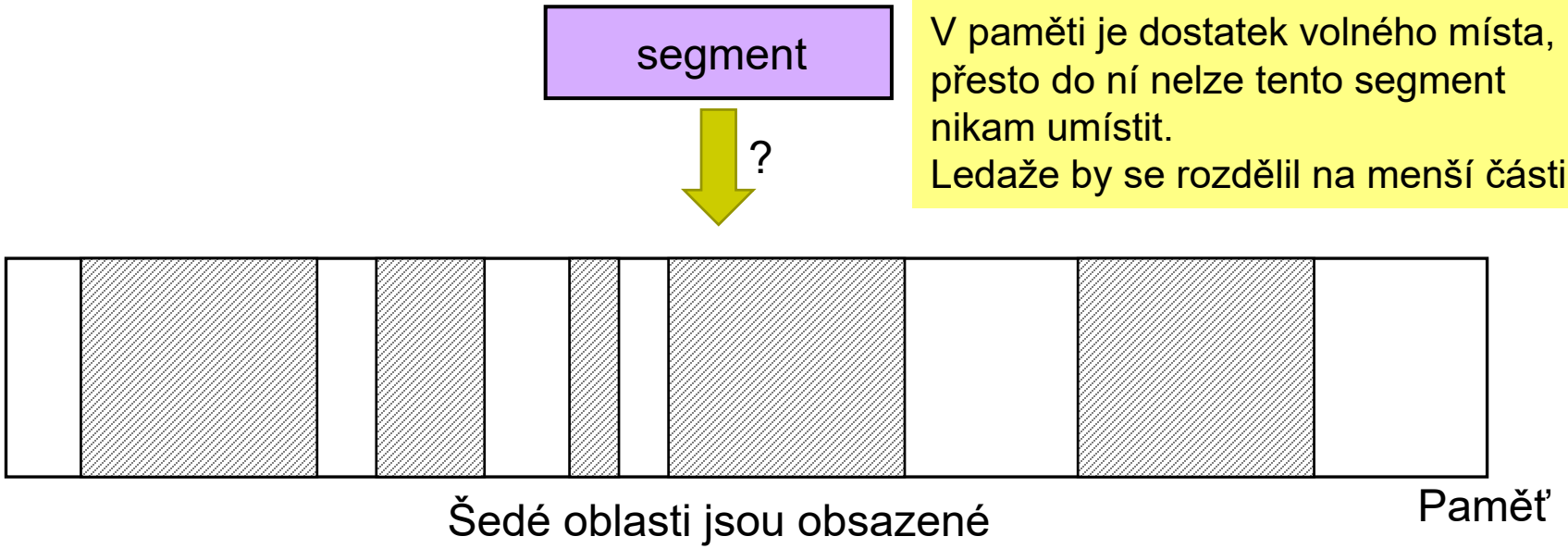
Stránkování



- Segmenty mohou mít nyní ohromnou velikost až **4 GB**
- To by přineslo velký problém s virtuální pamětí v okamžiku, kdy je třeba načíst segment z disku nebo odložit na disk
- Segmenty jsou příliš velké a docházelo by k výměně zbytečně velkých bloků dat mezi pamětí a diskem
- Při přístupu do segmentu, který není fyzicky uložen v paměti, obvykle není potřeba celý segment, ale jen jeho část
- Mikroprocesor 80286 fungoval tak, že byl z disku do paměti přemísťován celý segment
- Neustálé **přemísťování velkých segmentů** by ale fungování virtuální paměti v praxi velmi zkomplikovalo (zejména časově)
- Díky zavedení stránkování nemusí být celý segment buď odložený na disku nebo umístěný v paměti
- Segment se rozdělí na **4 kB stránky**
- Některé stránky (části segmentu) mohou být odloženy na disku, některé stránky jsou uloženy v paměti
- Segment je tedy rozdělen na více částí (stránek)
- V případě potřeby práce s daty, která jsou odložena na disku, se načítá z disku do paměti pouze malá **4 kB stránka** a nemusí se načítat **celý obrovský segment**

Stránkování

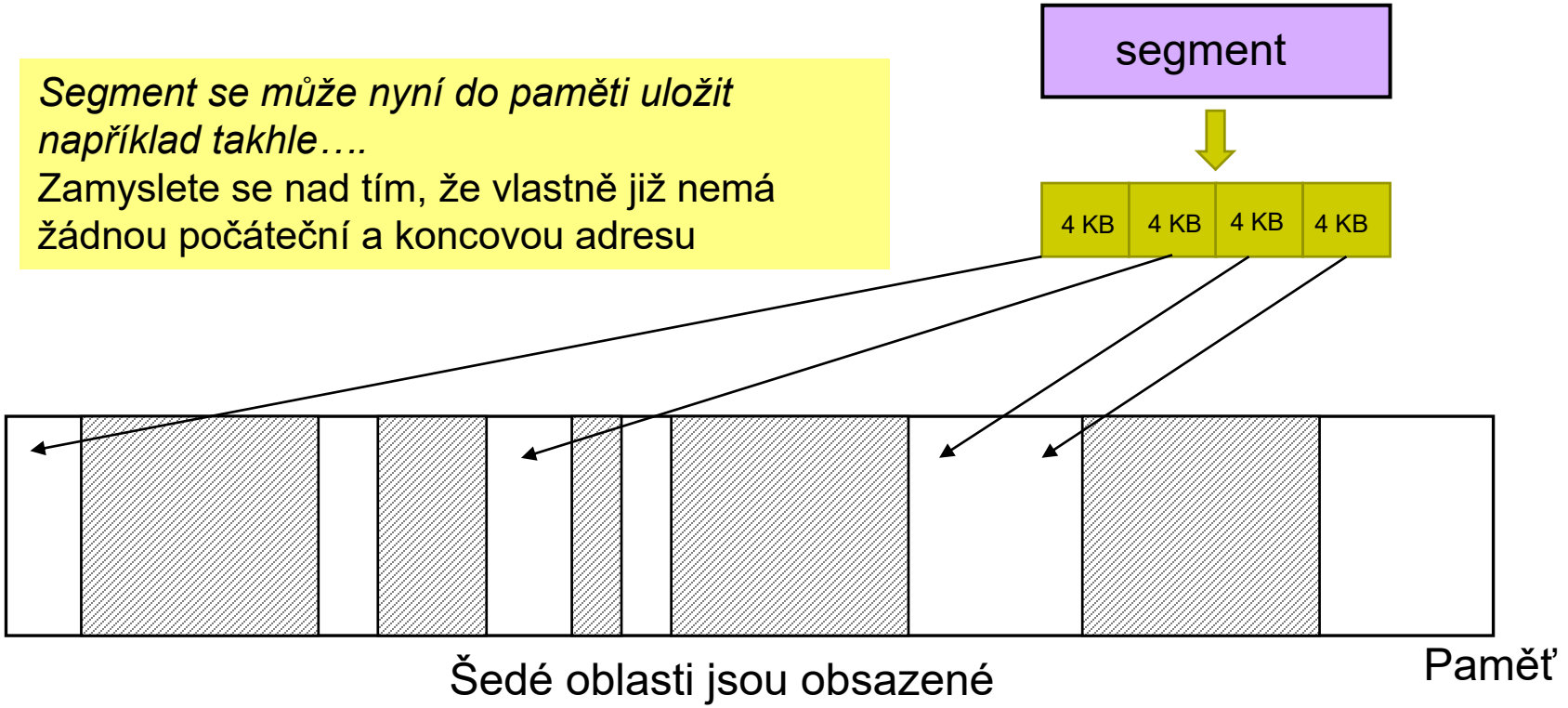
- Kvůli velkým segmentům (až 4 GB), které by nešly rozdělit na menší části, by nastával také problém s **fragmentací** paměti
- Po určité době běhu počítače by v paměti mohl být velký počet volných míst ale příliš malé jednotlivé délky, než aby se tam dal fyzicky uložit nějaký ze segmentů
- Mohla by tedy snadno nastat situace, že je poměrně hodně volné paměti, ale neexistuje **dostatečně velký použitelný souvislý kus**
- Pro **velký segment** by se obtížně hledalo volné **souvislé** místo v paměti



Stránkování

- Díky zavedení stránkování **nemusí** být segment uložen v paměti **souvisle**
- Stránky (jednotlivé části segmentu) mohou být v paměti rozmístěny libovolně
- Některé části segmentu mohou tedy zůstat odloženy na disku a ty části segmentu, které jsou uloženy v paměti, mohou ležet na přeskáčku na mnoha různých spolu nesousedících místech v paměti

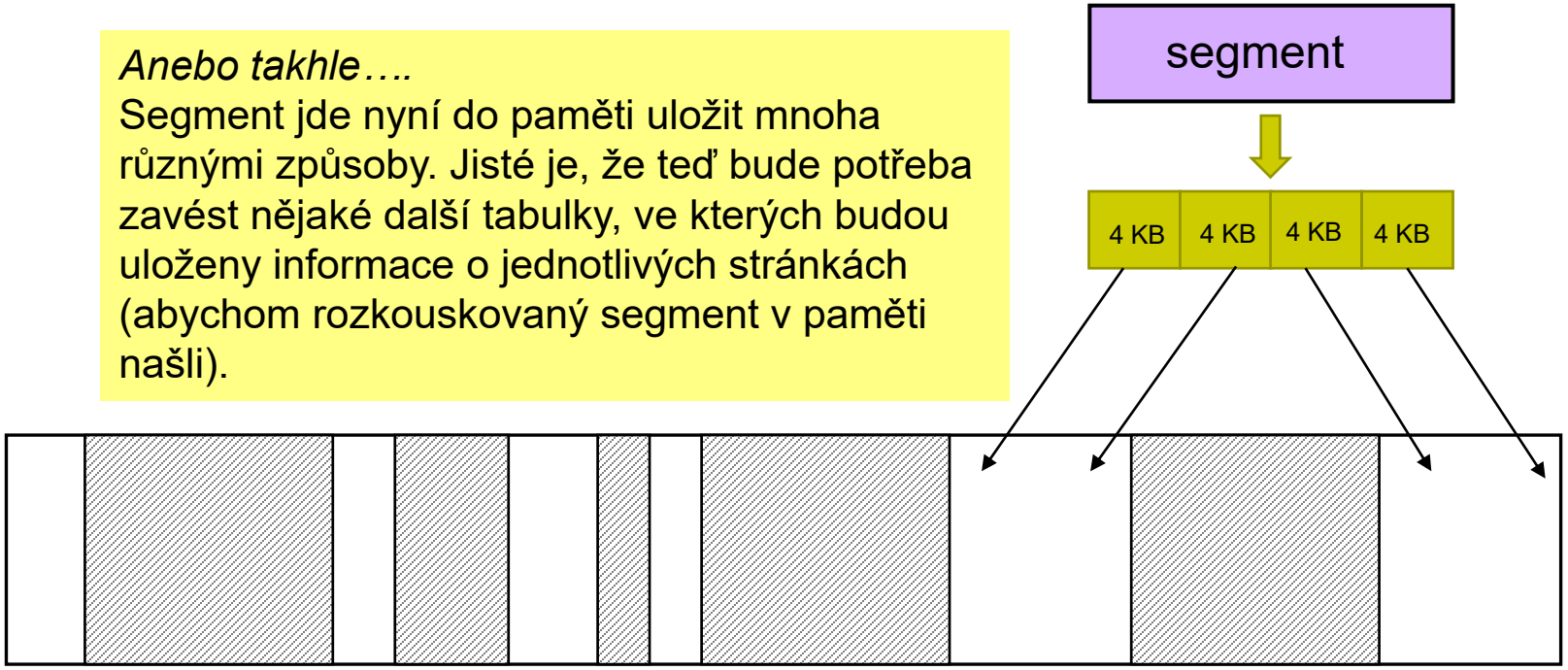
Segment se může nyní do paměti uložit například takhle….
Zamyslete se nad tím, že vlastně již nemá žádnou počáteční a koncovou adresu



Stránkování

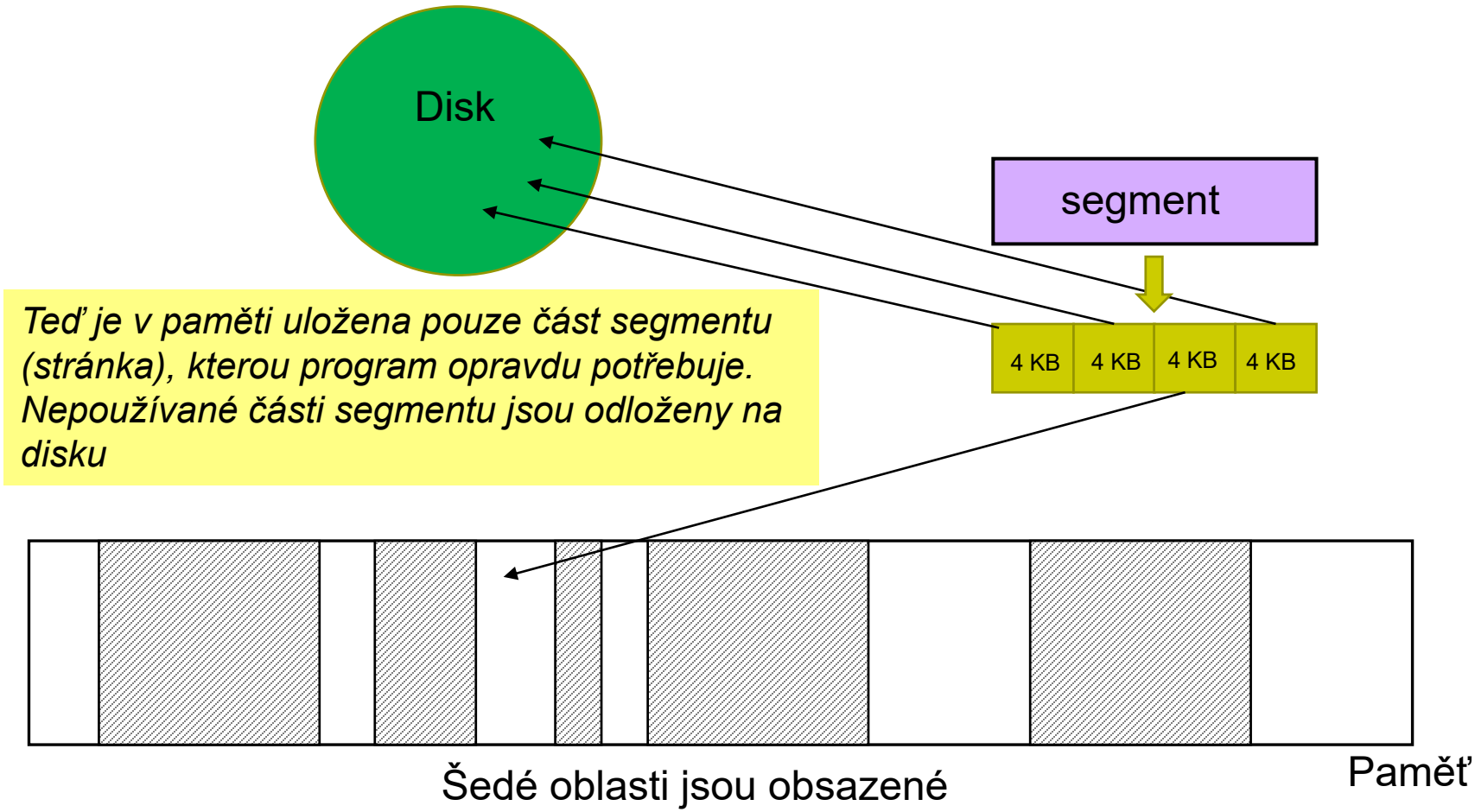
- Díky zavedení stránkování **nemusí** být segment uložen v paměti **souvisle**
- Stránky (jednotlivé části segmentu) mohou být v paměti rozmístěny libovolně
- Některé části segmentu mohou tedy zůstat odloženy na disku a ty části segmentu, které jsou uloženy v paměti, mohou ležet na přeskáčku na mnoha různých spolu nesousedících místech v paměti

Anebo takhle….
Segment jde nyní do paměti uložit mnoha různými způsoby. Jisté je, že teď bude potřeba zavést nějaké další tabulky, ve kterých budou uloženy informace o jednotlivých stránkách (abychom rozkouskovaný segment v paměti našli).

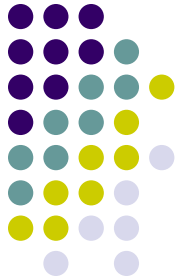


Stránkování

- Některé části segmentu mohou zůstat odloženy na disku

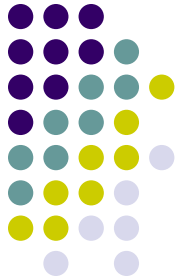


Stránkování



- Problémy způsobené velkými segmenty jsou řešitelné rozdělením segmentů a paměti na menší stejně velké úseky – **stránkování**
- Segmenty jsou tedy dál rozděleny na **stránky**, se kterými lze lépe manipulovat
- Základní úsek paměti – **stránka** – je velká **4 kB** (všechny stránky jsou **stejně velké**)
- Stránky tvořící jeden segment nemusí být v paměti uloženy za sebou (mohou být rozházené po paměti)
- Segment nemusí být fyzicky celý přítomen v paměti – nepoužívané stránky mohou být odloženy na disku
- Segment již nemá žádnou **počáteční adresu** – jeho části jsou různé rozházené po paměti a některé stránky v paměti nejsou vůbec, takže nelze říct, kde začíná nebo končí

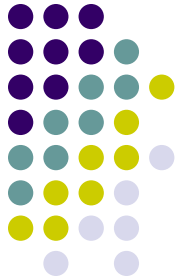
Příklad – 1. situace
vypnuté stránkování



- Třída 3.D (30 žáků) jde do kina
- V kině je ještě 50 volných míst, která jsou náhodně rozmístěna
- 30 volných míst vedle sebe tu není
- Třída musí sedět souvisle, všichni žáci pěkně jeden vedle druhého
- Přestože je v sále dost volných míst, třídu nelze nikam umísit, není tu pro ní dostatečně dlouhý úsek volných sedadel vedle sebe

- Program používá 30 MB velký datový segment
- V paměti je 50 MB volného místa
- Není tu ale souvislý úsek 30 MB volné paměti
- Segment musí být umístěn do paměti vcelku a souvisle
- Přestože je v paměti dostatek volného místa, segment nelze nikam umístit. Není tu pro něj dostatečně dlouhý volný úsek

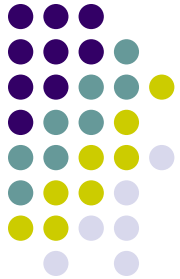
Příklad – 2. situace
vypnuté stránkování



- Třída 3.D (30 žáků) jde do kina
- V kině je ještě 50 volných míst
- Naštěstí lze nalézt i 30 volných míst vedle sebe
- Třída musí sedět souvisle, všichni žáci pěkně jeden vedle druhého
- Žáci si sednou vedle sebe podle abecedy na po sobě jdoucí sedadla
- Třída má počáteční a koncové místo
- Známe-li sedadlo, kde sedí první žák, lze snadno určit číslo sedadla každého žáka ve třídě (přičtením jeho pořadového čísla v třídní knize k počátečnímu číslu sedadla)

- Program používá 30 MB velký datový segment
- V paměti je 50 MB volného místa
- Lze nalézt i souvislý úsek 30 MB volné paměti
- Segment musí být umístěn do paměti vcelku a souvisle
- Segment má počáteční a koncovou adresu
- Poloha každého bajtu v segmentu je jednoznačně dána jeho offsetem
- Přičtením offsetu k počáteční adrese segmentu lze vypočítat fyzickou adresu bajtu v paměti

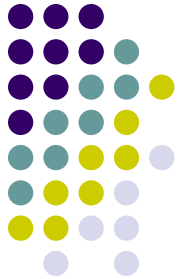
Příklad – 3. situace zapnuté stránkování



- Třída 3.D (30 žáků) jde do kina
- V kině je ještě 50 volných míst, která jsou náhodně rozmístěna
- 30 volných míst vedle sebe tu není, ale to nevadí
- Třída nemusí sedět souvisle, jednotliví žáci si sednou na různá volná místa po celém sále
- Nelze přesně říct, kde má 3.D počátek a kde konec – žáci sedí všude možně
- Někteří žáci zůstali doma a v kině nejsou
- Je potřeba vytvořit tabulku (zasedací pořádek), abychom nalezli v sále jednotlivé žáky

- Program používá 30 MB velký datový segment
- V paměti je 50 MB volného místa
- Není tu souvislý úsek 30 MB volné paměti, ale to nevadí
- Segment nemusí být umístěn do paměti vcelku a souvisle
- Segment se rozdělí na 4 KB stránky
- Jednotlivé části segmentu (stránky) se umístí na různá volná místa v paměti
- Segment je rozházený po celé paměti
- Nelze tedy určit počáteční nebo koncovou adresu segmentu
- Některé stránky jsou odložené na disku a nebyly načteny do paměti
- Je třeba vytvořit stránkovací tabulky, abychom jednotlivé části segmentu v paměti nalezli

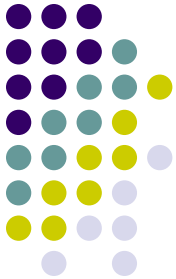
Stránkování paměti



- **Paměť** se rozdělí na 4 kB „okénka“ – **stránkové rámce**
- Stránkový rámec může být buď volný nebo obsazený nějakou 4 kB velkou stránkou
- Velikost segmentů může být 4 kB, 8 kB, 12 kB, 16 kB… atd.
- Segment teď už ale nemůže mít velikost např. 10 kB

- V paměti není buď žádné volné místo, nebo je zde 4 kB volného místa (jeden volný stránkový rámec), 8 kB volného místa (2 volné stránkové rámce), 12 kB volného místa atd……
- Nemůže teď nastat situace, že situace, že by bylo v paměti 10 kB nebo 1 KB volného místa

Stránkování



Paměť– až 4 GB

Stránka může být uložena v některém z 2^{20} stránkových rámců v paměti nebo je odložena na disku

Segment – 24 kB

4 kB

4 kB

4 kB

4 kB

4 kB

4 kB

4 kB

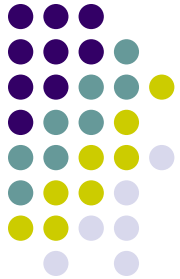
Stránkové rámce 4 kB „okénka“, na které se rozdělí celá paměť

4 GB RAM lze rozdělit na 2^{20} stránkových rámců

DISK

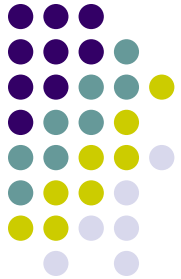
Bude třeba vytvořit tabulku s informacemi o tom, kde jsou jednotlivé stránky umístěné, abychom jednotlivé části segmentu nalezli v paměti (takový jejich zasedací pořádek)

Stránkování



- Stránkování funguje pouze v chráněném režimu
- Každý segment má svůj deskriptor
- V deskriptoru segmentu je uložena jeho „počáteční adresa“
- Nejde však o skutečnou počáteční adresu
- Segment vlastně nyní žádnou počáteční ani koncovou adresu nemá – je v paměti uložen nesouvisle na několika různých místech, takže nelze říct, kde začíná nebo končí
- K čemu je pak tedy počáteční adresa segmentu?
- Počáteční adresa segmentu pouze odkazuje na správné místo do tabulky stránek tak, aby zde byly nalezeny informace o jednotlivých stránkách daného segmentu
- V paměti je uložena **tabulka stránek** s informacemi o jednotlivých stránkách
- K počáteční adrese segmentu se přičte offset, čímž vznikne **lineární adresa**
- 32-bitová lineární adresa je rozdělena na **20 bitový index** do tabulky stránek a **12 bitový offset**
- Stránkový offset ukazuje na konkrétní pozici v 4kB velké stránce (2^{12}=4096)

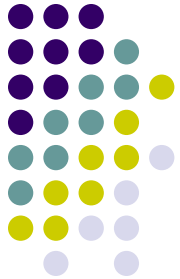
Příklad – 4. situace
zapnuté stránkování



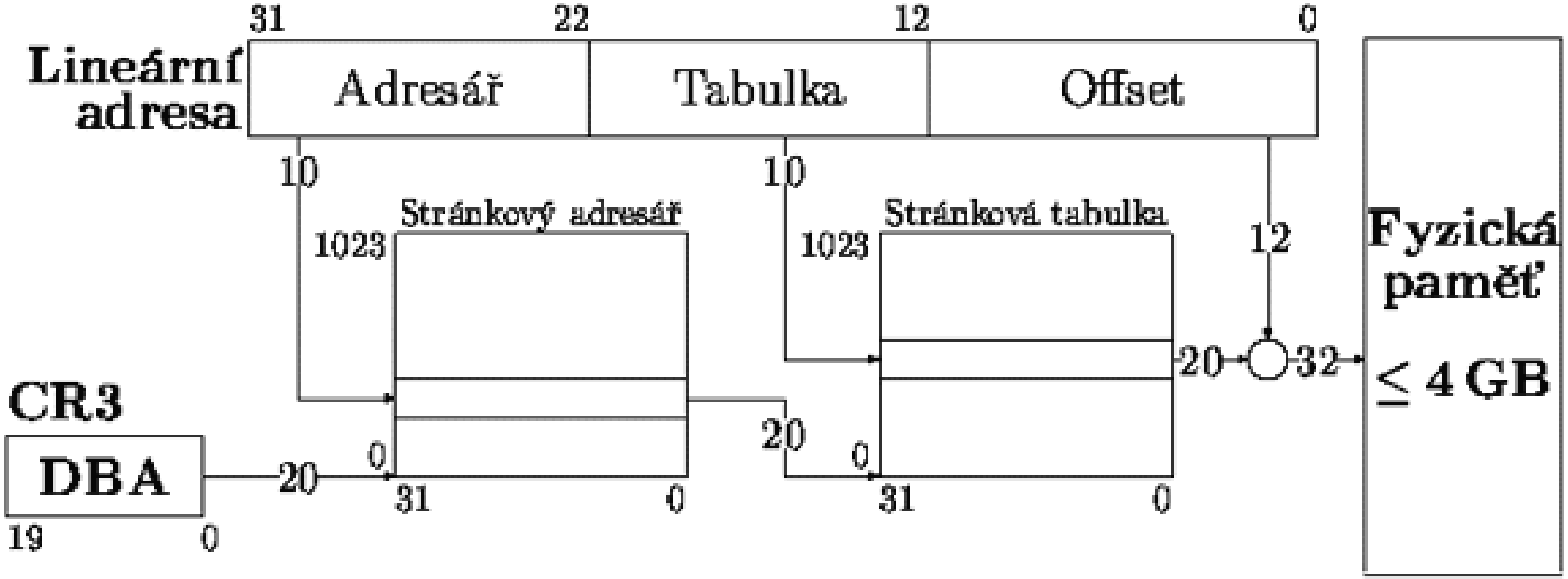
- Třída 3.D (30 žáků), 3.E (28 žáků), 3.F (27 žáků), 3.G (29 žáků) jdou do kina
- V kině je ještě dost volných míst, která jsou náhodně rozmístěna
- Maximálně je tu 5 volných míst vedle sebe tu není, ale to nevadí
- Třídy nemusí sedět souvisle, jednotliví žáci si sednou na různá volná místa po celém sále
- Nelze přesně říct, kde má každá třída počátek a kde konec – žáci sedí všude možně
- Je potřeba vytvořit tabulku (zasedací pořádek), abychom nalezli v sále jednotlivé žáky
- V tabulce budou informace o všech třídách a jejich žácích
- O každé třídě máme zapsanou informaci, kde v tabulce začínají informace o jejich žácích
- Nemáme tedy uloženou počáteční pozici třídy v kině, ale počáteční pozici informací o třídě v „zasedací tabulce“

- Běžící programy používají mnoho různě velkých segmentů
- V paměti je dost různých úseků volného místa, které jsou náhodně rozmístěny
- Segmenty nemusí být umístěny do paměti vcelku a souvisle
- Segmenty se rozdělí na 4 KB stránky
- Jednotlivé části segmentů (stránky) se umístí na různá volná místa v paměti
- Segmenty je rozházený po celé paměti
- Nelze tedy určit počáteční nebo koncovou adresu každého segmentu
- Je třeba vytvořit stránkovací tabulku, abychom jednotlivé části segmentu v paměti nalezli
- Každý segment má svůj deskriptor, ve kterém je uložena jeho „počáteční adresa“
- Nejde ale o jeho počáteční adresu v paměti (takový pojem teď ani nedává smysl)
- Jedná se odkaz do jiné tabulky – kde ve stránkovací tabulce začínají informace o jednotlivých stránkách tohoto segmentu

Stránkování



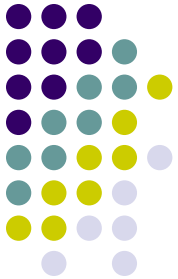
- Horních 20 bitů lineární adresy je použito jako ukazatel do tabulky stránek
- Tabulka stránek může obsahovat informace o **2^{20}** stránkách (1 mega stránek)
- Položky v tabulce stránek mají rozsah **32 bitů** (informace o každé stránce)
- Kompletní tabulka stránek by byla ohromně velká $4 \times 2^{20} = 4$ MB a takový rozsah paměti není pro účel uložení tabulky stránek za normálních okolností možné obsadit
- Proto bylo zvoleno **dvouúrovňové** schéma organizace tabulky stránek
- Dvouúrovňové stránkování znamená, že vlastně samotná tabulka stránek bude stránkována
- První v hierarchii tabulek je **adresář stránek** - obsahuje informace o tabulkách stránek
- Položka v adresáři stránek odkazuje na **tabulku stránek** (druhá úroveň) a v ní už je konečně 32-bitová informace o stránce
- **Stránkovací tabulka** = Tabulka stránek, je velká **4 kB** a obsahuje informace o **1024 stránkách**
- Každý proces má vlastní stránkový adresář odkazující na jeho tabulky stránek
- Tabulek stránek může být až **1024** – podle potřeby
- Pokud proces používá například 5400 stránek, bude pro něj potřeba vytvořit 5 tabulek stránek



Adresář stránek obsahuje 20-bitový odkaz na tabulku stránek

Tabulka stránek obsahuje horních dvacet bitů z 32 počáteční adresy stránky. (spodních dvanáct bitů je vlastně offsetem do stránky)

Stránkování



Příklad

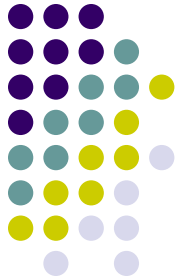
- 24 kB segment je rozdělen na 6 stránek
- Informace o těchto stránkách leží na 15.-20. řádku v padesáté stránkovací tabulce
- Jak bude vypadat „počáteční adresa“ segmentu uložená v jeho deskriptoru?

- Nejvyšších 10 bitů lineární adresy odkazuje do **stránkového adresáře**, ve kterém je potřeba najít informaci o *padesáté* stránkovací tabulce
- Dalších 10 bitů musí do této **stránkovací tabulky** odkazovat na *15. řádek*
- „Počáteční adresa segmentu“ bude tedy nastavena takto:

0C80F000h

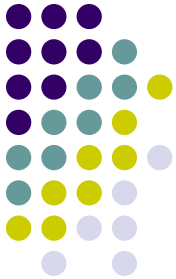
0000110010 0000001111 000000000000 - 32-bitová adresa

50 15



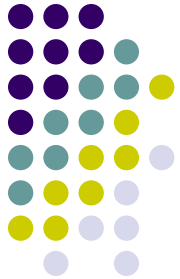
- Pokud chce program pracovat se stránkou, která není uložena v paměti, nastane **přerušení** (výpadek stránky – interrupt 15), které obslouží operační systém tak, že načte požadovanou stránku z disku
- Za odkládání stránek na disk a jejich načítání v případě potřeby je zodpovědný operační systém
- Každý operační systém si organizuje odkládání stránek na disk sám dle vlastní potřeby
- Některé OS k tomu například používají na disku samostatný oddíl (swap)
- OS Windows odkládá stránky do souboru **pagefile.sys**

32-bitový chráněný režim



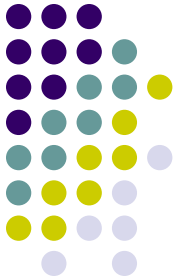
- Na mikroprocesoru 80386 se objevila nová, upravená 32-bitová verze chráněného režimu
- Tento 32-bitový chráněný režim s možností 4 kB stránkování se používá dodnes
- V tomto režimu dnes běží tzv. **32-bitové aplikace**
- 32-bitová aplikace tedy používá stejný způsob adresace paměti, jaký byl zaveden na mikroprocesoru 80386 a pracuje se 32-bitovými registry EAX, EBX, ECX….
- Mikroprocesory, které umí pracovat v tomto 32-bitovém chráněném režimu jsou všechny navzájem kompatibilní a označujeme je jako **IA-32 procesory**
- **IA-32** = Intel architecture 32 bit
- Předchůdcem architektury IA-32 byla 16-bitová architektura (Intel 8086 a Intel 80286) a nástupcem bude 64-bitová architektura x86-64 (AMD64)
- Současné moderní procesory umí pracovat jak v režimu IA-32, tak v novém režimu x86-64 a lze na nich spouštět 32-bitové i 64-bitové aplikace

Stránkování paměti



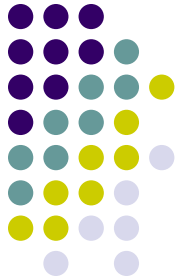
- Stránkování přináší zpomalení a další komplikaci při adresaci – proces výpočtu fyzické adresy se rozšířil o další fázi (nestačí přičíst offset k počáteční adrese segmentu, ale pracuje se dále se stránkovým adresářem a stránkovací tabulkou a dalším offsetem)
- Stránkování však vede k zjednodušení práce s velkými segmenty
- Stránkování se ukázalo být velmi praktické a se používá se drobnými změnami dodnes
- Stránkování paměti dnes používají i prakticky všechny ostatní procesorové platformy, nejen procesory řady x86
- Stránkování používají například i procesory v mobilních telefonech a tabletech

TLB



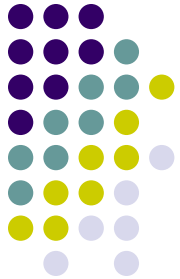
- **T**ranslation **L**ookaside **B**uffer
- Stránkování přineslo problém dvou přístupů do tabulek, které jsou uloženy v operační paměti a přístup k nim může procesor zdržovat
- TLB je **malá paměť** přímo na čipu mikroprocesoru pro zrychlení opakovaného přístupu k nedávno použitým stránkám
- v TLB jsou uchovány informace o naposledy provedených převodech lineárních adres na fyzické – tedy vlastně informace o několika naposledy používaných stránkách
- Běžící program obvykle přistupuje do stejné stránky vícekrát – má tendenci pracovat opakovaně se stejnou adresou nebo adresami v jejím okolí, které jsou součástí stejné stránky
- TLB funguje jako „asociativní paměť“
- Hledání v TLB proběhne okamžitě – asociativní paměť obsahuje komparační obvody a všechny záznamy v ní jsou naráz otestovány, zda se neshodují s překládanou lineární adresou (testuje se horních dvacet bitů)
- Lineární adresa, která se má přeložit na fyzickou, je nejprve hledána v TLB – Nepracovali jsme s touto stránkou v nedávné minulosti?
- Dojde-li ke shodě s nějakou s položek v TLB, informace o stránce jsou ihned k dispozici, jsou zapamatované „z minula“, čímž se překlad adresy zrychlil, protože se nemuselo přistupovat do adresáře a tabulky stránek

V86 režim



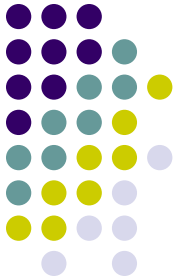
- Režim **virtuální 8086**
- V operační paměti je vytvořen prostor **1 MiB**, ve kterém je adresováno pomocí segmentu a ofsetu stejně, jako je tomu v **reálném režimu**
- V této oblasti paměti pak mohou být spuštěny programy, které byly napsány pro reálný režim (tedy bez selektorů, deskriptorů atd.)
- Původně se předpokládalo, že se zavedením „dokonalého a skvělého“ chráněného režimu se zcela přestane používat neperspektivní režim reálný
- Opak byl ale pravdou
- Reálný režim nezanikl, vývoj SW zdaleka nestíhal rychlost vývoje HW (což vlastně platí dodnes) a bylo třeba i na moderním procesoru umožnit efektivní běh starých programů psaných pro reálný režim
- V režimu V86 je využito všech výhod víceúlohového prostředí, správy paměti a ochran v chráněném režimu
- Úlohy v tomto režimu jsou vždy prováděny s nejnižší úrovní oprávnění
- Z hlediska úlohy běžící v tomto režimu je adresa generována stejně jako v režimu reálném a lze využívat 1 MB paměti
- Na pozadí ve skutečnosti probíhá složitý správa paměti

Další doporučené odkazy



- http://www.pc-teritory.wz.cz/teorie/ep4/HW_PC/Procesory/Spr_pam.HTM
- http://cs.wikipedia.org/wiki/Str%C3%A1nkov%C3%A1n%C3%AD
- http://www.root.cz/clanky/adresovani-procesoru-intel-x86/

Kontrolní otázky



- Jaká je šířka datových registrů mikroprocesoru 80386 ?
- Jaká je šířka registrů CS, DS, ES a SS mikroprocesoru 80386 ?
- Jaká je šířka fyzické adresy v chráněném režimu na mikroprocesoru 80386 ?
- Jaká je maximální možná velikost segmentu na mikroprocesorech 8086, 80286 a 80386 ?
- Porovnejte počet tranzistorů v mikroprocesorech 8086, 80286 a 80386
- V čem se liší varianta 80386SX a 80386DX ?
- Popište roli jednotek SU a PU
- Je-li EAX=56789ABh, určete hodnotu AX, AH, AL
- Porovnejte, jak velkou paměť lze adresovat mikroprocesory 8086, 80286 a 80386
- Kolikabitový index, selektor, deskriptor a offset používá mikroprocesor 80386 v chráněném režimu ?
- Vysvětlete pojmy a vzájemné vztahy: logická adresa, lineární adresa, fyzická adresa
- Vysvětlete důvod zavedení stránkování z hlediska virtualizace paměti.
- Vysvětlete problém fragmentace paměti
- Vysvětlete vzájemný vztah mezi adresářem stránek, tabulkou stránek a offsetem
- Proč se nepoužívá jednoúrovňová tabulka stránek ?
- Přístup do paměti při zapnutém stránkování se zrychlí nebo zpomalí ? Zdůvodněte svou odpověď
- K čemu slouží TLB ?
- Při kterých přístupech do paměti TLB zrychlí nalezení polohy požadované stránky a kdy nikoliv
- Proč vznikl režim V86 ?
- Kde je uložen stav rozpracovaného procesu ?
- Kde je uložena bázová adresa lokální tabulky deskriptorů ?
- Jaké nejvyšší číslo lze uložit do registru EBX ? (hexadecimálně)
- Kolikabitový je offset ukazující pozici v stránce a kde je umístěn, kolikabitový je offset ukazující pozici v segmentu
- Co je to IA-32?