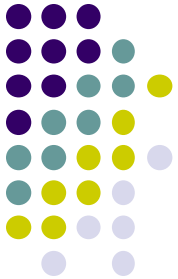
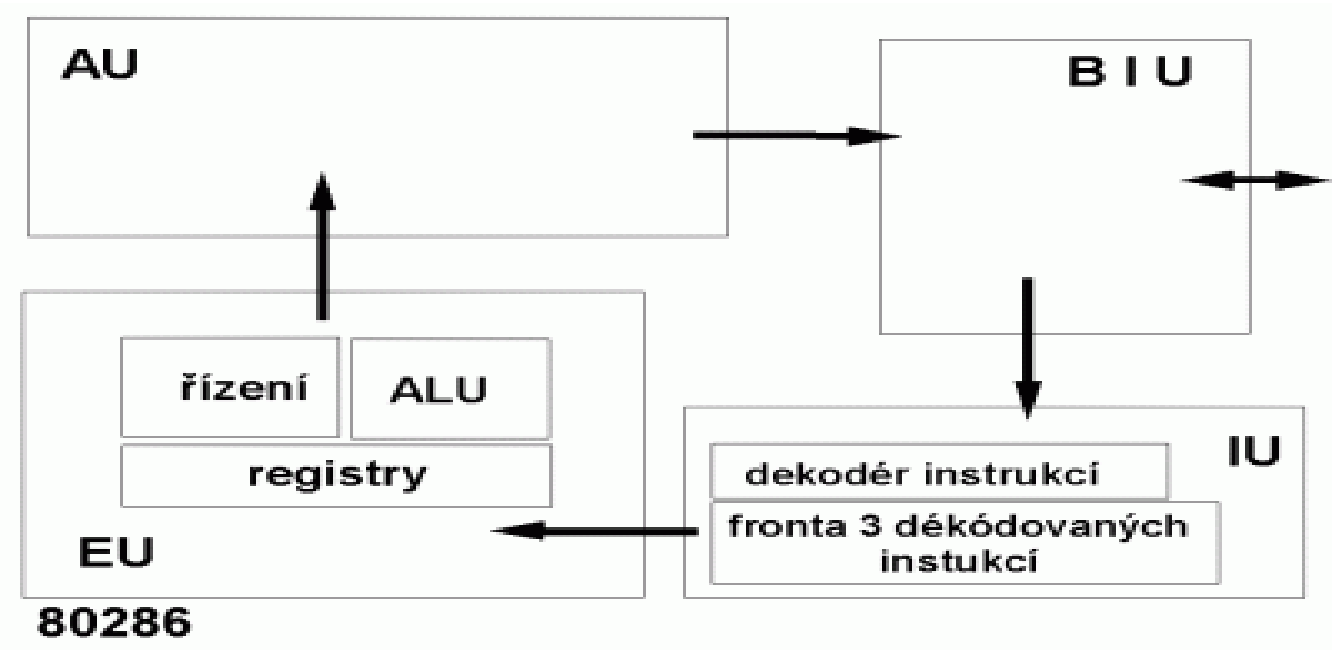
80286 - základní vlastnosti

- Uveden na trh v roce 1982
- 16-bitový procesor
- 130 000 tranzistorů
- Frekvence 6-12 MHz
- Je asi 2,5x výkonnější než 8086
- Vnitřní struktura obsahuje 4 jednotky
  - BU (Bus Unit)
  - AU (Adress Unit)
  - IU (Instruction Unit)
  - EU (Execution Unit)
- Podporuje **multitasking**
- Byly jím vybavovány počítače řady IBM PC/AT
- Vyráběl se v pouzdře PGA a později PLCC - 68 vývodů a QPF
- Zásadní novinka - možnost činnosti ve dvou různých režimech
  - **reálný režim**
  - **chráněný režim** (k čemu je to dobré si podrobně povíme později)

80286



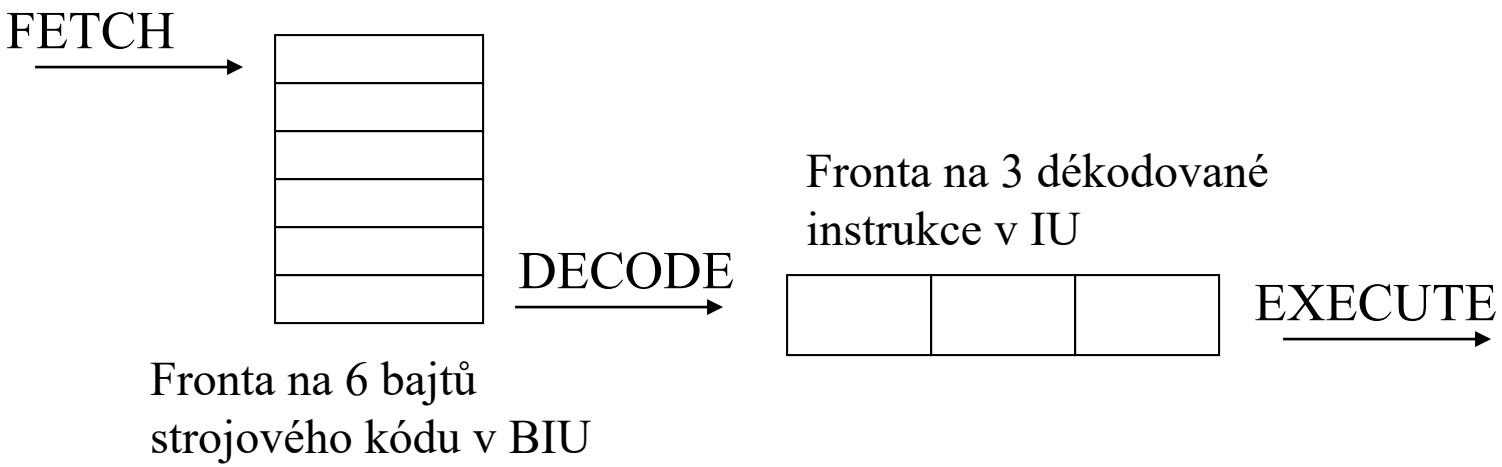
- 8086 uměla generovat **20-bitovou adresu** a tak bylo možné adresovat **1MB** paměti
- 80286 umí generovat **24 bitů adresy** a adresní prostor se tak zvětšuje na **16MB** (2^{24}=16 Mega)
- V reálném režimu lze adresovat stále pouze 1MB a plný 16MB velký adresní prostor lze využít jen ve chráněném režimu
- V chráněném režimu ovšem nelze spouštět programy pro režim reálný. Původní programy psané pro 8086 na 80286 bez problémů běží, ale pouze v reálném režimu (takže s omezením 1MB paměti)
- IBM PC-AT se na trhu objevilo až v roce 1985, tři roky po uvedení procesoru 80286
- Software silně zaostával za hardwarem a až v roce 1988 se objevili první programy, které dokázaly využít chráněný režim
- Intel sám nestíhal tento procesor vyrábět a tak zadal jeho výrobu i dalším výrobcům - IBM, AMD, Harris, Siemens a taktovací frekvence se zvedla na 16, 20 a až 25 MHz
- Procesor se používal téměř 10 let



- **BU - Bus Unit** - Stará se o komunikaci s okolím. Zásobuje mikroprocesor strojovým kódem, který se načítá v předstihu do 6 B dlouhé fronty (stejně, jako u 8086)
- **IU - Instruction Unit** - Dekóduje instrukce připravené ve frontě BU a ukládá 3 dekódované instrukce do fronty pro EU. Díky dekódování ví mikroprocesor předem, jaké instrukce na něj čekají (například skok)
- **EU - Execution Unit** - výkonná jednotka, provádí instrukce (obsahuje ALU)
- **AU - Adress Unit** - adresovací jednotka. Tvoří fyzickou adresu ze segmentu a ofsetu nebo ze selektoru a ofsetu v chráněném režimu. Zajišťuje ochranu paměti před neoprávněnými přístupy v chráněném režimu.

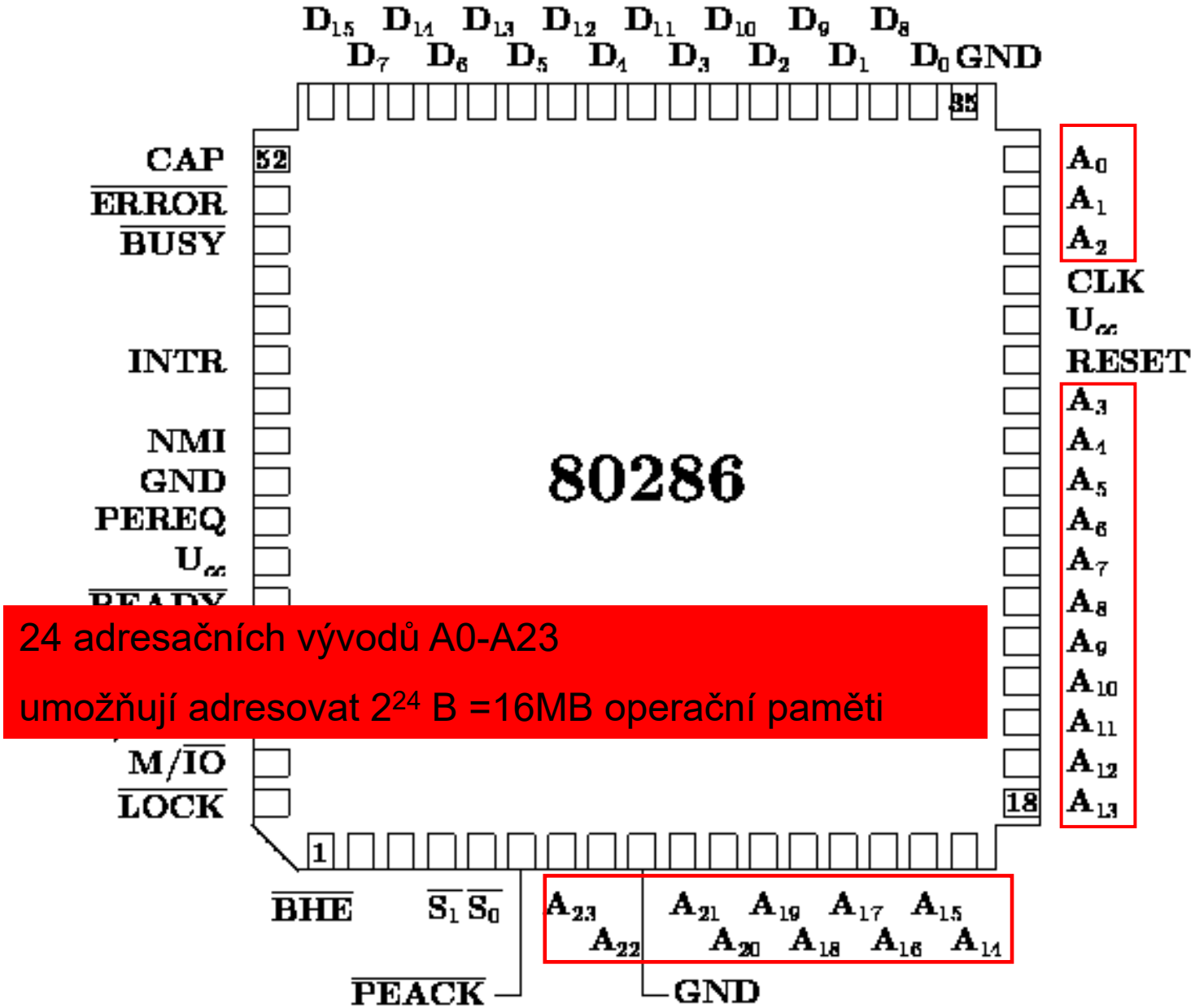
3-stupňový pipelining

- Pipelining = proudové zpracování
- Zpracování instrukce probíhá ve třech fázích Fetch - Decode - Execute



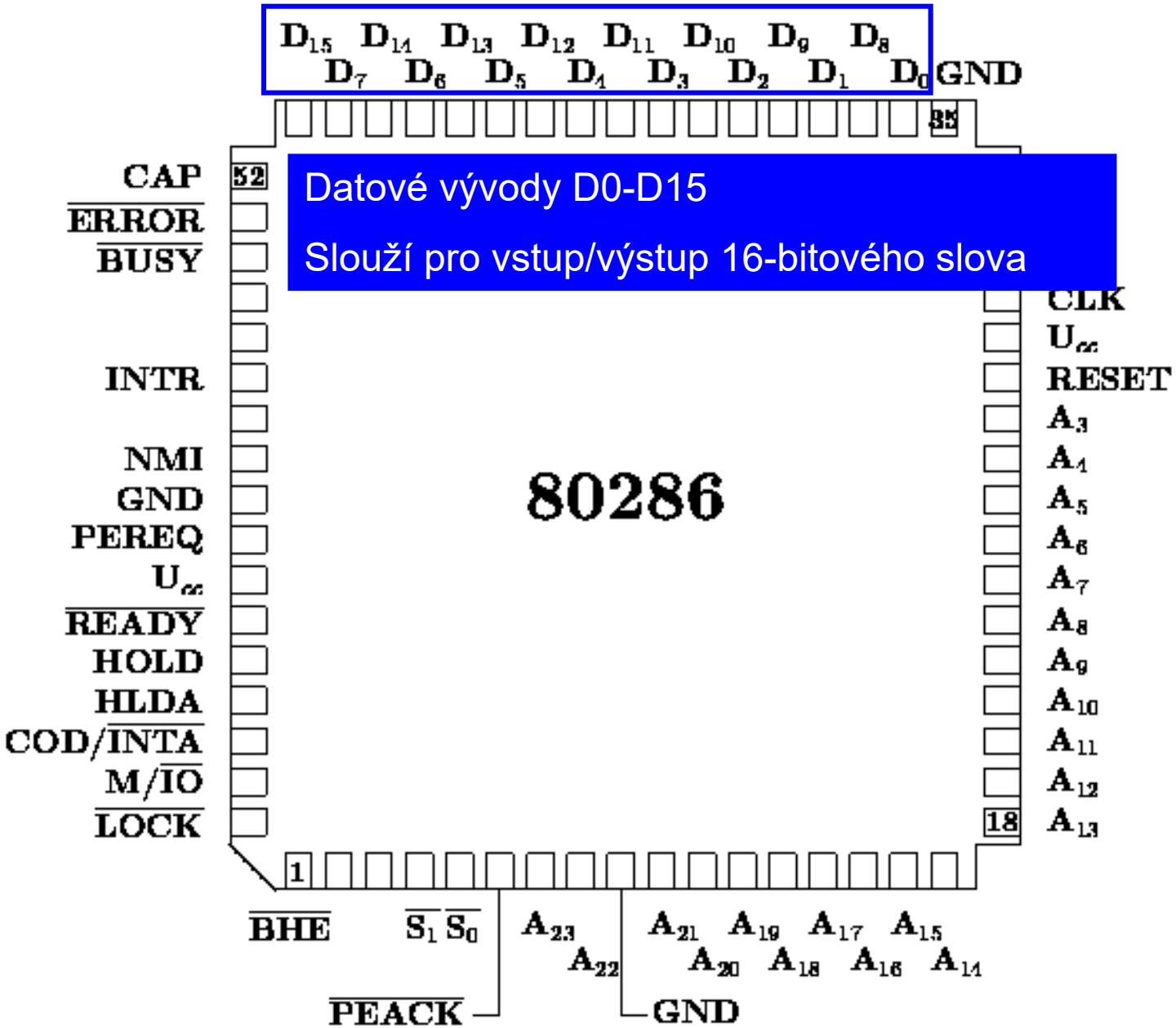
Koprocesor 80287

- Matematický koprocesor
- Jde vlastně o specializovaný mikroprocesor, pracující s čísly v plovoucí řádové čárce (odborný název pro reálná čísla) – Floating point operations
- má vestavěny 80 bitů široké registry
- Koprocesor je v určitých operacích, jako jsou například výpočty hodnot některých funkcí (sinus, kosinus, logaritmus) až 20x rychlejší než procesor
- mikroprocesor poté, co předá koprocesoru nějaký výpočet, získá čas na jinou činnost

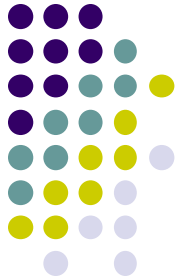


24 adresačních vývodů A0-A23

umožňují adresovat 2^{24} B =16MB operační paměti

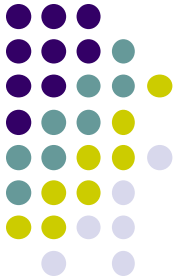

D15 D14 D13 D12 D11 D10 D9 D8
D7 D6 D5 D4 D3 D2 D1 D0 GND
Datové vývody D0-D15
Slouží pro vstup/výstup 16-bitového slova
80286
CAP
ERROR
BUSY
INTR
NMI
GND
PEREQ
Ucc
READY
HOLD
HLDA
COD/INTA
M/IO
LOCK
CLK
Ucc
RESET
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
BHE
S1 S0
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
PEACK
GND

Reálný režim



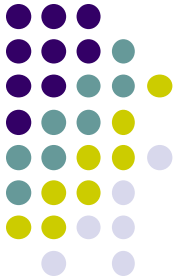
- Základní režim (procesor se v něm nachází po zapnutí)
- Pokud procesor pracuje v tomto režimu, pak je plně kompatibilní se svým předchůdcem i8086. Jediný rozdíl je v tom, že je rychlejší (instrukce mají jiné časování, vyžadují jiný počet taktů)
- Používá se 20-bitová a adresa generovaná metodou segment:offset a tak lze adresovat **jen 1 MB** paměti (stejně jako u 8086)
- Všechny mikroprocesory řady x86 se po zapnutí nebo po resetu dostanou do reálného režimu
- Tedy i dnešní nejmodernější mikroprocesory se po svém zapnutí chovají jako prastarý i8086, dokud není zaveden operační systém, který je přepne do jiného složitějšího a výkonnějšího režimu

Chráněný režim



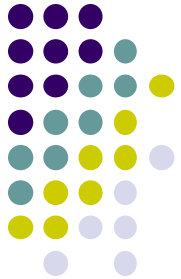
- Chráněný režim vznikl kvůli podpoře víceúlohového prostředí
- **Multitasking** = současné provádění více programů
- Současné provádění více programů je pouze iluzí
- Skutečnost je taková, že strojový kód více programů je uložen v paměti v několika různých kódových segmentech a jednotlivé úlohy se velmi rychle přepínají (přibližně v milisekundových intervalech) a je jim přidělován „strojový čas“.
- Při přepnutí je třeba uložit stav rozpracované přerušované úlohy
- Za přepínání mezi úlohami je zodpovědné jádro operačního systém
- Operační systém potřebuje mít k dispozici prostředky, které by zabraňovaly nežádoucímu vzájemnému ovlivňování úloh
- Každá úloha musí běžet izolovaně, nezávisle na ostatních – úlohy si nesmí škodit (přepisovat si navzájem data nebo strojový kód)
- Tyto prostředky jsou operačnímu systému k dispozici právě ve chráněném režimu
- Bez chráněného režimu by se nedal realizovat stabilní multitasking

Chráněný režim



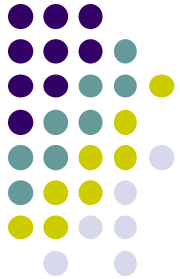
- Co tedy vlastně chráněný režim přináší:
  - **Vzájemná ochrana úloh** – ochrana před nežádoucím přepsáním paměťových oblastí přiřazených jednotlivým úlohám
  - **Podpora přepínání úloh** – uložení a obnovení stavu úloh (vždy běží jedna úloha, ostatní jsou přerušené, úlohy se velmi rychle střídají)
  - **Privilegování operačního systému** při provádění určitých instrukcí (OS může vše, může přistupovat kamkoliv do paměti a ke vstupu a výstupu, běžné úlohy nikoliv)
  - Podpora pro práci s „**virtuální pamětí**“ – lze vytvořit a programům přidělit více segmentů, než kolik se skutečně může vejít do operační paměti

Chráněný režim a paměť



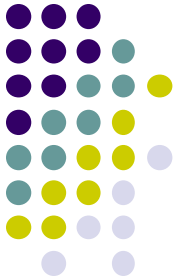
- Informace o každém **segmentu** jsou uloženy v jeho **deskriptoru**
- **Deskriptor** obsahuje informaci o tom, na jaké adrese segment začíná, jak je velký a o jaký typ segmentu se jedná (datový, kódový…)
- Dále jsou v každém deskriptoru nastavena **přístupová práva** pro daný segment – do některých segmentů například nelze zapisovat, pro přístup k některým segmentům je třeba mít patřičnou úroveň oprávnění
- Díky deskriptorům je v paměti pořádek – víme, kde leží data, kde leží strojový kód, a který program smí pracovat se kterými úseky paměti
- Přestože jde o Von Neumannovu architekturu, program již nemůže přepsat sám sebe ani nemůže dojít ke „spuštění dat“ místo strojového kódu

Tabulka deskriptorů



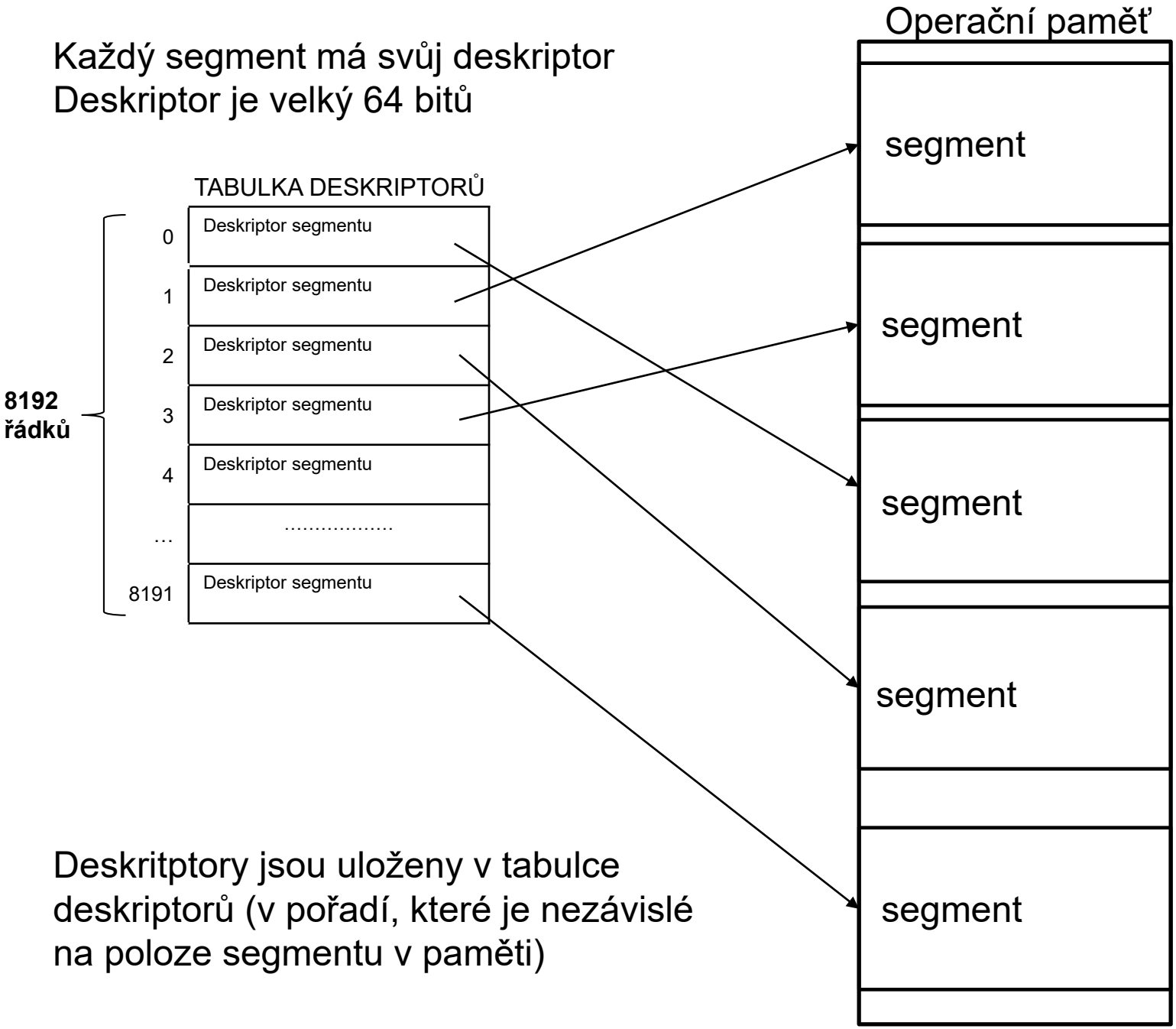
- Pro práci s pamětí jsou vytvořeny operačním systémem **tabulky popisu paměti** (tabulky **deskriptorů**)
- Každý segment má svůj deskriptor v tabulce deskriptorů
- Segmentů mohou být stovky, tisíce…
- Program si vybírá „z nabídky vytvořených segmentů“, se kterým chce pracovat – vybírá si v tabulce deskriptorů jeho deskriptor
- Výběr deskriptoru požadovaného segmentu se provádí **selektorem**
- **Selektor** je registr CS, DS, ES, SS (dříve segmentové registry)
- Například selektorem DS se vybírá datový segment (z tabulky deskriptorů se vybere deskriptor segmentu, který bude program dále používat jako aktuální datový segment)

Tabulka deskriptorů



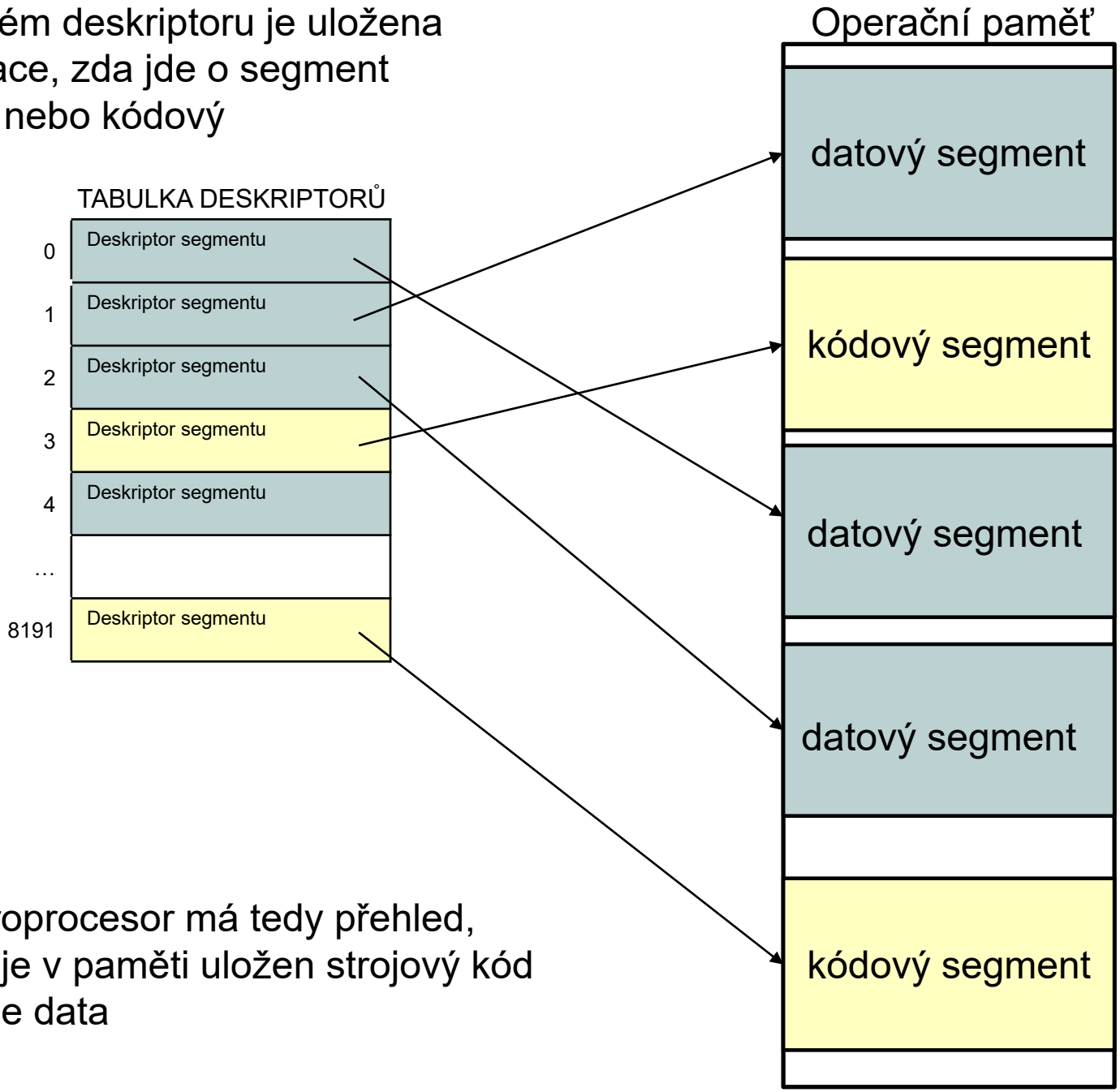
- Ukazatel do tabulky deskriptorů = **index** - hodnota uvedená v **selektoru** - z tabulky vybere řádek s deskriptorem segmentu, se kterým chce program pracovat
- Z vybraného deskriptoru se procesor dozví 24-bitovou počáteční adresu segmentu
- K počáteční adrese segmentu se přičte hodnota offsetu
- Každá položka v tabulkách popisu paměti (1 deskriptor) nese informace o jednom segmentu (nejen jeho počáteční adresu, ale mnoho dalších informací potřebných pro podporu víceúlohového prostředí)
- V tabulce deskriptorů lze definovat až 8 192 (2^{13}) deskriptorů – tedy informace o 8192 segmentech

Každý segment má svůj deskriptor
Deskriptor je velký 64 bitů



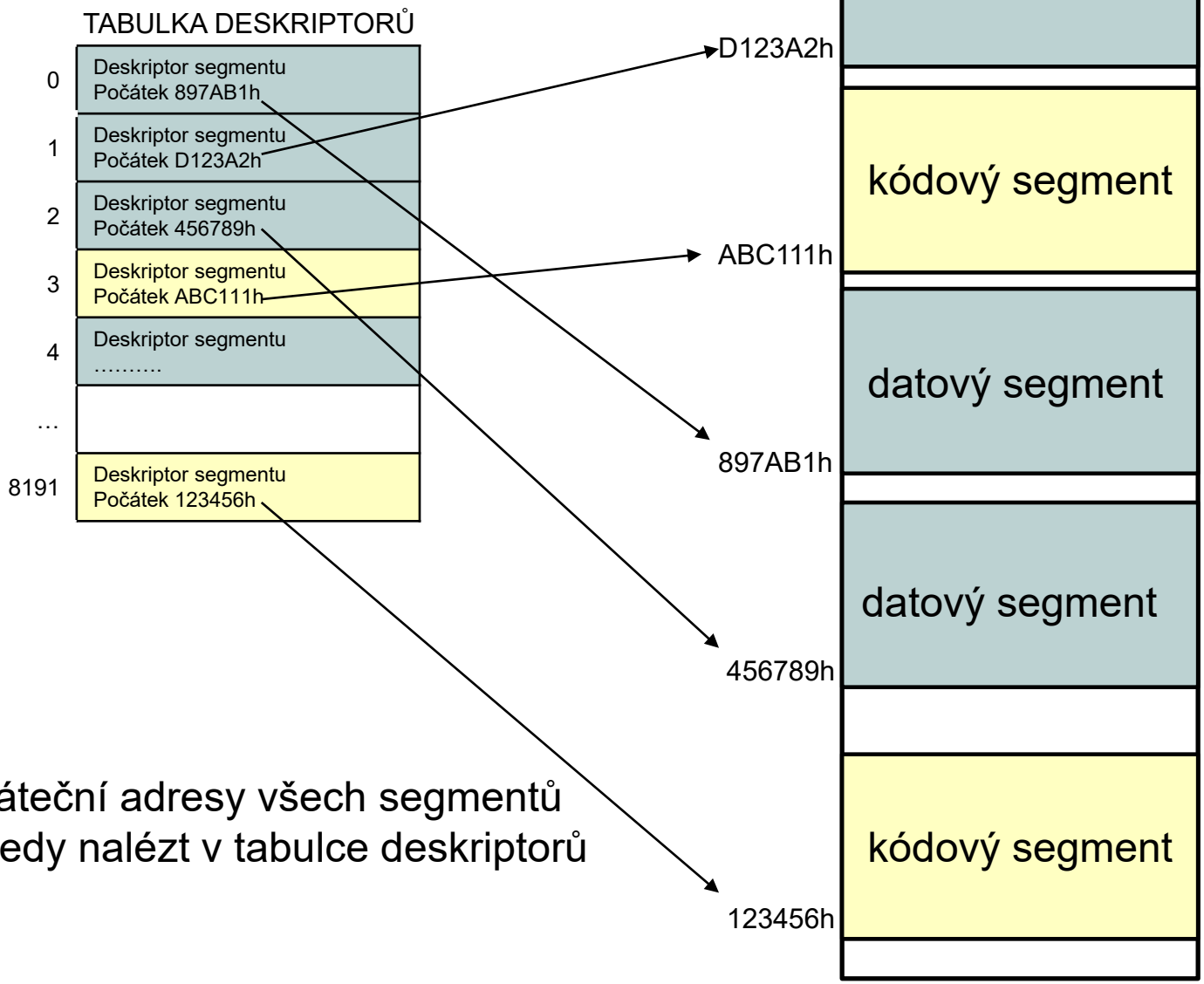
Deskritptory jsou uloženy v tabulce deskriptorů (v pořadí, které je nezávislé na poloze segmentu v paměti)

V každém deskriptoru je uložena informace, zda jde o segment datový nebo kódový



Mikroprocesor má tedy přehled, kde je v paměti uložen strojový kód a kde data

V každém deskriptoru je mimo jiné uvedena také **počáteční adresa segmentu**

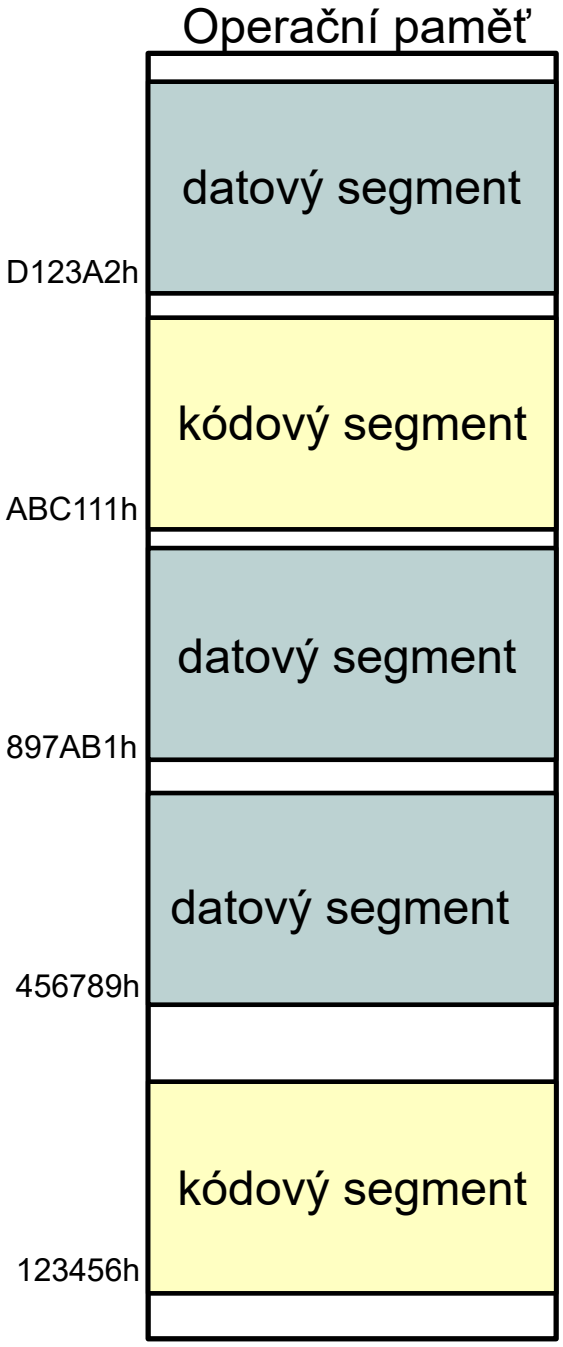


Počáteční adresy všech segmentů lze tedy nalézt v tabulce deskriptorů

V každém deskriptoru je mimo jiné uvedena také **počáteční adresa segmentu**

TABULKA DESKRIPTORŮ

0	Deskriptor segmentu Počátek 897AB1h
1	Deskriptor segmentu Počátek D123A2h
2	Deskriptor segmentu Počátek 456789h
3	Deskriptor segmentu Počátek ABC111h
4	Deskriptor segmentu …….
…	
8191	Deskriptor segmentu Počátek 123456h

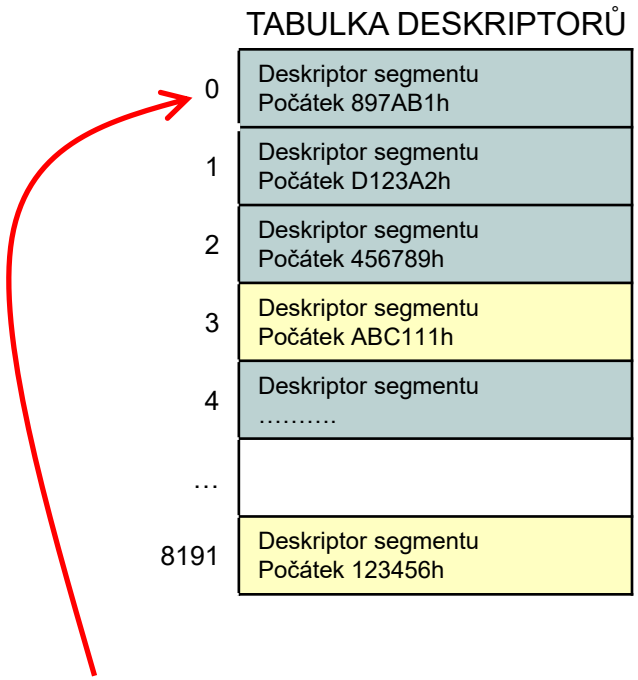


MOV [5],AL

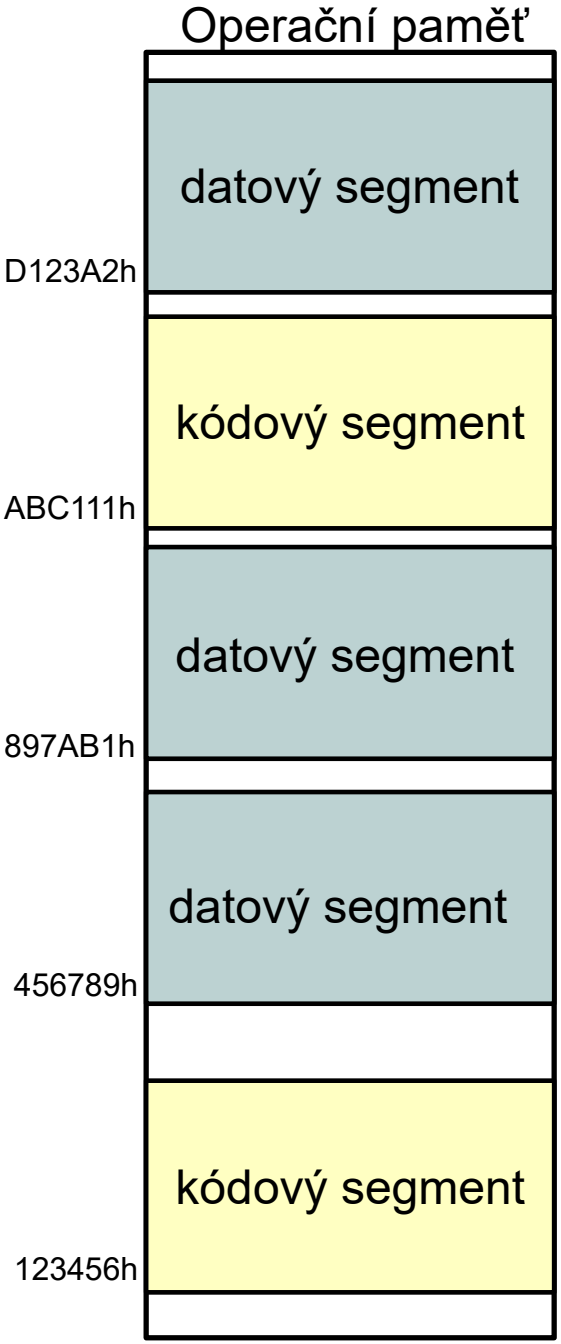
Tato instrukce zapíše obsah registru AL do datového segmentu na pozici s offsetem 5.
Ale do kterého datového segmentu ????

Záleží na tom, který datový segment je vybrán selektorem DS

V každém deskriptoru je mimo jiné uvedena také **počáteční adresa segmentu**

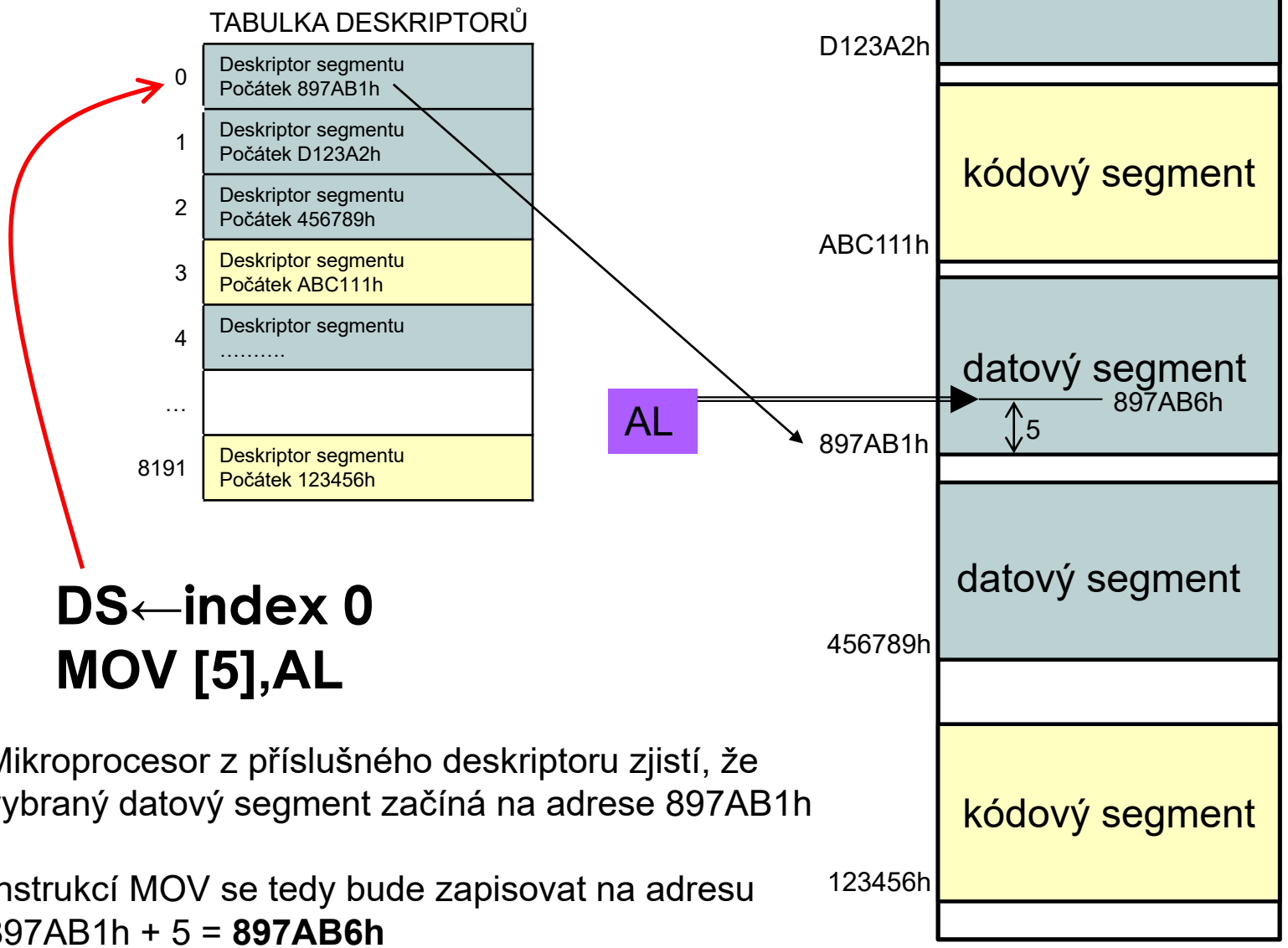


DS←index 0
MOV [5],AL



V selektoru DS je index=0. Vybraný je deskriptor, který leží v tabulce na 0. řádku Tento deskriptor obsahuje informace o datovém segmentu, který byl právě vybrán

Instrukce MOV provede zápis do vybraného datového segmentu

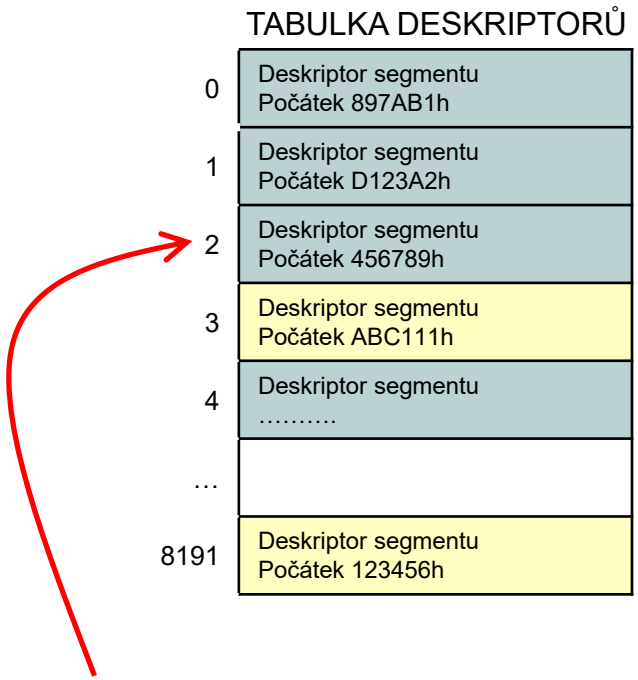


DS←index 0
MOV [5],AL

Mikroprocesor z příslušného deskriptoru zjistí, že vybraný datový segment začíná na adrese 897AB1h

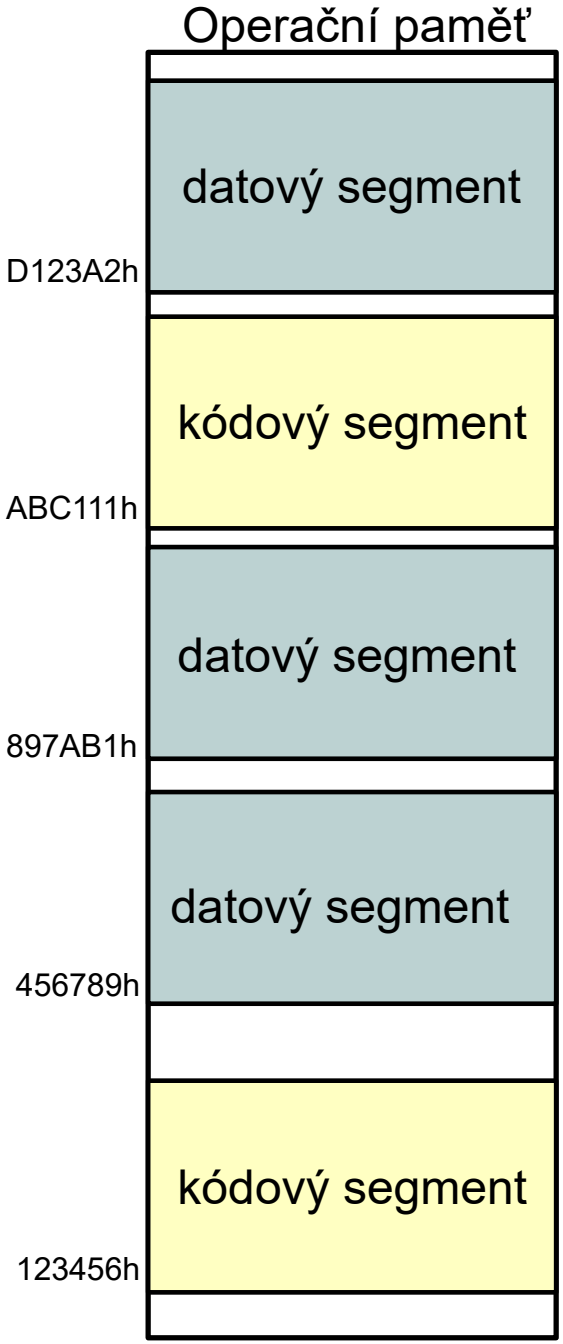
Instrukcí MOV se tedy bude zapisovat na adresu 897AB1h + 5 = **897AB6h**

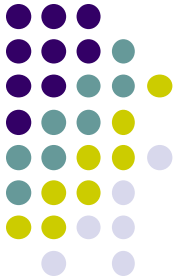
Teď je selektorem DS vybraný jiný datový segment, než v minulém příkladu



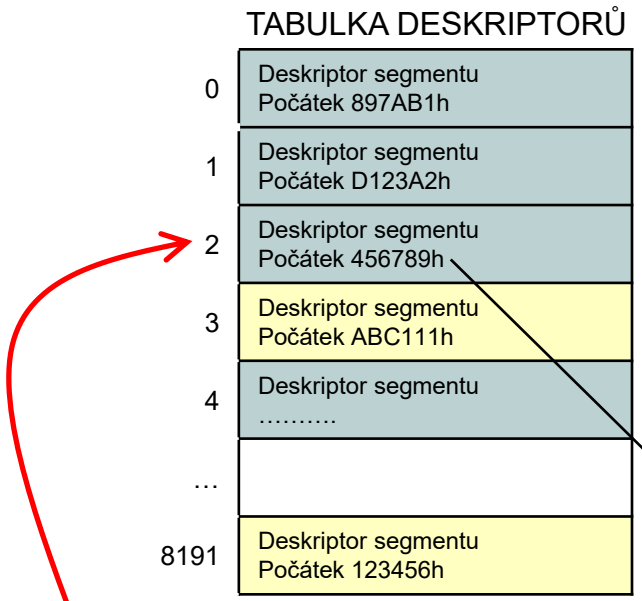
DS←index 2
MOV [5],AL

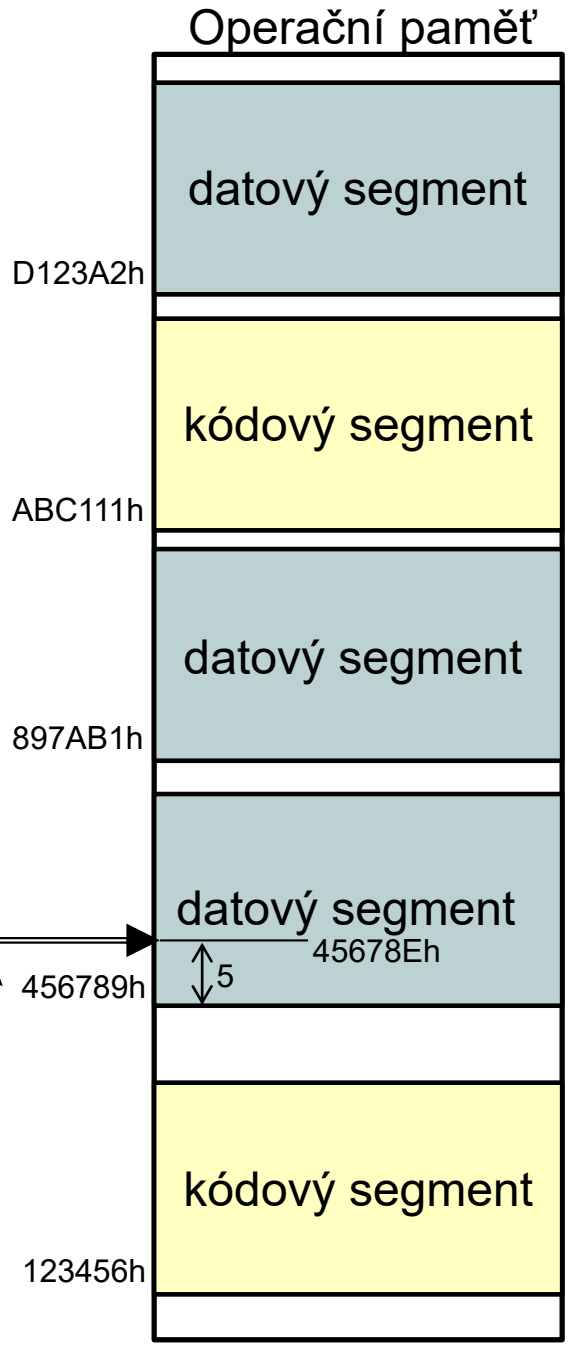
V selektoru DS je index=2. Vybraný je deskriptor, který leží v tabulce na 2. řádku





Instrukce MOV provede zápis do vybraného datového segmentu





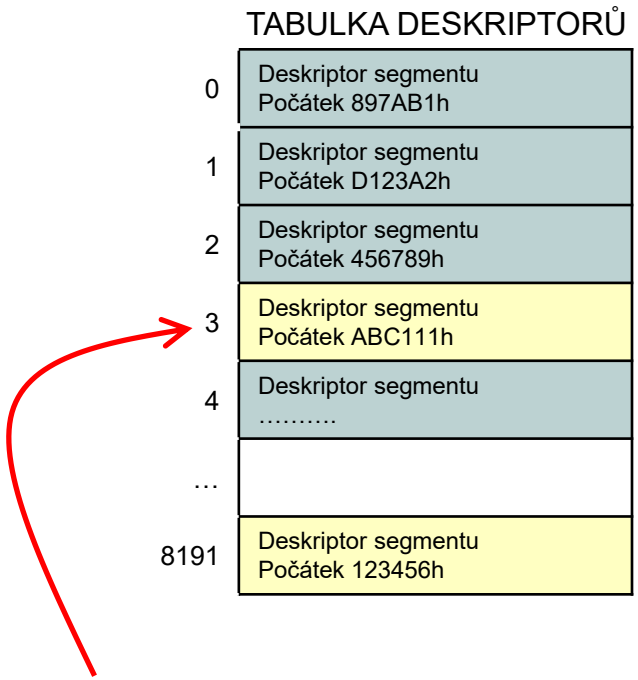
AL

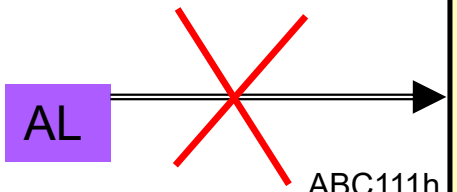
DS←index 0
MOV [5],AL

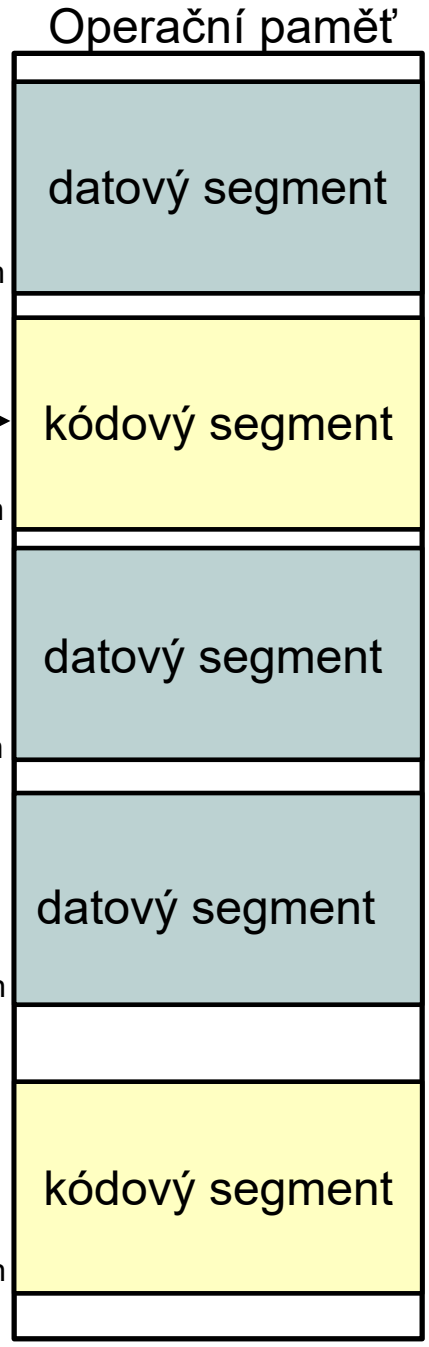
Mikroprocesor z příslušného deskriptoru zjistí, že vybraný datový segment začíná na adrese 456789h

Instrukcí MOV se tedy bude zapisovat na adresu 456789h + 5 = **45678Eh**

Co by se stalo, kdyby se program pokusil zapsat do **kódového** segmentu?





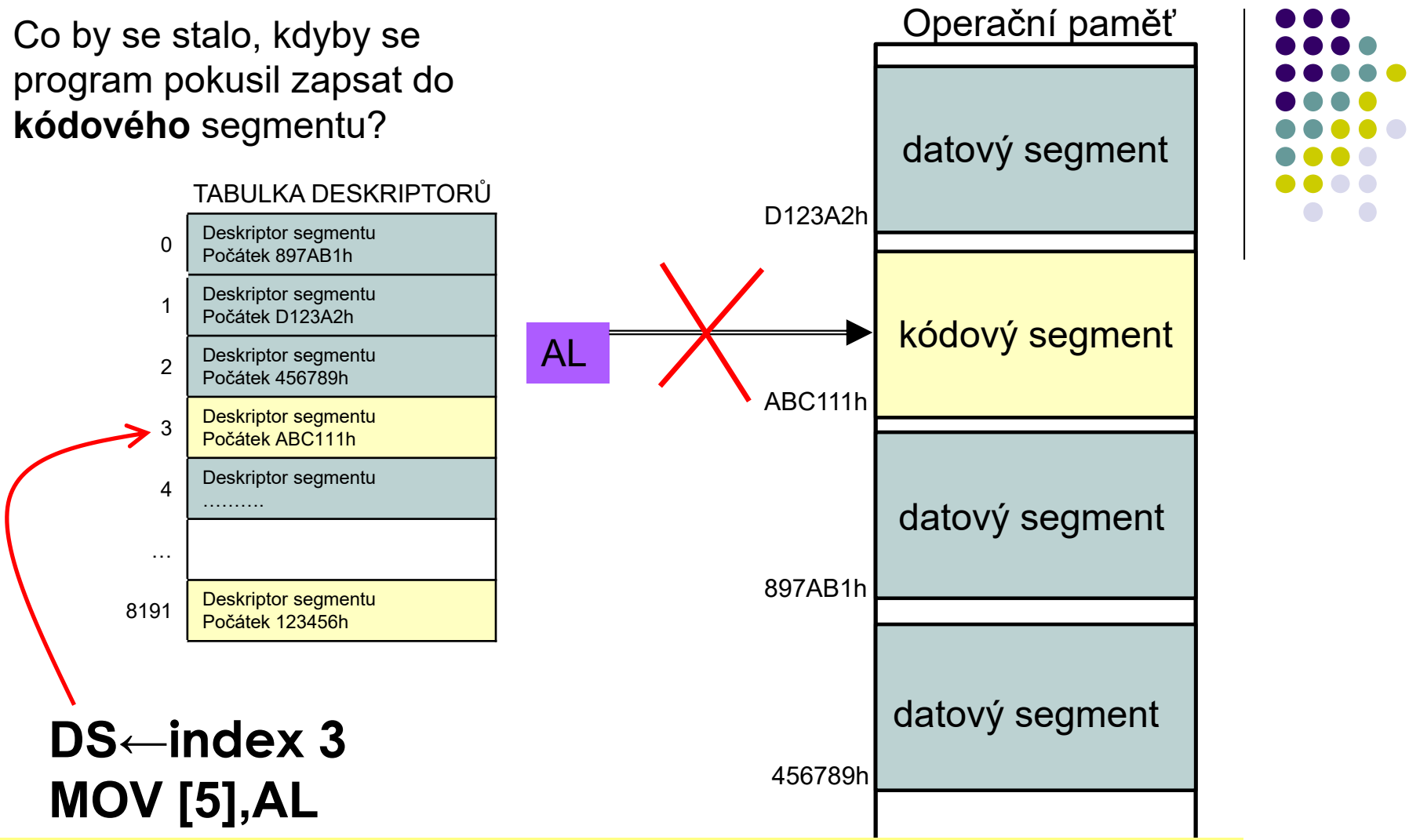


DS←index 3
MOV [5],AL

V selektoru DS je index=3. Vybraný je deskriptor, který leží v tabulce na 3. řádku

Mikroprocesor si v deskriptoru přečte, že ve vybraném segmentu je uložen strojový kód

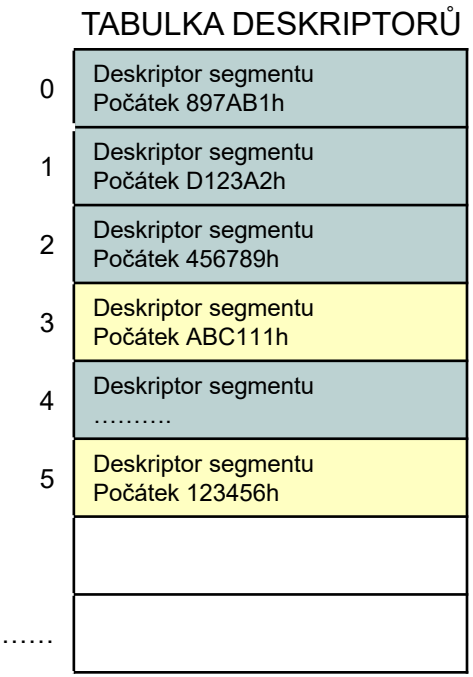
Co by se stalo, kdyby se program pokusil zapsat do **kódového** segmentu?



DS←index 3
MOV [5],AL

Do kódového segmentu **nelze zapsat** (program by tak mohl přepsat svůj vlastní strojový kód nebo strojový kód jiného programu). Mikroprocesor odmítne tuto instrukci MOV vykonat. Nastane přerušení vyvolané nepovoleným chováním programu. Program tedy bude přerušen a obsluhu provede operační systém. Ten vypíše chybovou hlášku (Např. Memory Access Violation, Obecná chyba ochrany paměti, Segmentation Fault, Program provedl neplatnou operaci a bude ukončen…..)

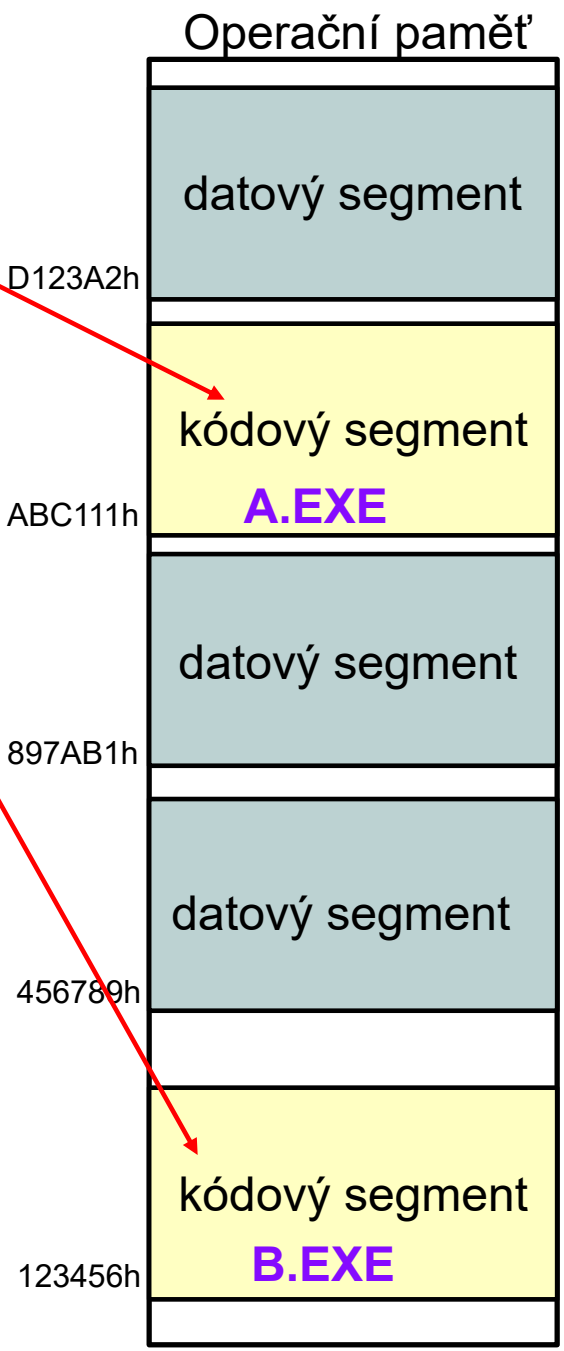
V paměti je uložen strojový kód více různých programů



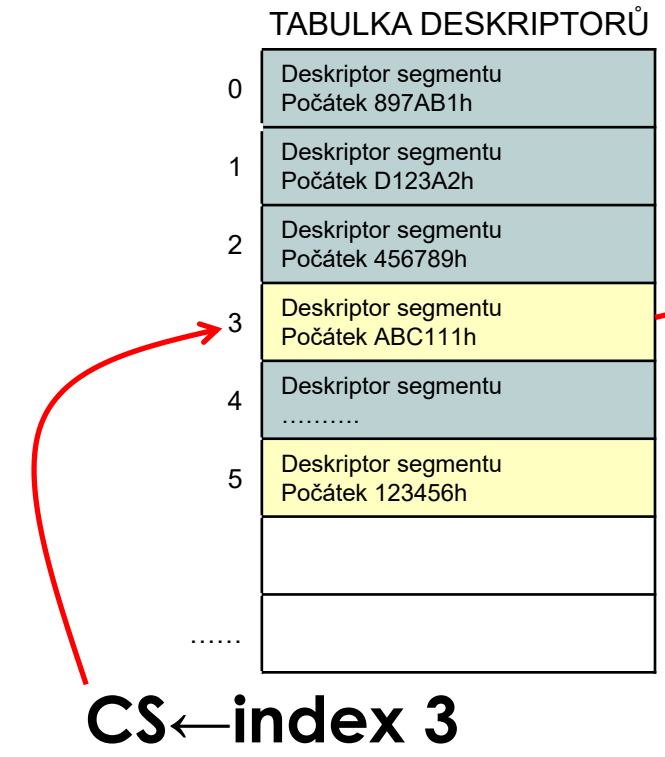
CS← ?

Selektorem CS vybereme deskriptor kódového segmentu, ve kterém je uložen strojový kód programu, který má právě běžet

Programy se v multitaskingu pravidelně střídají

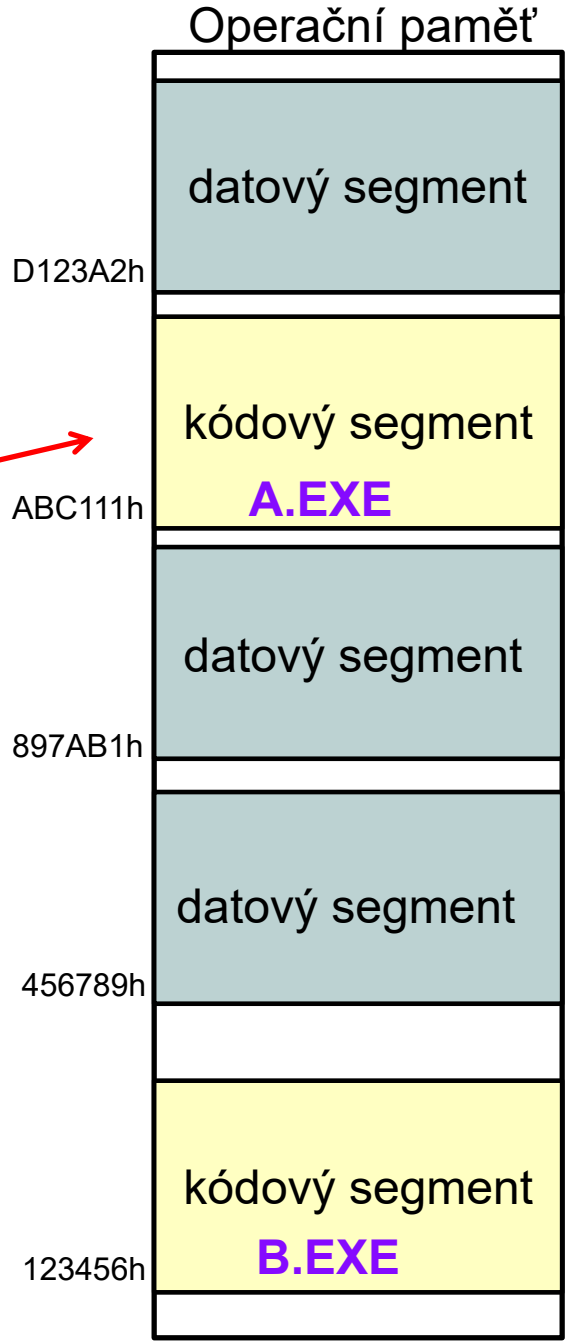


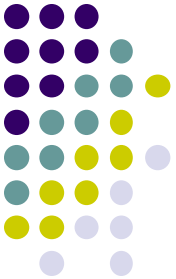
V paměti je uložen strojový kód
více různých programů



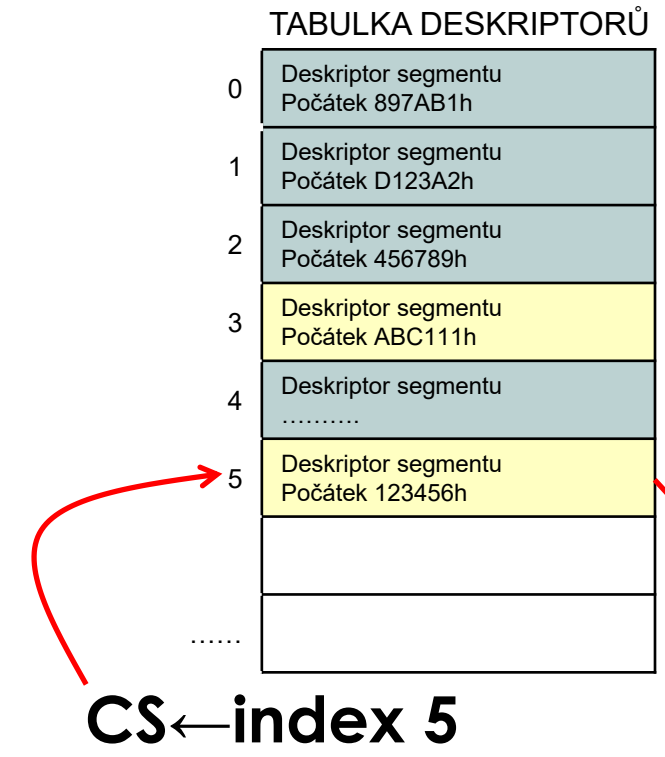
CS←index 3

Právě běží program **A.EXE**

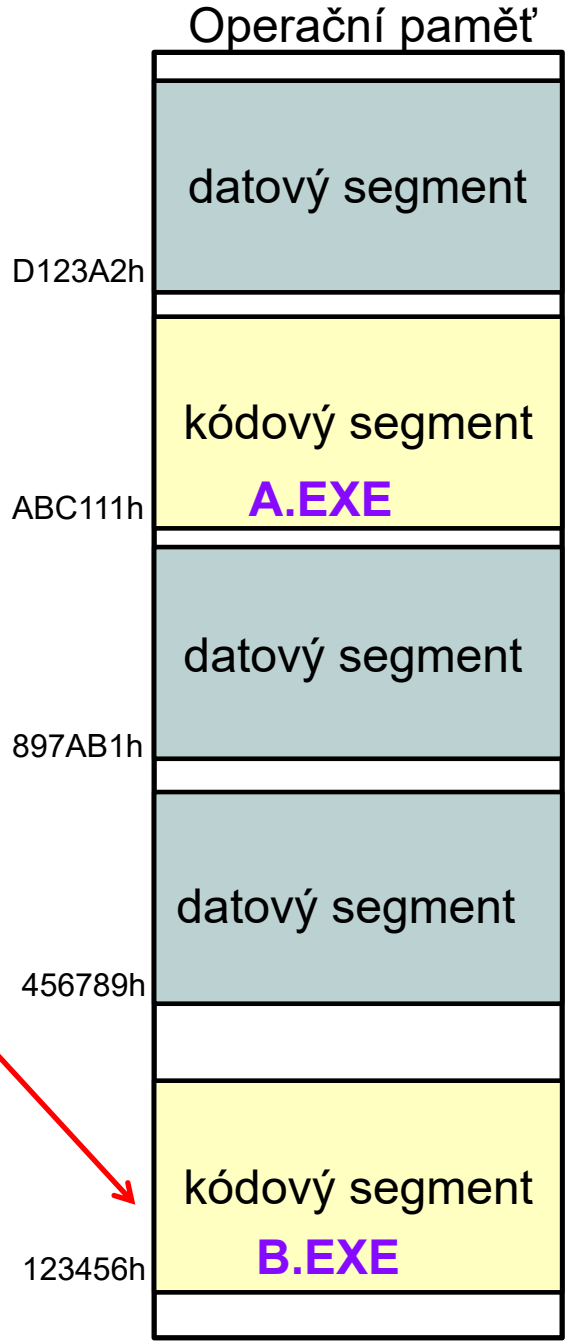


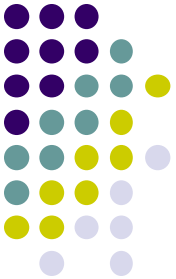


V paměti je uložen strojový kód více různých programů

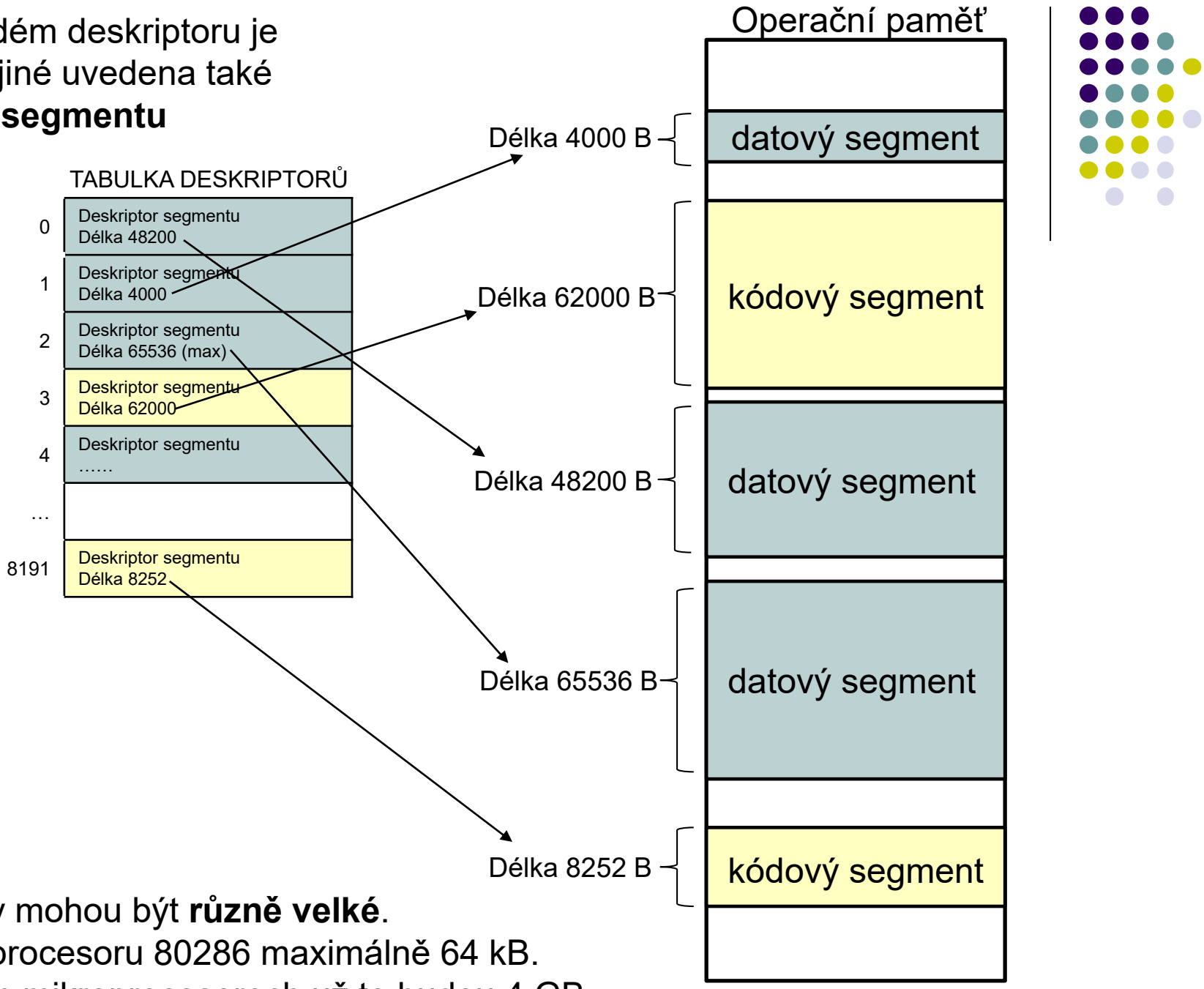


Právě běží program **B.EXE**



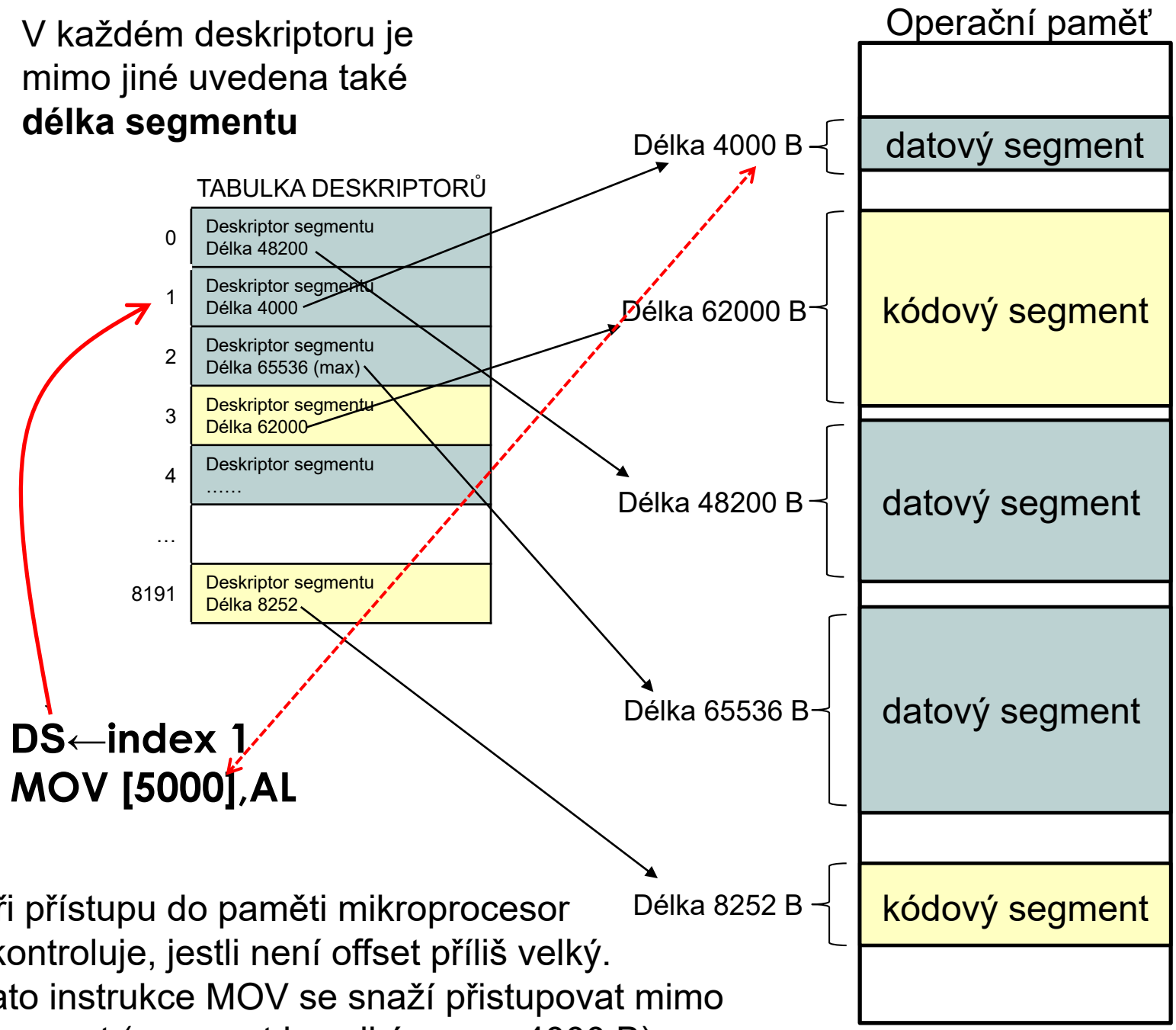


V každém deskriptoru je mimo jiné uvedena také **délka segmentu**



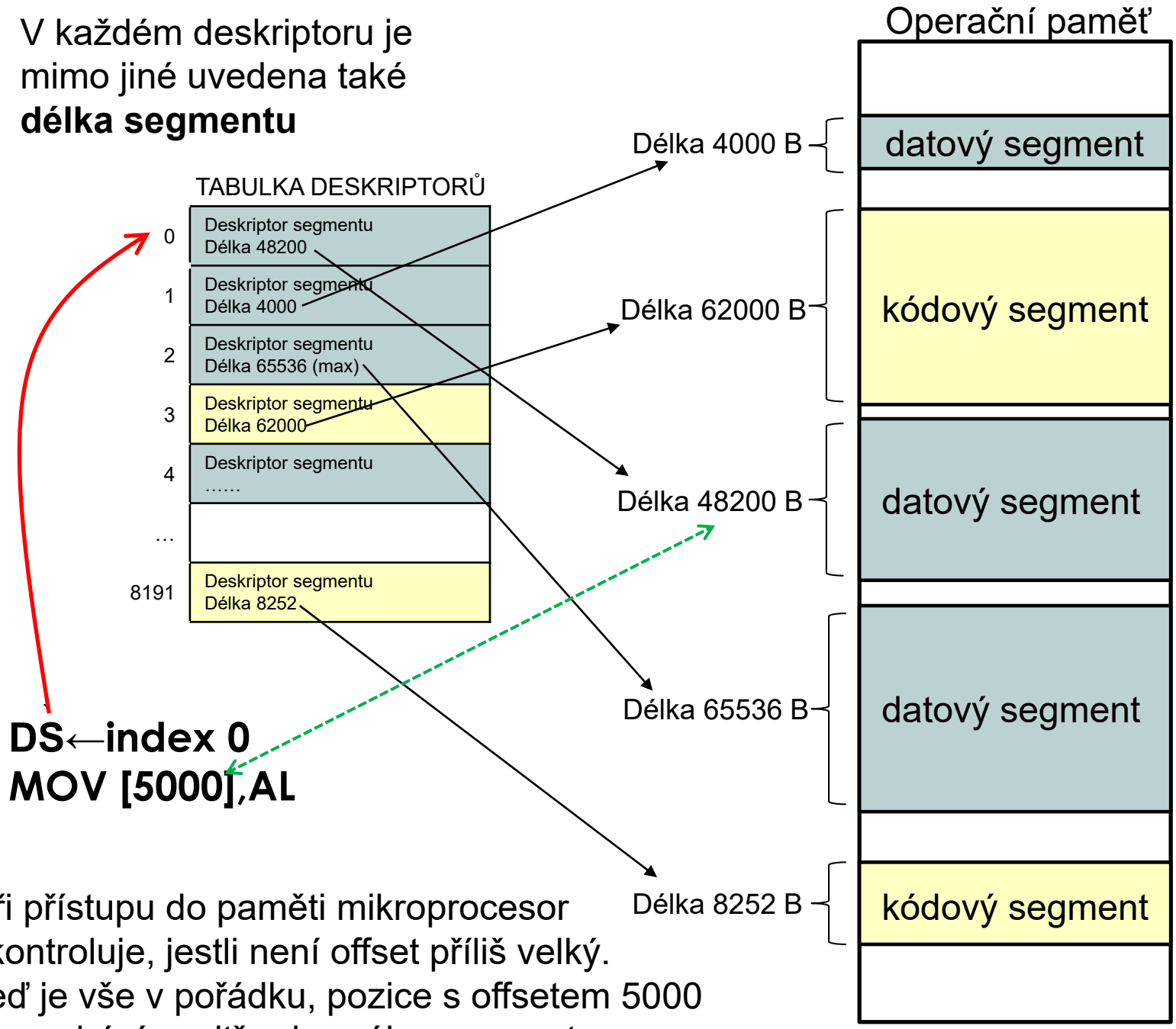
Segmenty mohou být **různě velké**.
Na mikroprocesoru 80286 maximálně 64 kB.
Na dalších mikroprocesorech už to budou 4 GB

V každém deskriptoru je mimo jiné uvedena také **délka segmentu**



Při přístupu do paměti mikroprocesor zkontroluje, jestli není offset příliš velký. Tato instrukce MOV se snaží přistupovat mimo segment (segment je velký pouze 4000 B)

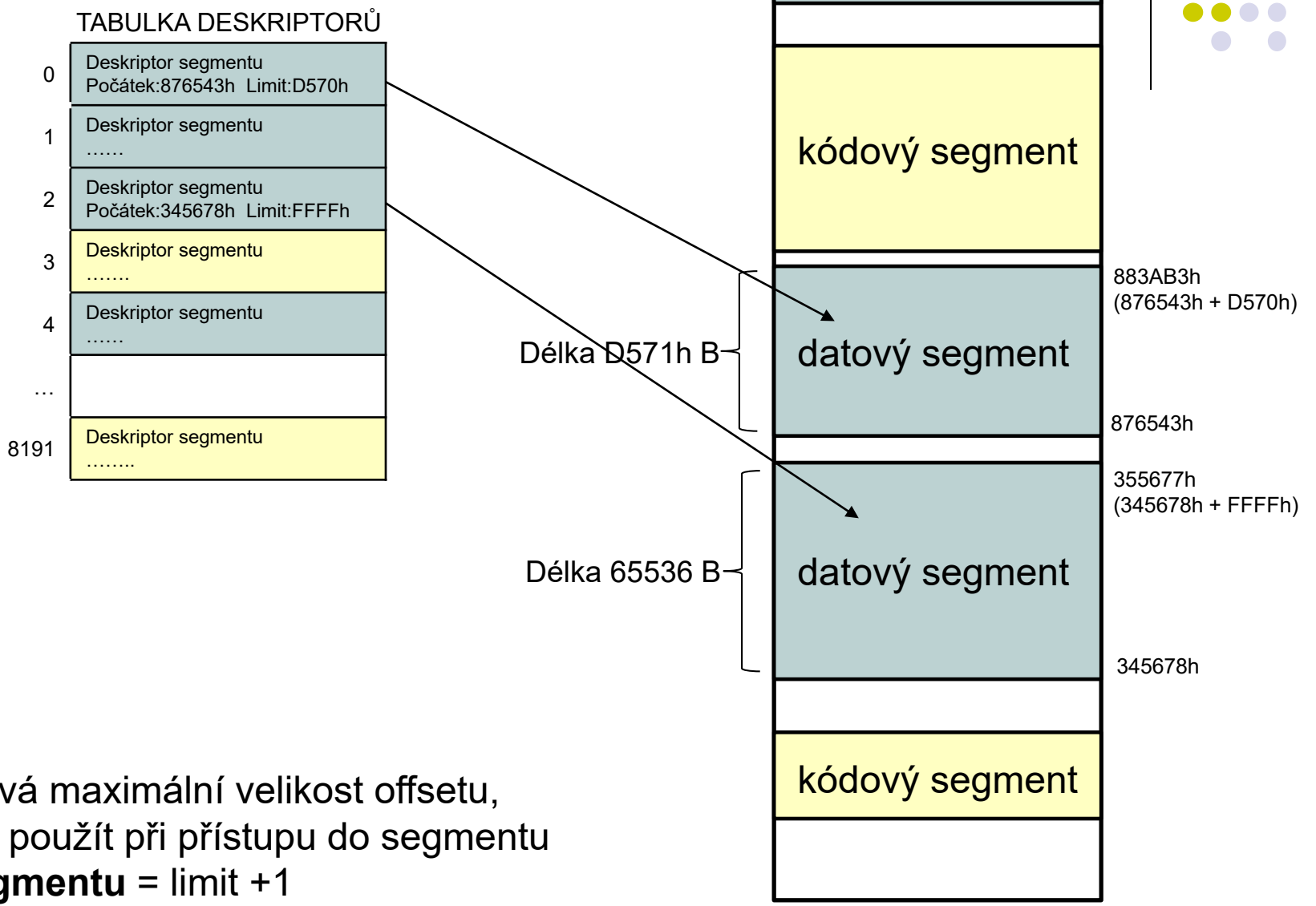
V každém deskriptoru je mimo jiné uvedena také **délka segmentu**



DS←index 0
MOV [5000],AL

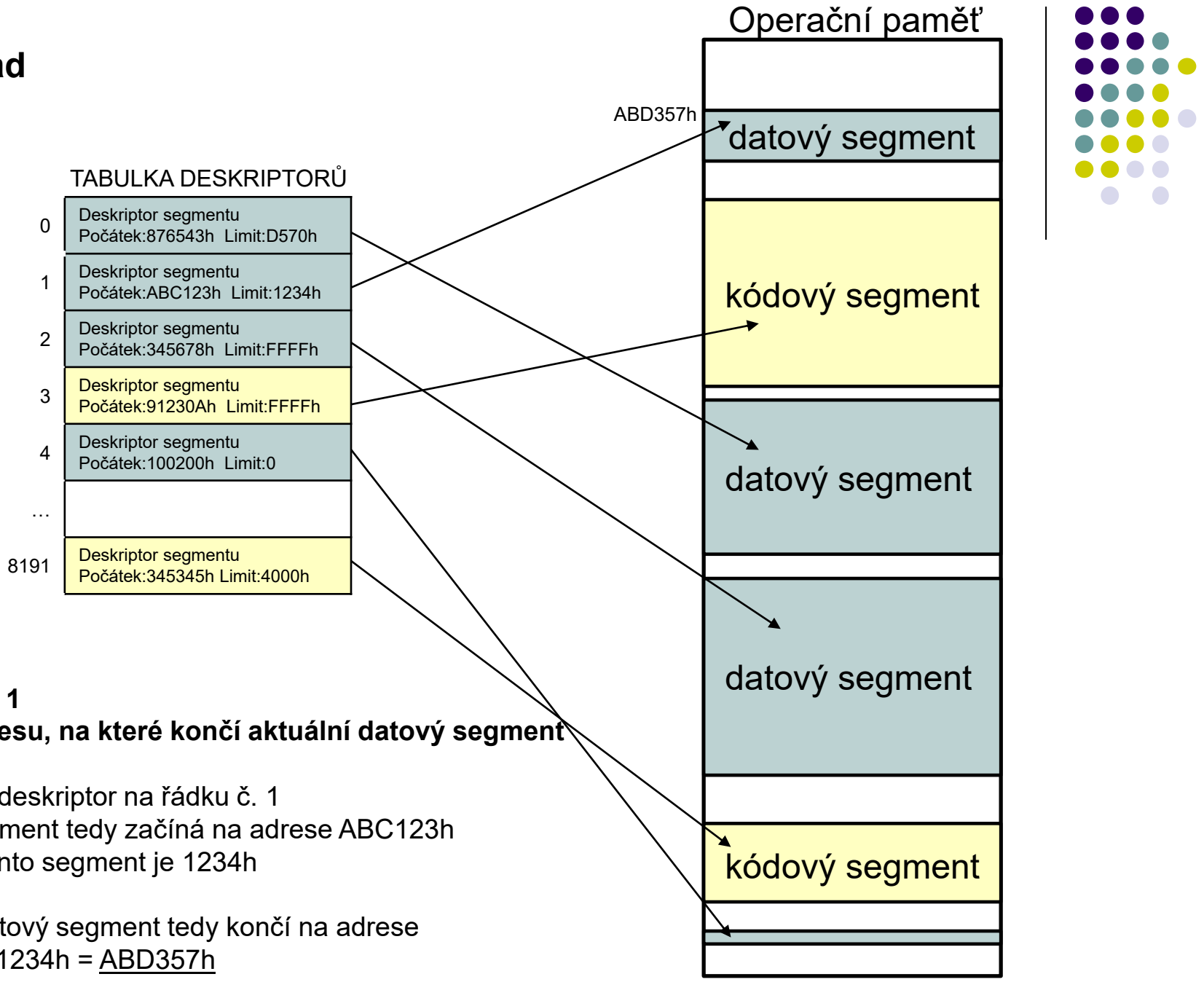
Při přístupu do paměti mikroprocesor zkontroluje, jestli není offset příliš velký. Teď je vše v pořádku, pozice s offsetem 5000 se nachází uvnitř vybraného segmentu

Ve skutečnosti není v deskriptoru uvedena *délka* segmentu, ale **limit**



Limit udává maximální velikost offsetu, kterou lze použít při přístupu do segmentu
Délka segmentu = limit +1
Koncová adresa segmentu = počáteční adresa + limit

Příklad

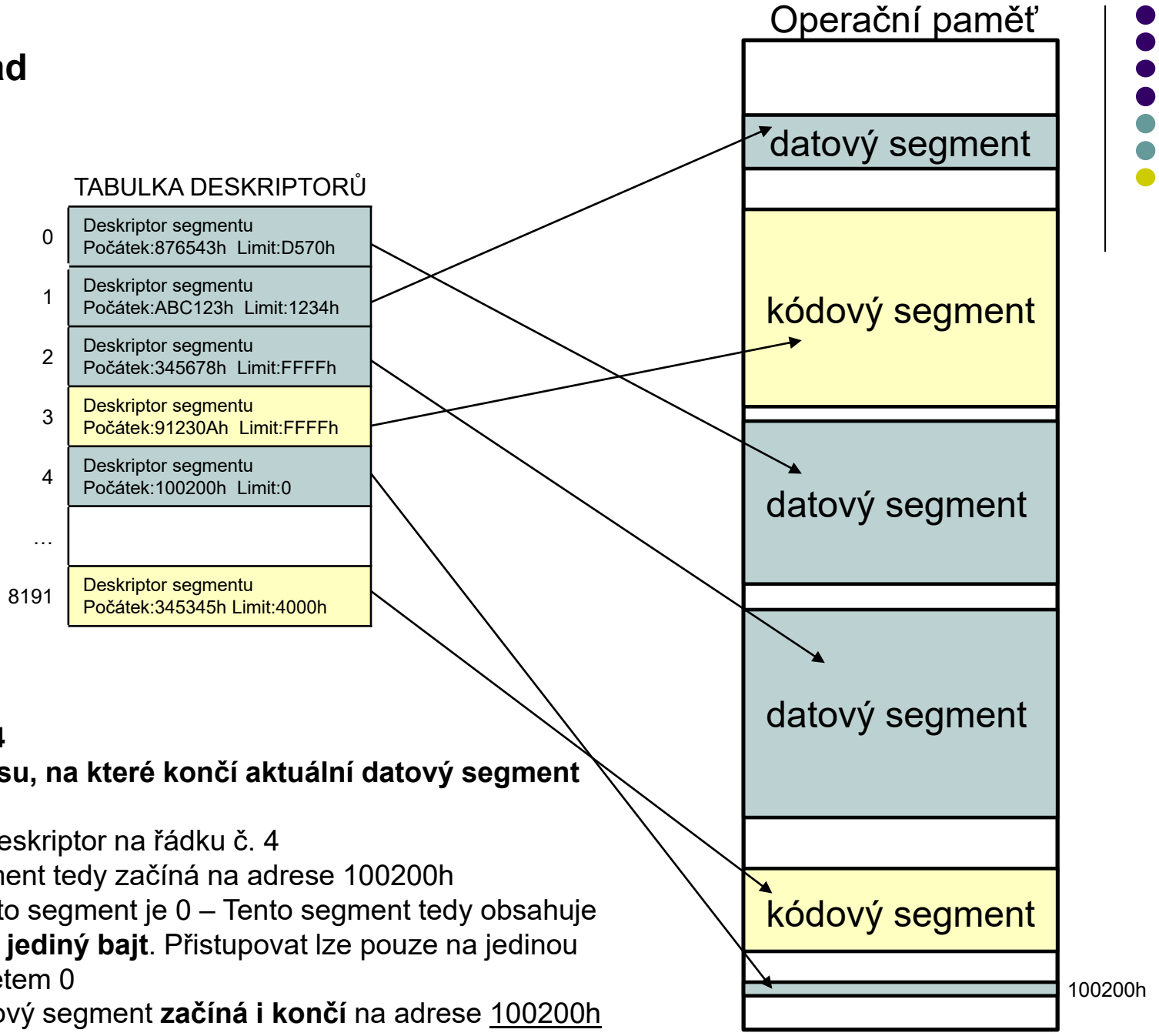


DS←Index 1
Určete adresu, na které končí aktuální datový segment

Vybraný je deskriptor na řádku č. 1
Datový segment tedy začíná na adrese ABC123h
Limit pro tento segment je 1234h

Vybraný datový segment tedy končí na adrese
ABC123h+1234h = ABD357h

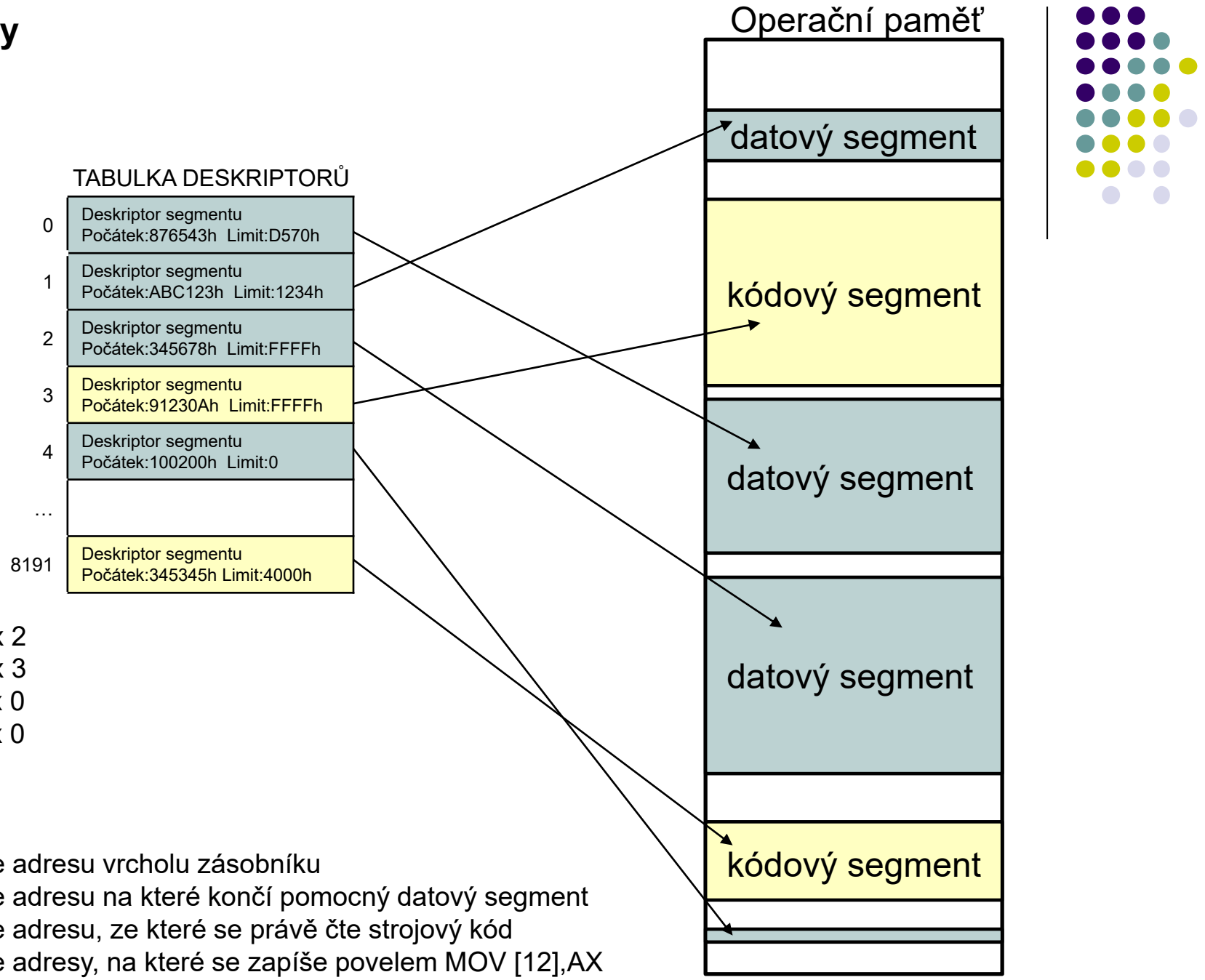
Příklad



DS←Index 4
Určete adresu, na které končí aktuální datový segment

Vybraný je deskriptor na řádku č. 4
Datový segment tedy začíná na adrese 100200h
Limit pro tento segment je 0 – Tento segment tedy obsahuje pouze **jeden jediný bajt**. Přistupovat lze pouze na jedinou pozici s offsetem 0
Vybraný datový segment **začíná i končí** na adrese 100200h

Příklady

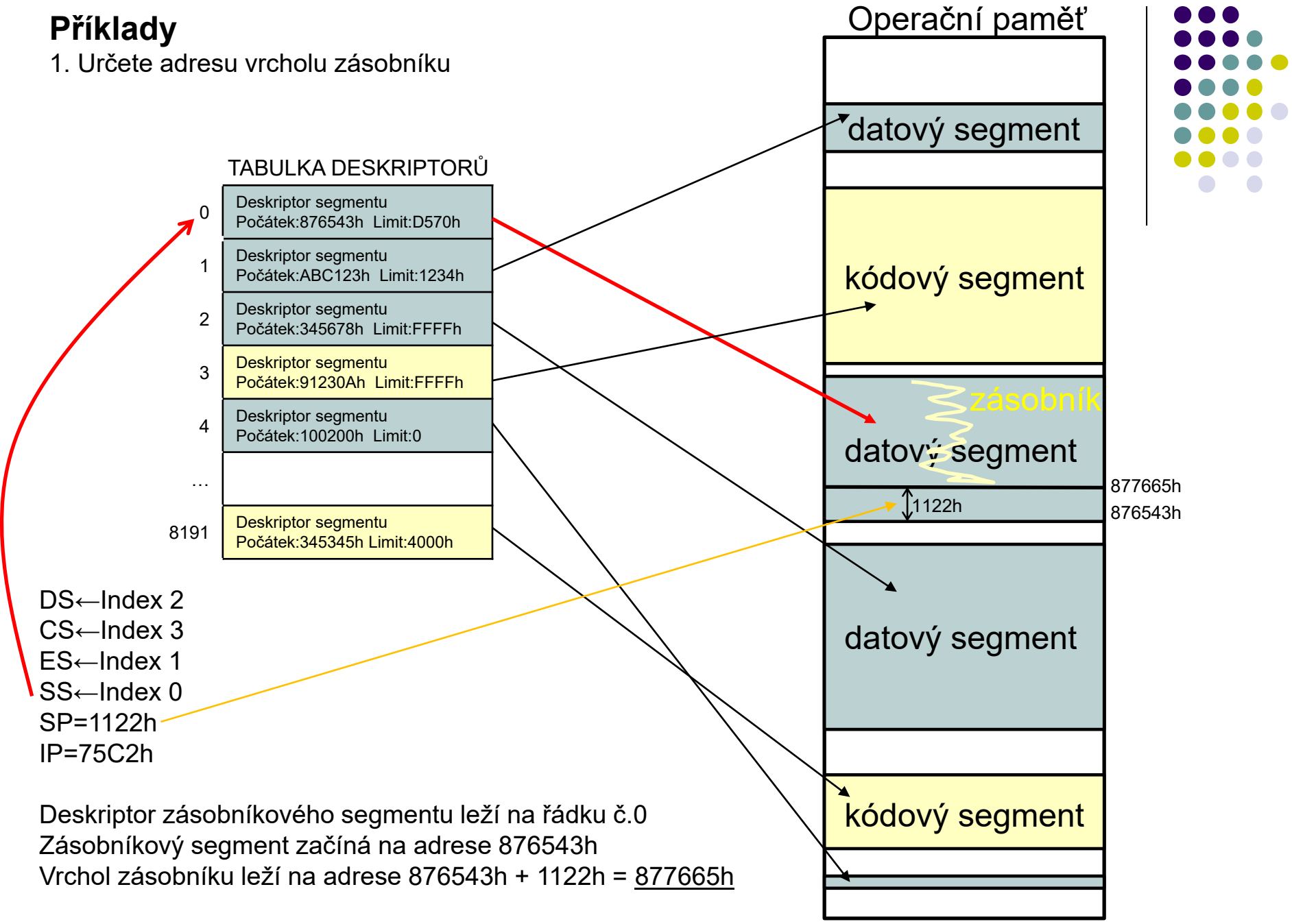


DS←Index 2
CS←Index 3
ES←Index 0
SS←Index 0
SP=1122h
IP=75C2h

1. Určete adresu vrcholu zásobníku
2. Určete adresu na které končí pomocný datový segment
3. Určete adresu, ze které se právě čte strojový kód
4. Určete adresy, na které se zapíše povelem MOV [12],AX

Příklady

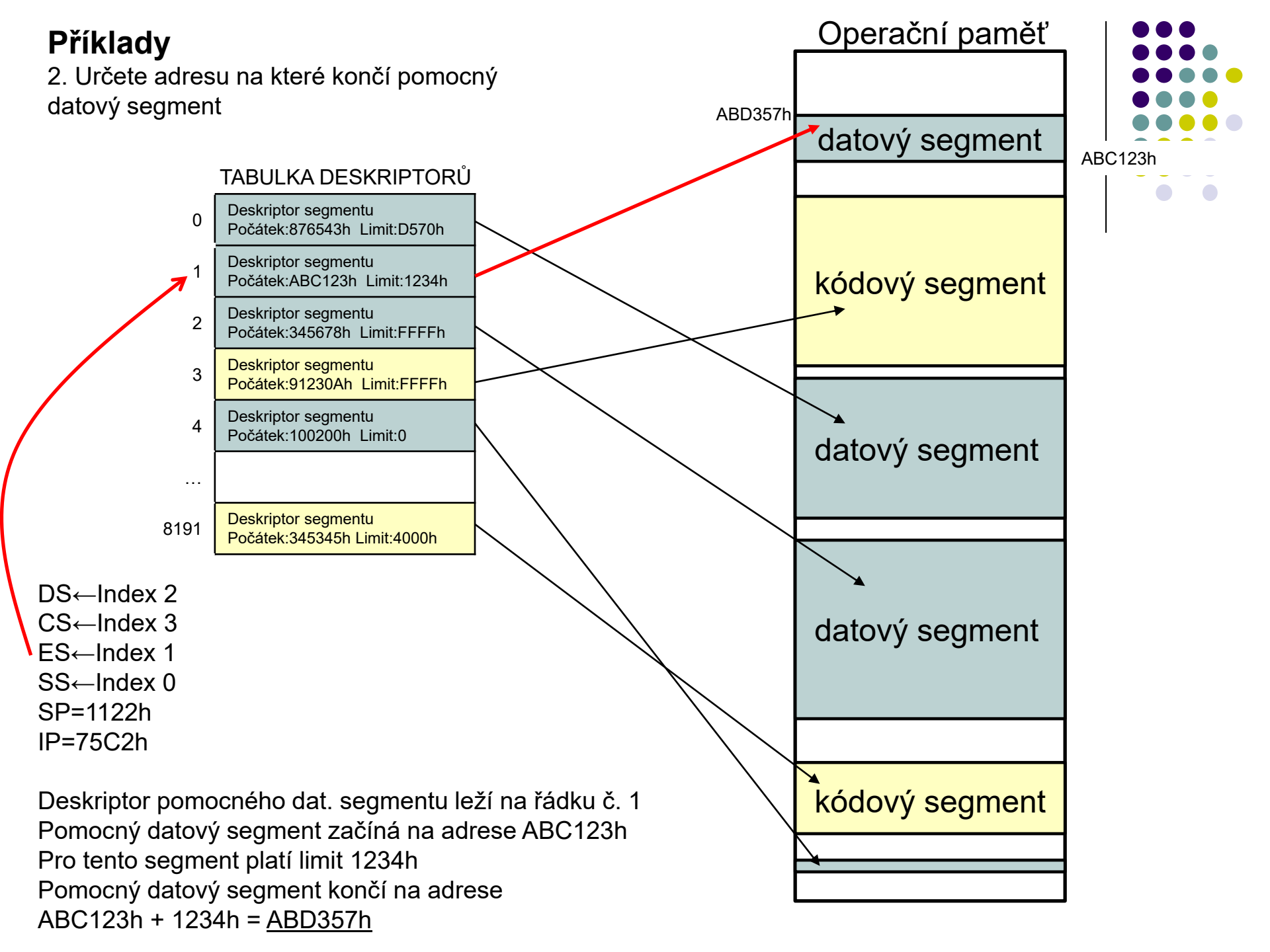
1. Určete adresu vrcholu zásobníku



Deskriptor zásobníkového segmentu leží na řádku č.0
Zásobníkový segment začíná na adrese 876543h
Vrchol zásobníku leží na adrese 876543h + 1122h = <u>877665h</u>

Příklady

2. Určete adresu na které končí pomocný datový segment

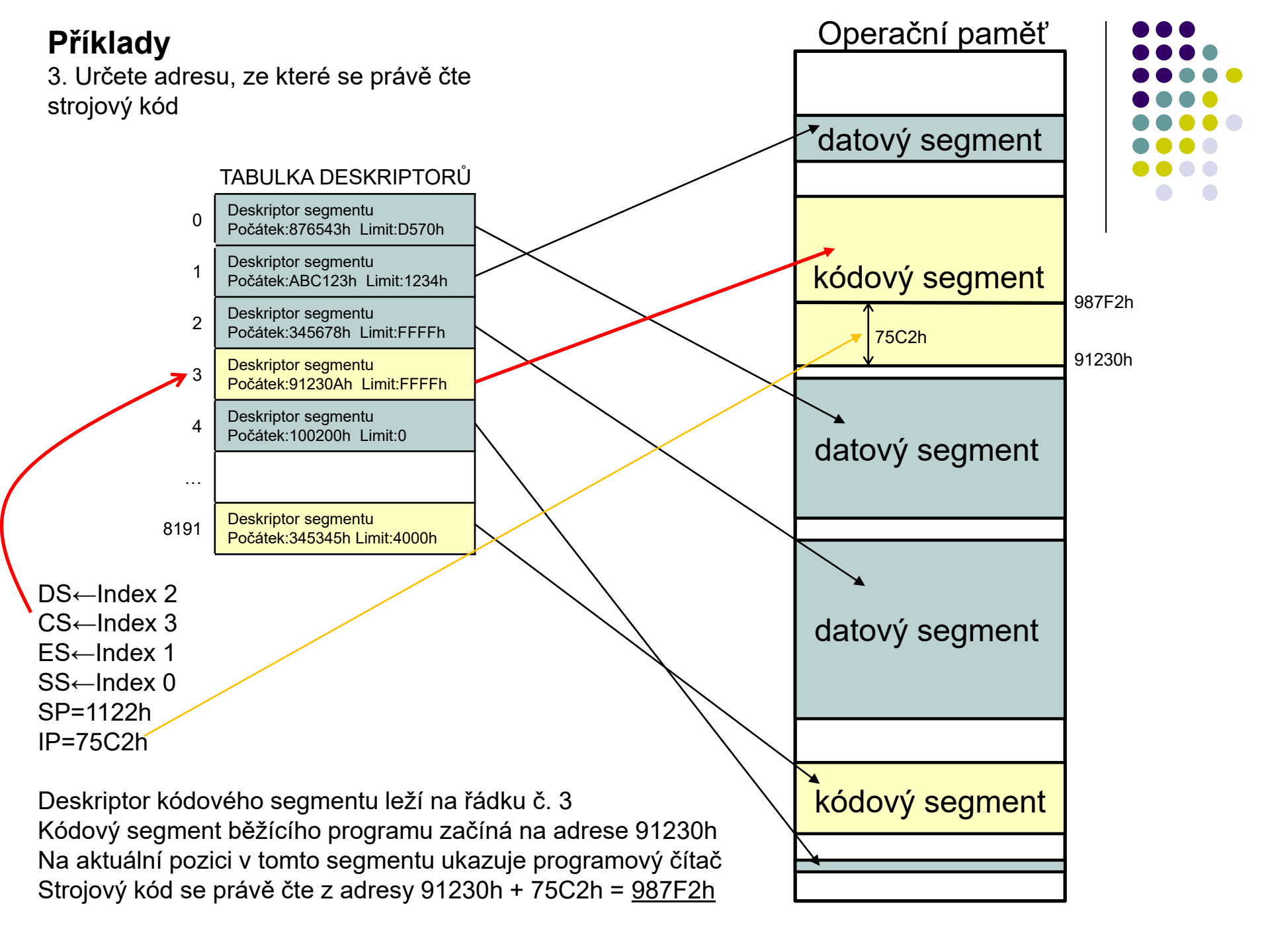


DS←Index 2
CS←Index 3
ES←Index 1
SS←Index 0
SP=1122h
IP=75C2h

Deskriptor pomocného dat. segmentu leží na řádku č. 1
Pomocný datový segment začíná na adrese ABC123h
Pro tento segment platí limit 1234h
Pomocný datový segment končí na adrese
ABC123h + 1234h = ABD357h

Příklady

3. Určete adresu, ze které se právě čte strojový kód

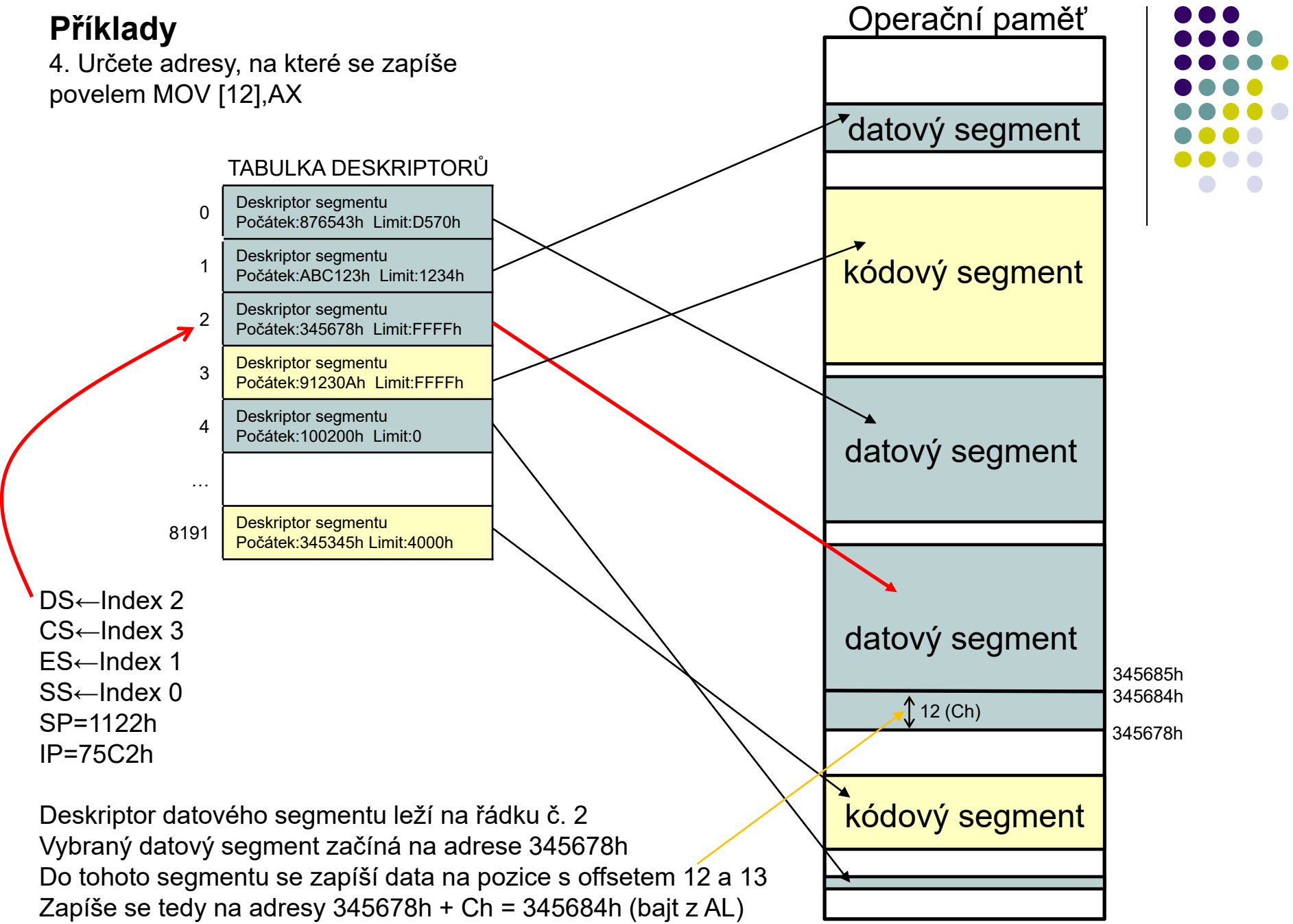


DS←Index 2
CS←Index 3
ES←Index 1
SS←Index 0
SP=1122h
IP=75C2h

Deskriptor kódového segmentu leží na řádku č. 3
Kódový segment běžícího programu začíná na adrese 91230h
Na aktuální pozici v tomto segmentu ukazuje programový čítač
Strojový kód se právě čte z adresy 91230h + 75C2h = 987F2h

Příklady

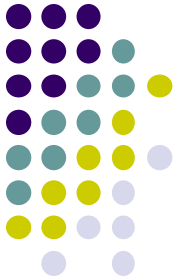
4. Určete adresy, na které se zapíše povelem MOV [12],AX



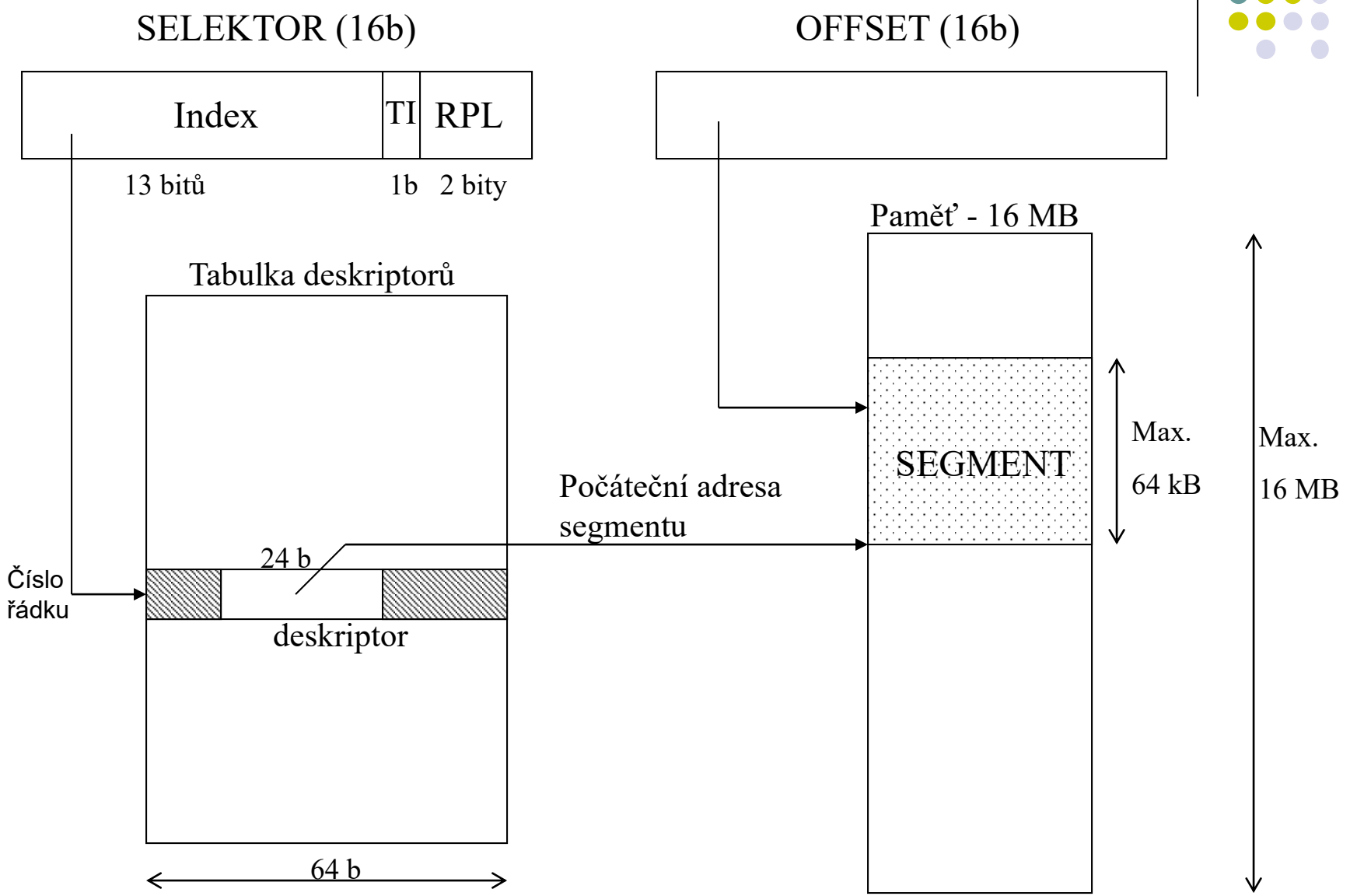
DS←Index 2
CS←Index 3
ES←Index 1
SS←Index 0
SP=1122h
IP=75C2h

Deskriptor datového segmentu leží na řádku č. 2
Vybraný datový segment začíná na adrese 345678h
Do tohoto segmentu se zapíší data na pozice s offsetem 12 a 13
Zapíše se tedy na adresy 345678h + Ch = 345684h (bajt z AL)
a také na 345678h + Dh = 345685h (bajt z AH)

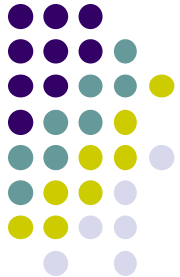
Chráněný režim a paměť



- Procesor v tomto režimu používá zcela jiný postup pro vytváření fyzické adresy než v režimu reálném
- Adresa je vytvářena ze dvou 16bitových složek nazývaných **selektor** a **offset** za pomoci **tabulek deskriptorů**.
- Výsledná adresa je potom **24-bitová**, což umožňuje procesoru adresovat maximálně 2^{24} B = **16 MB** operační paměti
- Význam **offsetu** se nezměnil – ukazuje pozici uvnitř 64 kB velkého segmentu a je 16-bitový
- Význam 16-bitového **selektoru** (CS, DS, ES, SS) je poměrně komplikovaný
  - Nejvyšších 13 bitů selektoru funguje jako **index** - ukazatel do tabulky deskriptorů (vybírá číslo řádku s požadovaným deskriptorem)
  - Jeden bit slouží pro výběr **globální či lokální** tabulky deskriptorů
  - Poslední dva bity pro nastavení jedné ze čtyř možných **úrovní oprávnění** při přístupu k segmentu (RPL)

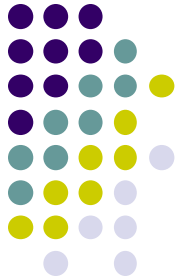

SELEKTOR (16b)
OFFSET (16b)
Index
TI
RPL
13 bitů
1b
2 bity
Paměť - 16 MB
Tabulka deskriptorů
SEGMENT
Max.
64 kB
Max.
16 MB
Počáteční adresa
segmentu
24 b
Číslo
řádku
deskriptor
64 b

Tabulka deskriptorů



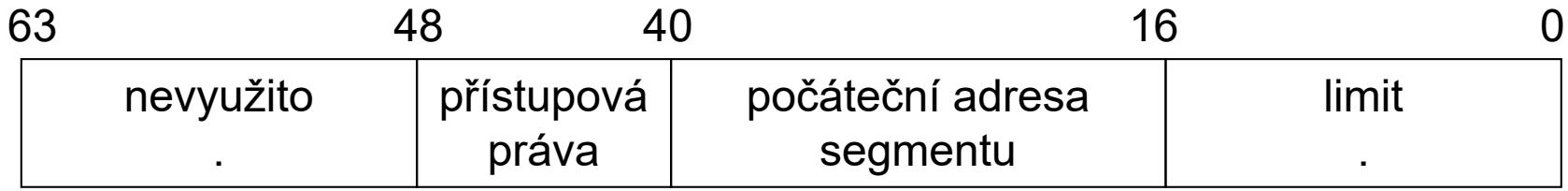
- V paměti jsou **operačním systémem** vytvořeny **tabulky deskriptorů**
- Tabulka deskriptorů obsahuje až 8192 položek – „popisovačů“ používaných paměťových segmentů
- Každý segment je popsán 64-bitovým deskriptorem
- Tedy tabulka deskriptorů může obsahovat až 8192 deskriptorů o šířce 64 bitů, které obsahují informace o 8192 segmentech
- Tabulka deskriptorů pak zabírá v paměti 8192x64 bitů, tj. 8192x8B, to je **64 kB**

Deskriptory a segmenty



- Podle typu segmentu rozlišujeme 4 typy deskriptorů
  - **deskriptor datového segmentu** (segment obsahuje data, lze do něj zapisovat, nelze do něj „skočit“ a spustit kód)
  - **deskriptor kódového segmentu** (segment obsahuje spustitelný strojový kód, nelze do něj zapisovat data, program nemůže přepsat sám sebe nebo být změněn či přepsán jiným programem)
  - **deskriptor systémového segmentu** (obsahuje informace pro OS, tabulky deskriptorů, stav rozpracovaných úloh)
  - **deskriptor brány** (slouží pro volání podprogramů, služeb a přerušení. Nepopisuje segment, ale konkrétní místo v paměti, na které lze skočit při volání nějaké služby)

Obsah deskriptoru



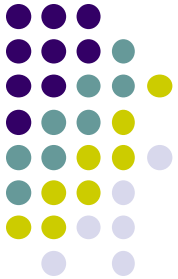
- Jedna **64-bitová** položka tabulky deskriptorů (tedy **deskriptor**) obsahuje tyto informace
  - **Bázovou adresu** segmentu (24 bitů)
  - **Limit** (16 bitů) – neboli velikost segmentu. Může být max. 64 kB, ale může být i libovolná menší. Například segment s velikostí 10 B bude mít limit 9 (poslední bajt leží na pozici s offsetem 9)
  - **Přístupová práva** k segmentu (8 bitů) – Definují, jakou úroveň oprávnění musí mít program, aby mohl k segmentu přistupovat a co je povoleno s obsahem segmentu provádět. Význam těchto bitů se liší dle typu segmentu.
  - Zbývajících 16 bitů je zatím (u 286) nevyužito

Selektor

- V chráněném režimu mají funkci selektoru registry, které jsme v reálném režimu nazývali jako **segmentové** – **CS, DS, ES, SS**

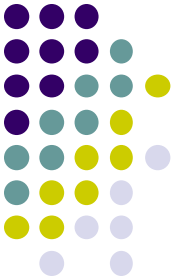
- **Selektor** obsahuje
  - **Index** to tabulky deskriptorů (13 bitů) – vybírá deskriptor na řádku 0…8191
  - **TI** (1 bit) – určuje výběr lokální nebo globální tabulky
  - **RPL** (2 bity) – nastavení požadované úrovně oprávnění (Requested Privilege Level)

Selektor



- Z důvodu urychlení výpočtu fyzických adres je ke každému 16 bitovému selektoru ještě přidána další 64bitová **skrytá část**, která je pro programátora nepřístupná
- Jakmile se selektorem vybere deskriptor nějakého segmentu, "natáhne" se do jeho neviditelné části příslušný deskriptor, na který selektor v dané tabulce ukazoval
- Mikroprocesor pak má k dispozici rychle veškeré informace o segmentech, se kterými se právě pracuje
- Deskriptory právě používaných segmentů jsou zkopírovány uvnitř procesoru ve skrytých částech selektorů, kterými byly vybrány
- Ve skryté části selektoru jsou tedy uloženy informace o segmentu, který byl právě programem vybrán a pokud se selektor nemění, není třeba přistupovat opakovaně pro deskriptor do tabulky deskriptorů
- Výpočet fyzické adresy v chráněném režimu trvá déle než v reálném a je za něj odpovědná jednotka AU

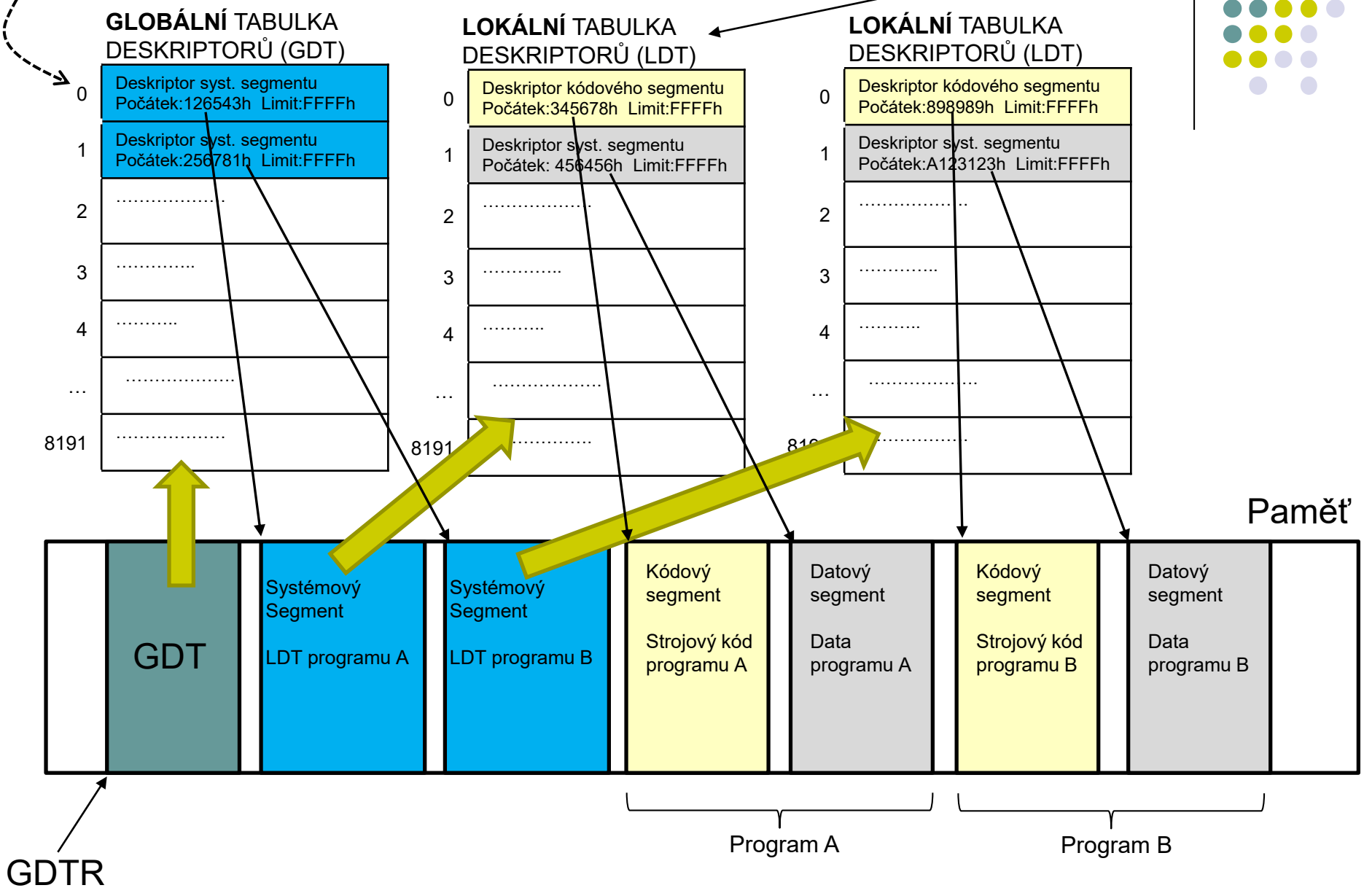
GDTR a LDTR



- Tabulek deskriptorů může být více
- Tabulky deskriptorů jsou uloženy na libovolných místech v paměti
- Každá tabulka deskriptorů je **systémovým segmentem**
- Jedna jediná tabulka je **globální**
- Počáteční 24-bitová fyzická adresa této globální tabulky je uložena v registru **GDTR**
- V globální tabulce se kromě deskriptorů "normálních" datových či kódových segmentů nachází i deskriptory popisující segmenty s lokálními tabulkami deskriptorů
- **Lokální tabulka** deskriptorů popisuje segmenty používané jednou úlohou (její lokální paměťový prostor)
- Lokální tabulka může obsahovat až 8192 položek o šířce 64 bitů (8B), takže její velikost je až 64kB a leží vlastně ve speciálním systémovém segmentu
- Lokálních tabulek může být více, typicky právě pro každý proces jedna
- Registr **LDTR** funguje jako selektor, který z globální tabulky vybírá příslušný deskriptor segmentu, ve kterém leží lokání tabulka
- Počáteční adresa lokální tabulky deskriptorů je tedy uložena v deskriptoru systémového segmentu (tato tabulka je vlastně systémovým segmentem a ten musí mít svůj deskriptor a v něm informaci o své počáteční adrese)

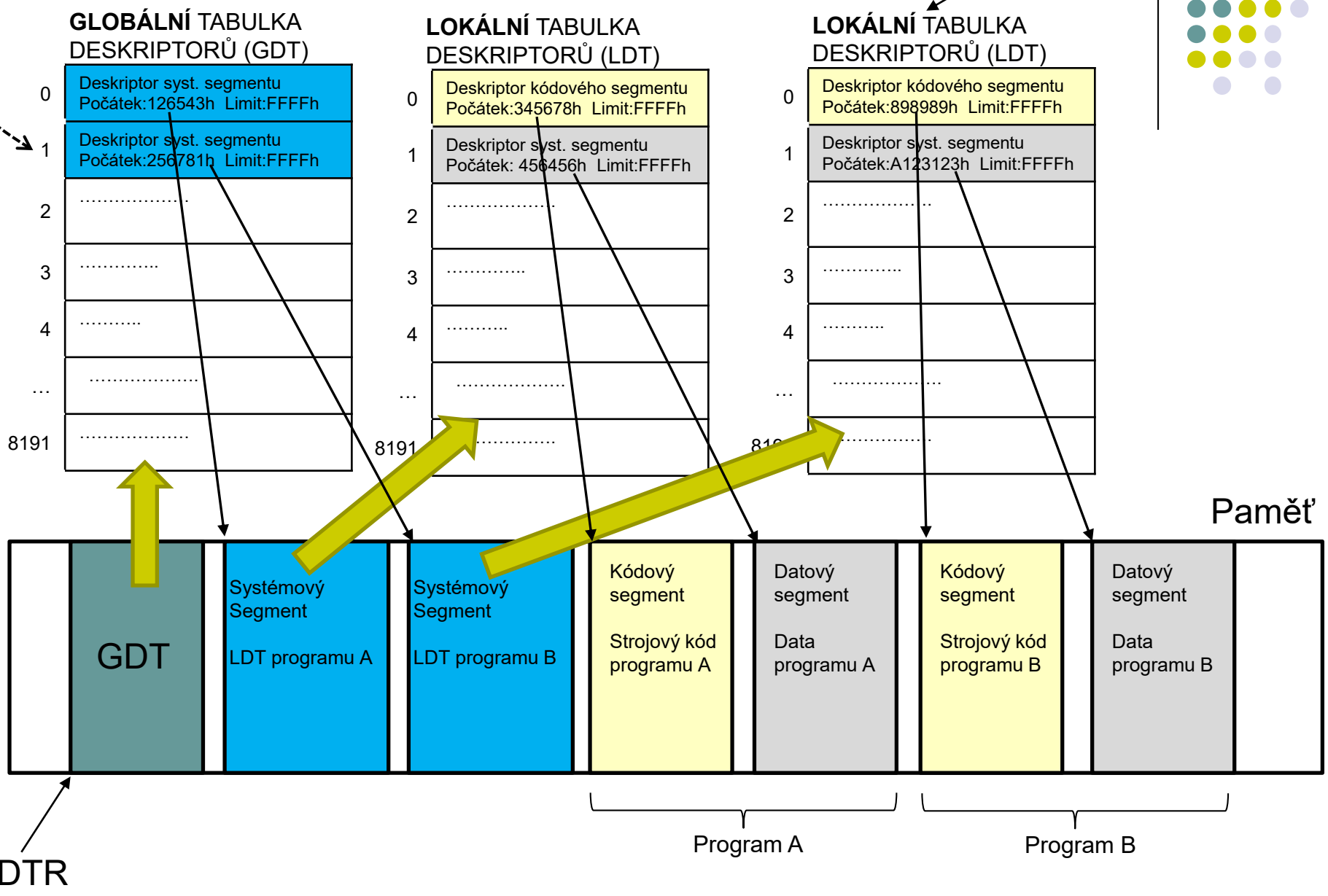
GDT a LDT

LDTR←index 0
Běží program **A**. Používá se tato lokální tabulka



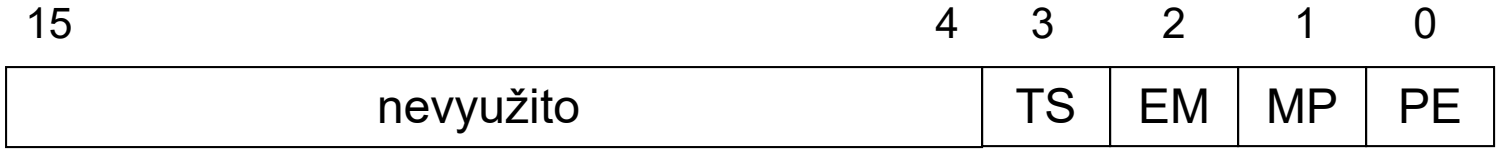
GDT a LDT

LDTR←index 1
Běží program **B**. Používá se tato lokální tabulka



Speciální registr MSW

- MSW - Machine Status Word



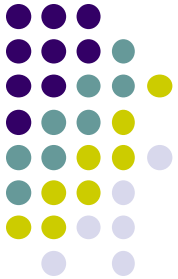
PE - Protected Mode Enable - zapíná chráněný režim procesoru. Po inicializaci procesoru (signálem RESET) je zapnut reálný režim. Nastavením tohoto príznaku se procesor přepne do chráněného režimu. Zpět do reálného režimu lze procesor vrátit pouze inicializací procesoru (RESET)

MP - Monitor Processor Extension - indikuje fyzickou přítomnost koprocesoru

EM - Emulate Procesor Extension - zapíná programovou emulaci korprocesoru tehdy, není-li koprocesor instalován (je-li EM=1, způsobí instrukce koprocesoru přerušení INT 7)

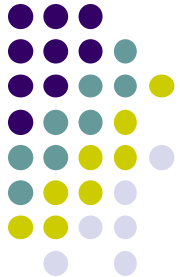
TS - Task Switch - se nastavuje vždy přepnutím procesu. Je používán koprocesorem ke zjištění, že v procesoru se vyměnila zpracovávaná úloha.

Virtuální paměť



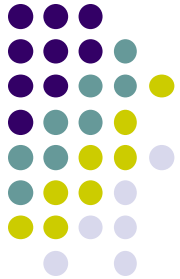
- Ve víceúlohovém prostředí snadno dochází k situaci, kdy fyzická paměť (max. 16 MB) nestačí součtu paměťových požadavků jednotlivých úloh
- Řešením je ***předstírat***, že je k dispozici podstatně větší paměťový prostor než ve skutečnosti
- Každá úloha může používat svých **8192 segmentů** (max. počet položek v její **lokální tabulce deskriptorů** segmentů)
- Při maximální velikosti segmentu 64 kB je možné, aby každá úloha operovala se svými 8192 x 64 kB paměti, což je 512 MB
- Dalších 512 MB paměti je k dispozici všem úlohám společně v segmentech **globálního prostoru**
- Jedna úloha tedy může mít k dispozici **až 1 GB** paměti (zatím)
- Spuštěno může být několik takových úloh náraz (multitasking)
- V případě, kdy suma paměťových požadavků překročí **16 MB** je třeba použit mechanismu **virtuální paměti**
- **Segmenty, které se nevejdou do paměti, se odkládají na disk**
- V deskriptoru každého segmentu je jeden bit rezervován pro informaci o tom, zda je segment skutečně fyzicky přítomen v paměti nebo zda byl odložen na disk
- Přístup do segmentu, který v RAM není, má za následek **přerušení**, které OS obslouží nahráním příslušného segmentu z disku
- OS má na starost organizaci dat v odkládacím souboru na disku

Virtuální paměť



- Pokud máte malou operační paměť a spustíte současně mnoho programů, které používají velké množství dat, vznikne velké množství segmentů, které se nevejdou do paměti a budou uloženy v odkládacím souboru na disku
- V paměti budou uloženy vždy pouze segmenty, se kterými se často pracuje
- V multitaskingu ale dochází k rychlému střídání programů a segmenty, které často používal jeden program, druhý program vůbec nepoužívá
- Takže při přepnutí úloh (což se děje třeba každou milisekundu) je najednou paměť plná segmentů, se kterými aktuálně aktivovaná úloha pracovat vůbec nechce a naopak veškeré segmenty s daty, který potřebuje, jsou odložené na disku
- Než tedy vůbec může aktivovaná úloha začít pracovat, je třeba načíst její segmenty z disku do paměti
- Paměť je ale plná, takže nejprve je potřeba uvolnit místo, aby se do ní mohly načíst segmenty, které teď bude aktivní úloha používat
- Místo se uvolní jedině tak, že jiné segmenty, které už v paměti jsou, se odloží na disk
- Tím, jak se úlohy neustále střídají, dochází pak k neustále výměně segmentů mezi operační pamětí a odkládacím souborem na disku
- Čím více programů naráz spustíte a čím menší máte skutečnou operační paměť, tím více se bude počítač trápit neustálým čtením a zapisováním segmentů na disk – aplikaci pak prakticky neběží, neustále se čeká na přesuny segmentů

Deskriptor datového segmentu

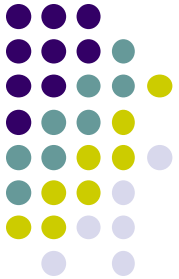


Obsah bajtu přístupová práva (bit 40 až 47 deskriptoru)

P	DPL	1	0	ED	W	A
7	6 5	4	3	2	1	0

- **P** (Segment Present) je nastaven na jedničku tehdy, je li obsah segmentu uložen v reálné paměti. Není-li, je nulový.
- **DPL** (Descriptor Privilege Level) 2-bitové zakódované číslo 0 až 3, určuje úroveň oprávnění přidělenou segmentu. Program, který chce s tímto segmentem pracovat, musí mít úroveň oprávnení stejnou nebo lepší
- **W** (Writable) je nastaven na 1, pokud je povoleno čtení i zápis do segmentu. Zásobník musím mít vždy W=1. Je-li bit nulový, pak je segment ReadOnly
- **A** (Accesssed) při každém přístupu k segmentu se automatický nastaví na 1. Procesor tento příznak nenuluje. Je určen operačnímu systému ke sledování četnosti přístupů ke konkrétním segmentům. Segmenty, které se nepoužívají, budou při každé kontrole deskriptorů mít tento bit nulový. Takové segmenty se pak budou jako první odkládat na disk při vyčerpání kapacity paměti a realizaci virtuální paměti.
- **ED** (Expansion Direction) indikuje, kterým směrem se bude obsah segmentu rozšiřovat (zásobník roste opačným směrem než běžná data)

Deskriptor kódového segmentu

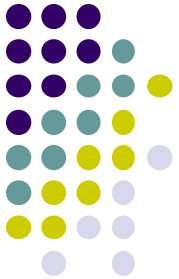


Obsah bajtu přístupová práva (bit 40 až 47 deskriptoru)

P	DPL		1	1	C	R	A
7	6	5	4	3	2	1	0

- **P** (Segment Present) je nastaven na jedničku tehdy, je li obsah segmentu uložen v reálné paměti. Není-li, je nulový.
- **DPL** (Descriptor Privilege Level) určuje úroveň oprávnění potřebnou pro přístup k 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.
- Zápis do kódového segmentu není možný

Deskriptor systémového segmentu

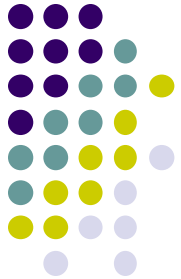


Obsah bajtu přístupová práva (bit 40 až 47 deskriptoru)

P	DPL		0	0	0	Typ	
7	6	5	4	3	2	1	0

- **Typ=1** označuje segment stavu procesu (TSS - Task State Segment) pro právě neaktivní proces. V tomto segmentu má OS uloženy informace o rozpracované úloze
- **Typ=3** označuje segment stavu procesu pro právě aktivní proces
- **Typ=2** označuje segment lokální tabulky deskriptorů segmentů (LDT - Local Descriptor Table)

Úrovně oprávnění

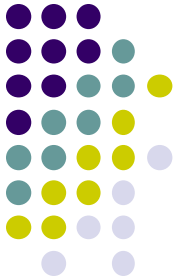


- K dispozici jsou **4 úrovně oprávnění**, které jsou číselně označeny 0 (nejvyšší) až 3 (nejnižší)

- **úroveň 0 - jádro operačního systému** (řízení procesoru, I/O operací, přidělování paměti, vytváření a ukončování úloh)
- **úroveň 1 -** služby poskytované operačním systémem
- **úroveň 2 -** systémové programy a podprogramy z knihoven (systém obsluhy souborů, správa knihoven)
- **úroveň 3 -** běžné uživatelské aplikace

- Vždy je kontrolováno, zda aplikace nemá snahu manipulovat se segmentem, který používá proces s vyšším oprávněním
- Běžné uživatelské aplikace (úroveň 3) nemají právo měnit data operačního systému. Naopak jádro operačního systému smí vše – zavádět nové aplikace do paměti, ukončovat úlohy, odebírat paměť atd.
- V případě nepovoleného přístupu do paměti je generováno přerušení, které vhodným způsobem obslouží operační systém
  - např. Windows generují *„Memory Access Violation“* a neposlušnou úlohu ukončí)
  - UNIXová chybová hláška *Segmentation fault*

IOPL

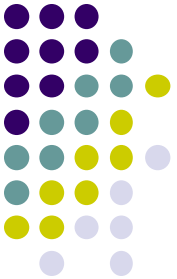


- V registru FLAGS přibyly nové dva příznakové bity, kterými operační systém nastaví úroveň opravnění potřebnou pro povolení vstupu/výstupu

- **IOPL** = Input Output Privilege Level

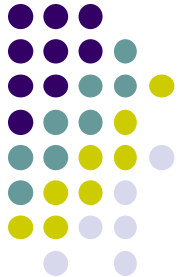
- Pomocí IOPL lze nastavit nejnižší možnou úroveň oprávnění, při které ještě úloha může provádět vstupně-výstupní operace (tzn. pomocí instrukci IN a OUT přímo přistupovat k periferiím)
- Úlohy s nižší úrovní oprávnění, než která je nastavená pomocí IOPL, nemohou přímo přistupovat k hardwaru (musí za tím účelem volat např. služby OS)
- Pokud se úloha s nedostatečným oprávněním pokusí provést operaci IN nebo OUT je generováno přerušení, které je obslouženo operačním systémem

IOPL



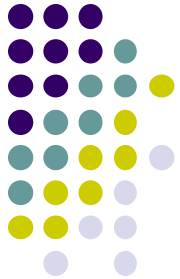
- Pokud **IOPL=3**
- Pak k provádění instrukcí IN a OUT stačí mít úroveň oprávnění 3, což je nejnižší možná úroveň oprávnění.
- IN a OUT mohou provádět procesy s úrovní oprávnění 0, 1, 2 ,3
- V tomto případě mohou tedy přistupovat ke vstupu a výstupu všechny běžící procesy a tím pádem operační systém ztrácí kontrolu nad vstupem a výstupem (programy mohou chaoticky zapisovat kamkoliv na disk, mazat jakýkoliv soubor, obcházet přístupová práva k souborům, dva programy současně posílají nesmyslná data tiskárně, odesílají se pakety do sítě a OS neví kam a nemůže je kontrolovat…)

IOPL



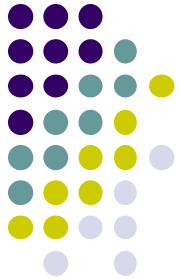
- Pokud **IOPL=2**
- Pak k provádění instrukcí IN a OUT je třeba mít alespoň úroveň oprávnění 2
- IN a OUT mohou provádět procesy s úrovní oprávnění 0, 1, 2
- Běžné uživatelské aplikace s úrovní oprávnění 3 nemohou přistupovat ke vstupu a výstupu
- Pokud chce takový program zapsat něco na disk, nemá k němu přímý přístup a musí využít službu operačního systému (vytvoř soubor, čti soubor, přejmenuj soubor, zapiš do souboru)
- Program sám s diskem neumí komunikovat – k tomu je potřeba instrukce IN a OUT, která je ale pro program zakázaná
- Program tedy nemůže zapsat jakákoliv dat na jakékoliv místo na disku
- Program nemá přístup k tiskárně – musí využívat službu pro tisk, kterou mu nabízí OS. Operační systém pak může zabránit tomu, aby tiskly dva programy současně tím, že vytváří tiskovou frontu.
- Program nemá přístup k síti. K navazování spojení, odesílání a přijímání paketů musí používat služby operačního systému a jeho porty. Operační systém pak může kontrolovat, který program kam co posílá (firewall, antivir) a přicházející pakety dle čísla portu předávat cílovým aplikacím
- Program nemá přístup ani ke klávesnici, myši, USB portům, časovači…. – vše je pro něj dostupné pouze voláním služeb operačního systému a přímo s těmito zařízeními nemůže komunikovat
- Operační systém má tedy naprostou kontrolu nad tím, kam koncové aplikace přistupují

IOPL



- Pokud **IOPL=0**
- Pak k provádění instrukcí IN a OUT je nutné mít úroveň oprávnění 0
- Úroveň oprávnění 0 má pouze jádro operačního systému
- IN a OUT může provádět pouze proces s úrovní oprávnění 0 – tedy jádro OS
- V tomto případě by nemohly přistupovat ke vstupu a výstupu ani ostatní procesy operačního systému, služby, ovladače atd.
- Vstup a výstup by byl v podstatě zakázaný

Skok do jiného kódového segmentu

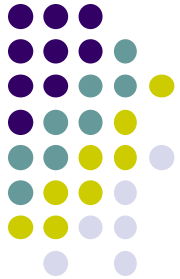


- Program **uvnitř svého kódového segmentu** může skákat z místa na místo bez omezení
- **Mezisegmentovým skokem** nebo voláním podprogramu nazýváme přesun programového čítače na adresu ležící v **jiném kódovém segmentu**

- Momentální úroveň oprávnění právě prováděného procesu **musí být stejná nebo lepší** než úroveň oprávnění, kterou je označen cílový kódový segment
- K mezisegmentovému skoku dojde například pokud program volá nějakou službu operačního systému

- Vzniká zde tedy problém v případě, že aplikace by chtěla využít některé systémové služby (služby OS)

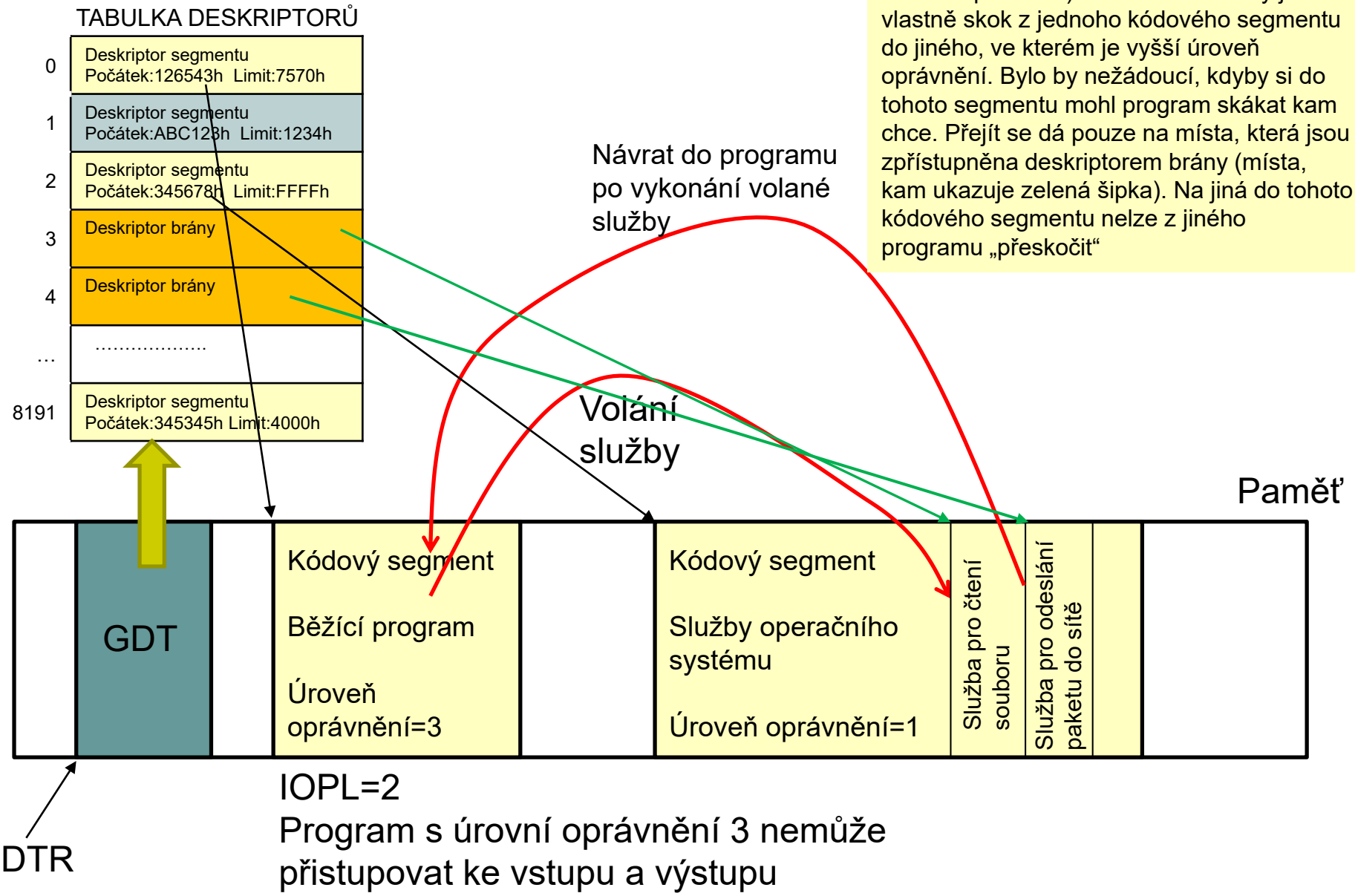
Brána pro předání řízení



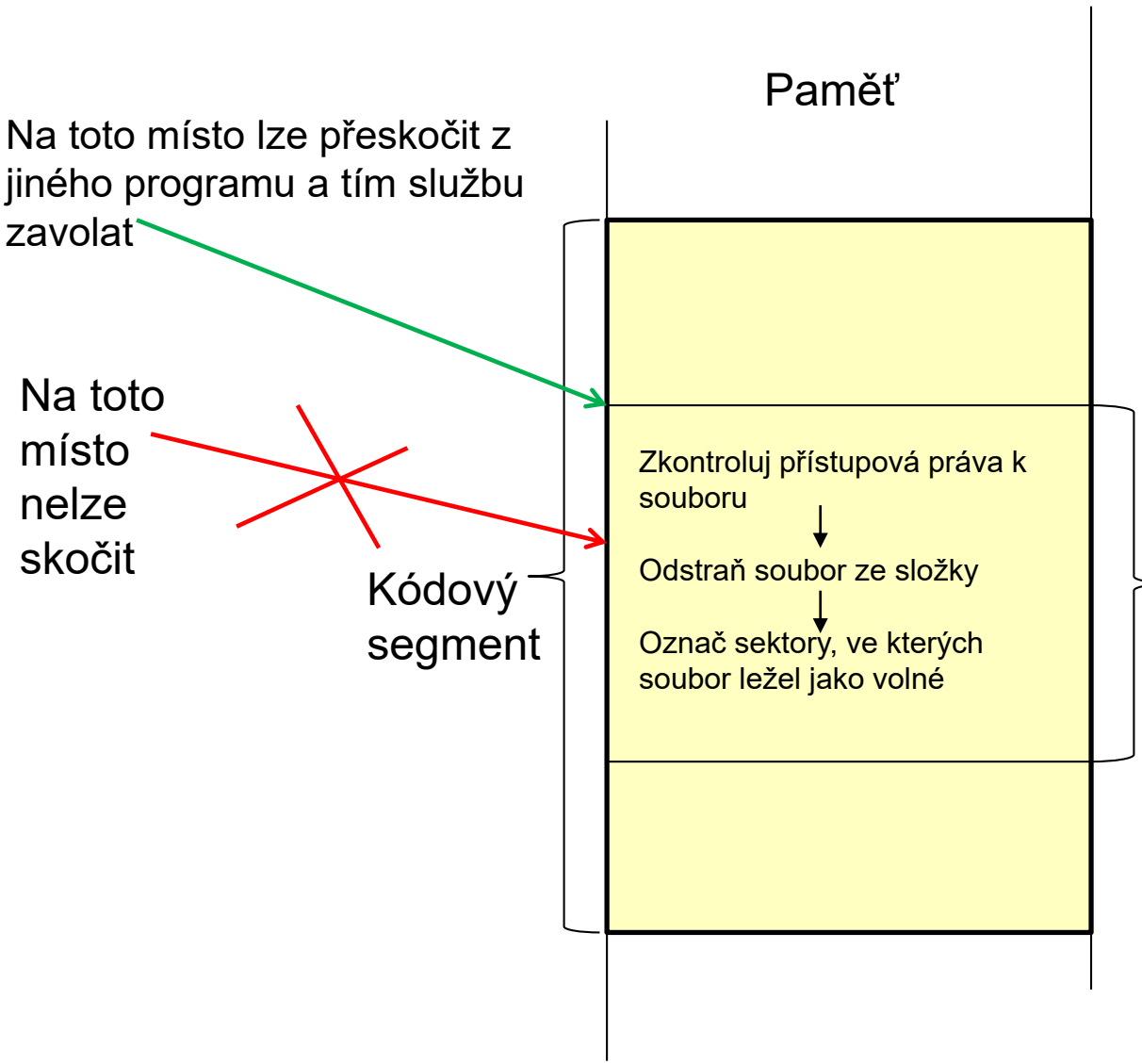
- Brána se používá v situaci, kdy chce proces na **nízké úrovni** oprávnění volat **podprogram úrovně vyšší** (např. uživatelská aplikace chce volat službu OS pro zápis na disk)
- Není možné skočit přímo do kódového segmentu OS, kde leží kód volané služby
- Nebylo by žádoucí, aby měl běžný program možnost skočit na libovolnou instrukci vyšší úrovně oprávnění
- Bránu si potom je možno představit opravdu jako vchod do jiného světa - do kódového segmentu operačního systému nevidím a nemohu z něj nic spouštět, kromě zavolání jednoho konkrétního vstupního bodu podprogramu (určeného bránou)
- Brána je realizována jako zvláštní typ deskriptoru

Brána pro předání řízení

Běžící program potřebuje číst soubor z disku. Sám s diskem nemůže komunikovat. Pro čtení souboru musí zavolat službu operačního systému (ta má dostatečnou úroveň oprávnění). Volání této služby je vlastně skok z jednoho kódového segmentu do jiného, ve kterém je vyšší úroveň oprávnění. Bylo by nežádoucí, kdyby si do tohoto segmentu mohl program skákat kam chce. Přejít se dá pouze na místa, která jsou zpřístupněna deskriptorem brány (místa, kam ukazuje zelená šipka). Na jiná do tohoto kódového segmentu nelze z jiného programu „přeskočit“

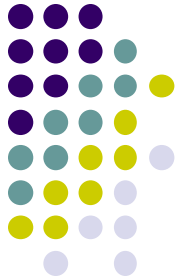


Brána pro předání řízení



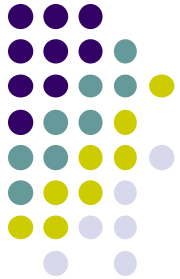
Skočit lze pouze na místa, pro která je vytvořen deskriptor brány. Tím je zajištěno, aby služba „Smaž souboru“ byla vždy zavolána od začátku a tím pádem nejde přeskočit kontrola přístupových práv k souboru, kterou služba před vlastním smazáním souboru provádí

Privilegované instrukce



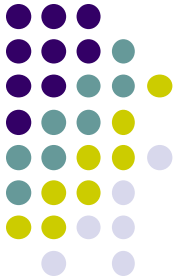
- Některé speciální řídící instrukce smí provádět pouze proces s určitým stupněm úrovně oprávnění (jádro operačního systému)
- Použití privilegovaných instrukcí je obvykle omezeno pouze na jádro OS
- Základní privilegované instrukce:
  - **LGDT** naplnění registru GDTR
  - **LIDT** naplnění registru IDTR
  - **LLDT** naplnění registru LDTR
  - **LTR** naplnění registru TR
  - **LMSW** naplnění registru MSW
  - **CLTS** nulování bitu TS v registru MSW
  - **HLT** zastavení procesoru
- Při pokusu provést privilegovanou instrukci bez potřebné úrovně oprávnění je generováno přerušení, které OS obvykle obslouží ukončením neposlušného procesu

Záměna úloh



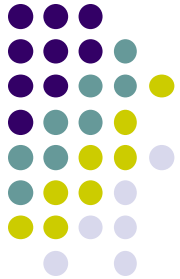
- Nejdůležitější charakteristikou chráněného režimu je možnost provádění několika úloh takovým způsobem, že uživatel má dojem, že se úlohy provádějí paralelně
- Tato simulace je prováděna metodou rychlého přepínání mezi jednotlivými úlohami
- Pro přepínání úloh mají zásadní význam registr **TR** a segment stavu úlohy **TSS**
- **TR –** Task register – selektor, který z globální tabulky vybírá deskriptor Task state segmentu právě běžící úlohy

Task State Segment



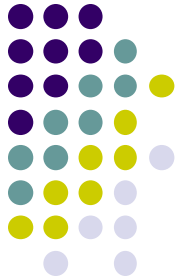
- Slouží k uchování informací o rozpracovaném procesu
- Každá úloha má svůj TSS
- Task State Segment má velikost minimálně 42 B a obsahuje hodnotu všech registrů v okamžiku přerušení procesu, LDTR, příznakový registr a ukazatel na vrchol zásobníku
- Operační systém si do tohoto segmentu může ukládat další důležité informace o úloze (čas spuštění, priorita, jaký uživatel úlohu spustil atd…)
- Informace o segmentu TSS obsahuje deskriptor systémového segmentu (smí být umístěn pouze v GDT).
- TR funguje jako selektor ukazující do GDT na TSS aktivní úlohy

Přerušení v chráněném režimu



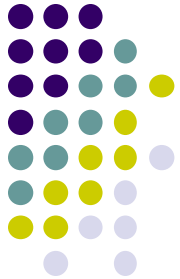
- V chráněném režimu již na počátku paměti (prvních 1024 B) nejsou uloženy vektory přerušení
- Místo vektorů se používá tabulka IDT
- Tabulka IDT (Interrupt Deskriptor Table) obsahuje 256 deskriptorů obslužných rutin přerušení
- Při vzniku přerušení se použije číslo přerušení jako index do tabulky IDT
- Položkami Tabulky IDT jsou deskriptory s informací o tom, kde leží v paměti pro dané přerušení jeho obsluha
- Tedy například na patnáctém řádku této tabulky nalezneme deskriptor, ve kterém je informace o procesu nebo službě, která se má spustit jako obsluha přerušení číslo 15
- Tabulka IDT může ležet kdekoliv v paměti, počáteční adresa této tabulky je uložena v registru IDTR

Kontrolní otázky



- Kolik bitů má fyzická adresa v reálném režimu a kolik v chráněném režimu ?
- Kolikabitový je offset v reálném režimu ?
- Kolikabitový je offset v chráněném režimu ?
- Jaká je maximální možná velikost segmentu v chráněném režimu ?
- Jaká je maximální velikost adresovatelné paměti chráněném režimu ?
- Jaká je maximální možná velikost segmentu v reálném režimu ?
- Jaká je šířka selektoru v bitech ?
- Jaká je šířka indexu v bitech ?
- Jaká je šířka deskriptoru v bitech ?
- Kde je při běhu v chráněném režimu uložena počáteční adresa segmentu ?
- Co je to multitasking a jak funguje ?
- Kolik položek obsahuje globální tabulka deskriptorů ?
- Kolik místa v paměti zabírá globální tabulka deskriptorů ?
- Jaký význam má skrytá část selektoru ?
- Kdy dochází ke čtení deskritprou z tabulky deskriptorů a při kterých přístupech do paměti se naopak deskriptor příslušeného potřebného segmentu z tabulky deskriptorů číst nemusí ?
- Na kolikátém řádku tabulky deskriptorů leží deskriptor datového segmentu, je-li DS=00FDh ?
- Kolik bitů deskriptoru je použito k uložení informace o bázové adrese segmentu ?
- Na konkrétní pozici uvnitř segmentu odkazuje selektor, offset, deskriptor nebo index ?
- Vysvětlete význam bitu A v deskriptoru datového segmentu
- Jakou nejvyšší fyzickou adresu lze přečíst v reálném režimu a jakou v chráněném režimu (zapište ji hexadecimálně) ?

Kontrolní otázky



- K čemu slouží registr GDTR ?
- LDTR je deskriptor, segment, selektor nebo offset ?
- LDT je deskriptor, segment, selektor nebo offset ?
- IP je deskriptor, segment, selektor nebo offset ?
- Kde je uložena bázová adresa aktuální lokální tabulky deskriptorů ?
- Kde je uložena bázová adresa globální tabulky deskriptorů ?
- Jak se pozná, zda index vybírá deskriptor z lokální nebo globální tabulky ?
- Index je uložen v deskriptoru, selektoru, segmentu, GDTR nebo datovém registru ?
- Co se stane při pokusu o čtení dat ze segmentu, v jehož deskriptoru je bit P=0 ?
- Co se stane při pokusu o zápis dat do segmentu, v jehož deskriptoru je bit W=0 ?
- TR je deskritptor, segment, selektor nebo offset ?
- Kdy se mění stav TR ?
- TSS je deskriptor, selektor, segment nebo offset ?
- Kolik úrovní oprávnění podporuje chráněný režim a jakou číselnou úroveň oprávnění má jádro operačního systému ?
- Co jsou to privilegované instrukce ?
- Jak lze přejít z chráněného režimu zpět do reálného ?
- Co je to IOPL ?
- Co je to systémový segment ? Co může obsahovat ?
- K čemu slouží brána pro předání řízení ?
- Kolik deskriptorů obsahuje tabulka IDT ?
- Jaká informace je uložena v deskriptorech v tabulce IDT ?