Programování v jazyce C

16 Programátorský model CPU x86

- Vlastnosti CPU
- Instrukční soubor
- Datové typy
- Techniky adresování paměti
- Základy programování v assembleru
- Souvislost s vyššími programovacími jazyky

Programátorský model CPU
Co musí programátor vědět o procesoru?

Programátorský model = soubor vlastností procesoru, které ovlivňují jeho programování v nízkoúrovňových jazycích (a které by měl programátor znát, aby psal „rozumný” kód).

Do programátorského modelu se zahrnuje zejména:

(i) instrukční soubor procesoru,
(ii) počet, uspořádání a pojmenování registrů,
(iii) nativní datové typy daného procesoru,
(iv) techniky a rozsahy adresování paměti,
(v) konstrukce a používání zásobníku,
(vi) přerušovací systém,
(vii) ovládání externích zařízení (interakce CPU s okolím),
a další vlastnosti...

Intel 80386 (i386)
Etalon 32-bitového procesoru

- první 32-bitový procesor společnosti Intel, uveden na trh v roce 1985 — současné nejmodernější CPU Intel stále využívají zavedené koncepce (např. instrukční sadu IA-32, techniky adresování paměti, pojmenování registrů, atd.)
  ⇒ **proto má smysl ho studovat**

- dokáže obhospodařovat 4 GB ***fyzické*** a 64 TB ***virtuální*** paměti (jednotka správy paměti integrovaná na čipu)
- 3 režimy činnosti: ***reálný*** (*Real*), ***chráněný*** (*Protected*) a ***virtuální 8086***
- CPU tvoří 6 paralelně pracujících jednotek (Bus Interface Unit, Code Prefetch Unit, Instruction Decoding Unit, Execution Unit, Segmentation Unit, Paging Unit)
- do fyzické paměti se data ukládají po 4KB stránkách

Typy dat
a deklarace proměnných těchto typů

- 8-bitový **Byte** (slabika)
- 16-bitový **Word** (slovo)
- 32-bitový **Double Word** (dvojslovo)
- 64-bitový **Quad Word** (čtyřslovo) — od 80586 dále

– všechny typy mohou být znaménkové/neznaménkové (*Signed*/*Unsigned*) podle instrukce použité pro výpočet (např. **SHL**/**SHR** vs **SAL**/**SAR**)

Toto jsou tzv. ***nativní*** datové typy. Jiné typy dat CPU nezná, protože pouze tyto lze přímo ukládat do registrů (aritmetiku reálných čísel se řeší FPU, má vlastní datové typy).

```
_DATA SEGMENT
  char    DB 'A'
  var1    DW ?
  handle  DD 0
  bignum  DQ 0

  capt    DB 'Retezec', 0
_DATA ENDS
```

Define Byte

Define Word

... DoubleWord

... QuadWord

Typy dat
a deklarace proměnných těchto typů

- 32-bitový procesor akceptuje jako operandy instrukcí maximálně 32-bitová slova (registry jsou 32-bitové).
- Je-li potřeba pracovat s 64-bitovými slovy (QW), používají se k tomu tzv. ***registrové páry***:
  EAX+EDX, EBX+ECX (obvykle jsou spárovány takto)

- v assembleru lze deklarovat i pole pomocí direktivy **DUP** (***duplicate***)

```
unsigned long int pointers[20] = {0};
```

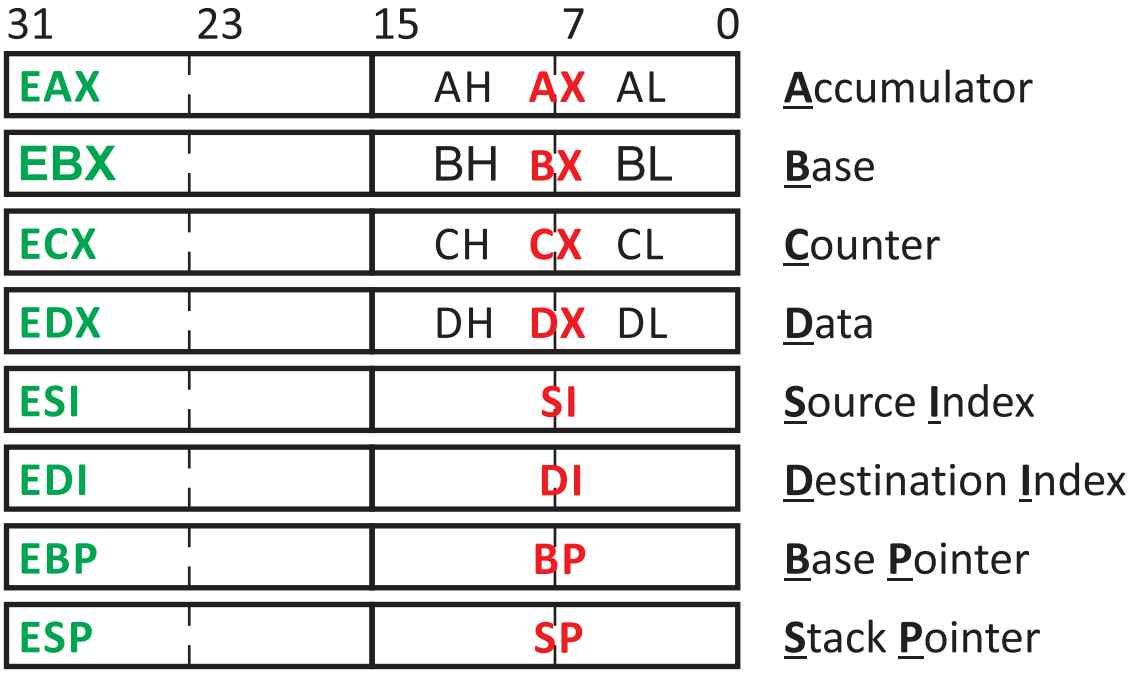
```
_DATA SEGMENT
  pointers DD 20 DUP(0)
  colour   DB 100 DUP(?)
  prompt   DB 13, 10, 'Zadej něco:', 0
_DATA ENDS
```

Registry procesoru
„Místo” přímo v CPU pro dočasné uložení dat

- malé úložiště dat umístěné přímo v procesoru (tj. odpadá zdlouhavá komunikace s vnější operační pamětí)
- extrémně rychlý přístup, dostupné pomocí strojových instrukcí – instrukce si v registrech „přebírají” argumenty a ukládají do nich své výsledky
- velikost registru je obvykle stejná jako velikost ***slova procesoru*** nebo jeho násobku (tj. 32-bitový procesor má obvykle 32-bitové registry)
- jednoduché procesory (mikrokontrolery, monolitické µ-počítače) mohou mít i jen jeden registr, tzv. **akumulátor ALU**; běžně jednotky až desítky registrů (CISC méně, RISC více); speciální procesory (DSP, GPU) mají stovky až tisíce registrů
- data se do registru ukládají pomocí instrukce **`MOV`** (Move), např.: **`MOV R0, 100`** = naplň registr R0 hodnotou 100

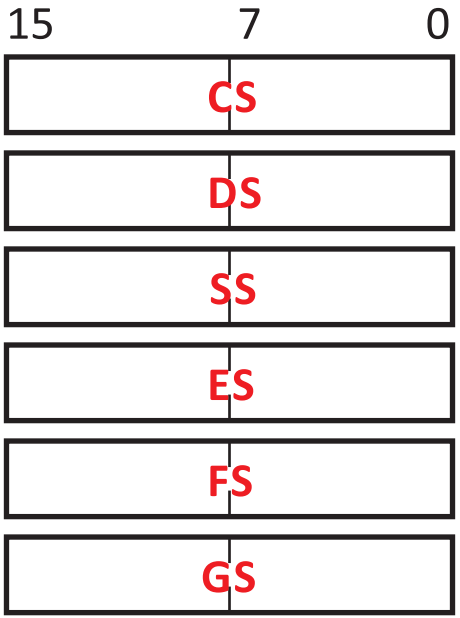
Registry procesorů Intel 80x86

Registry pro všeobecné použití



- Tyto registry může programátor (většinou) bez omezení využívat pro předávání/dočasné ukládání dat, **až na ESP**.

Registry procesorů Intel 80x86
Segmentové registry (od 80386 dále)



Code **S**egment

Data **S**egment

Stack **S**egment

Extra **S**egment

Extra Segment

Extra Segment

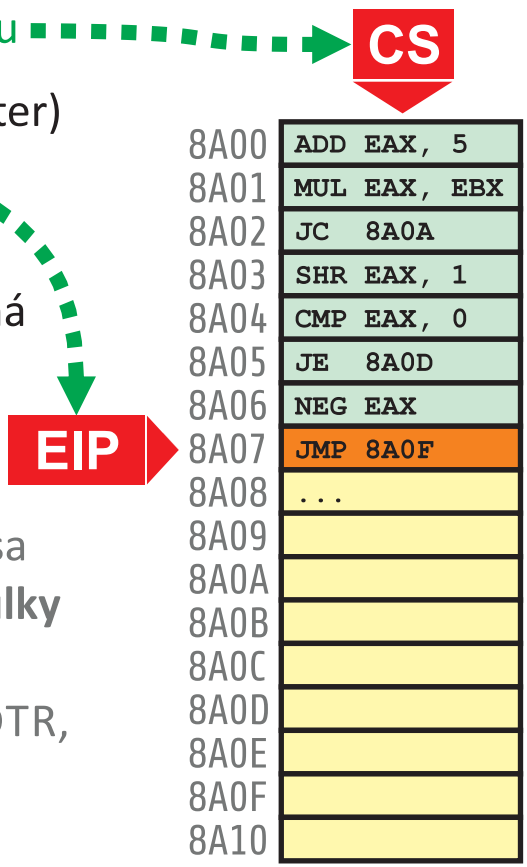
- viditelná část segmentových registrů je 16-bitová, plnit je lze instrukcí **`MOV`** nebo instrukcemi **`LDS`**, **`LES`**, **`LFS`**, **`LGS`**, **`LSS`**
- **změna obsahu CS má fatální následky** (obsahuje tzv. ***selektor kódového segmentu***)

Registry procesorů Intel 80x86
Registry se zvláštním významem

selektor kodového segmentu → CS

CS:EIP (**E**xtended **I**nstruction **P**ointer)

EIP ukazuje na pozici v kódovém segmentu, kde leží právě prováděná instrukce, tj. **pár CS:EIP udává pozici právě vykonávané instrukce**, hodnotu EIP zvyšuje prováděcí jednotka CPU (není to přímo adresa v paměti, jedná se o **index do tabulky GDT/LDT** (Global/Local Descriptor Table), pozice GDT je v registru GDTR, LDT v registru LDTR – viz dále **Segmentace paměti**)



Toto je pouze ilustrativní obrázek - instrukce ve skutečnosti nezabírají stejné množství paměti...

Registry procesorů Intel 80x86
Registry se zvláštním významem

CS:EIP - pozice vykonávané instrukce (mění ji sám procesor)

SS:ESP - pozice vrcholu zásobníku (mění ji instrukce **PUSH** a **POP**)

} pro programátora (zvláště nezkušeného) **read-only**!

DS:ESI - zdrojová adresa pro instrukce blokového přesunu dat (**MOVS/MOVSB/MOVSW**)

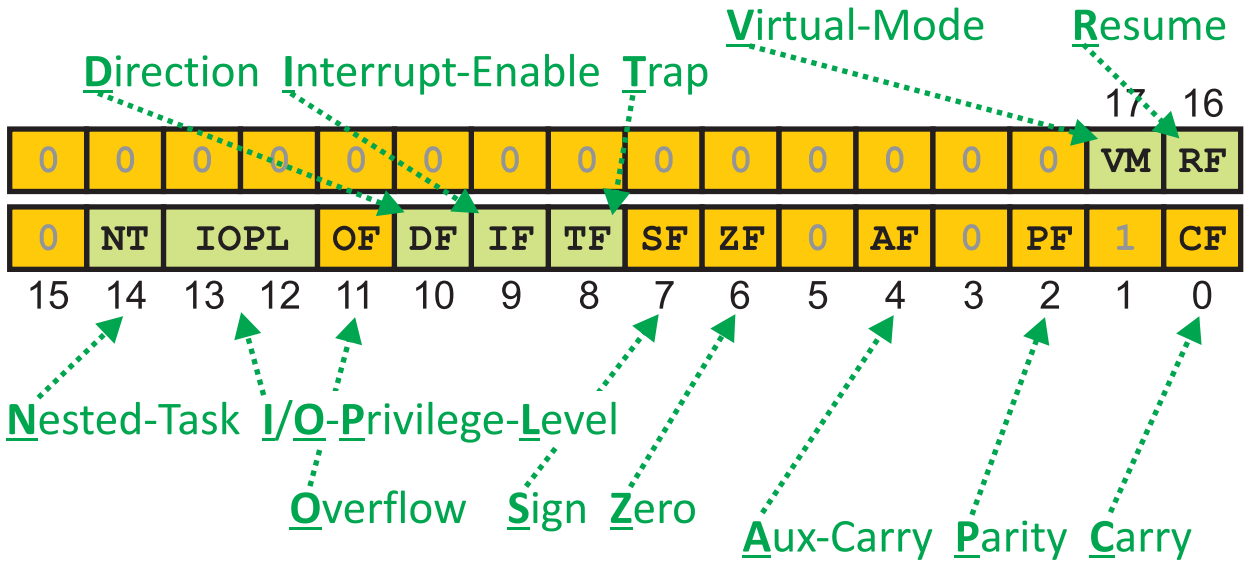
ES:EDI - cílová adresa pro instrukce blokového přesunu dat

Kopírování řetězce (pole bytů) v assembleru:

```
LDS SI, <adresa zdrojového řetězce>
LES DI, <adresa cílového řetězce>
MOV CX, <počet prvků řetězce>
REP MOVSB
```

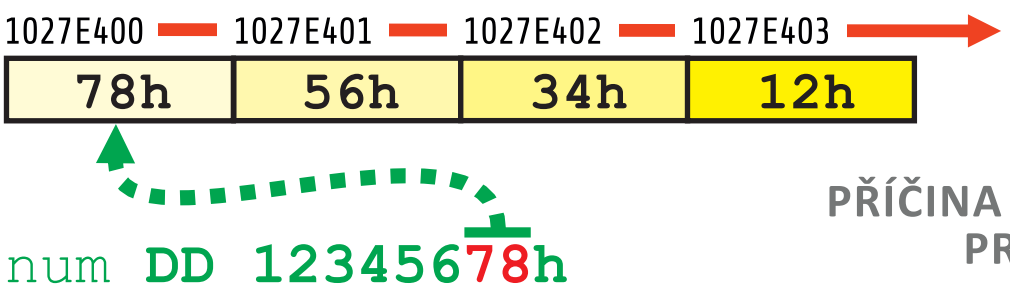
Registry procesorů Intel 80x86
Příznakový registr EFLAGS

- Obsahuje dva druhy **vlajek** (**1-bitových příznaků**):
  (i) **nastavované procesorem** po provedení instrukce indikují vlastnosti výsledku (**CF**, **PF**, **AF**, **ZF**, **SF**, **OF**),
  (ii) **nastavované programátorem** řídí činnost procesoru (**TF**, **IF**, **DF**, **VM**, **RF**, **NT**, **IOPL**).



Endian procesoru
Uspořádání dat ve slovech v paměti

Procesory Intel (a jejich klony) používají tzv. ***Little Endian***, což znamená, že nižší řády čísla jsou na nižších adresách:



PŘÍČINA ČASTÝCH PROBLÉMŮ

- Procesory Motorola, AIM PowerPC (po G5), Sun SPARC (do verze V9), IBM System/370 používají ***Big Endian***, tzn. nižší řády na vyšších adresách (tj. tak, jak se číslo píše na papír).
- Některé procesory umí endian podle potřeby přepínat, např. ARM, SPARC V9, MIPS, PA-RISC, IA64, DEC Alpha, některé PowerPC ⇒ tzv. ***Bi-Endian***.

Techniky adresování paměti
Typy adres

- 3 druhy adres:
  (i) ***logická*** (virtuální) – adresuje 64 TB virtuální paměti, skládá se z 16-bitového ***selektoru*** a 32-bitového ***offsetu***. Segmentační jednotka CPU jí převádí na lineární adresu.
  (ii) ***lineární*** – 32-bitová adresa (adresuje 4 GB paměti). Je-li stránkovací jednotka vyřazena, adresuje přímo fyzickou paměť počítače.
  (iii) ***fyzická*** – 32-bitová adresa (tj. adresuje až 4 GB), která představuje skutečnou pozici v paměti počítače. Je shodná s lineární adresu, pokud není v činnosti stránkovací jednotka.

Programátor aplikací nejčastěji pracuje pouze s offsetem logické adresy (méně často s úplnou logickou adresou). Adresy (ii) a (iii) se využívají pouze při programování jádra OS či driverů.

Adresování paměti v praxi
Čtení/zápis do paměti v assembleru

```
_DATA SEGMENT
   prompt  DB 'Hello, world!', 0
_DATA ENDS

_CODE SEGMENT
   MOV   EAX, OFFSET prompt
   ADD   EAX, 12
   MOV   BL, '?'
   MOV   [EAX], BL

   MOV   EAX, OFFSET prompt
   PUSH  EAX
   CALL  WriteLine@4
_CODE ENDS
```

získání adresy objektu (proměnné, funkce, atp.)

zapsání hodnoty z registru BL na adresu uvedenou v registru EAX

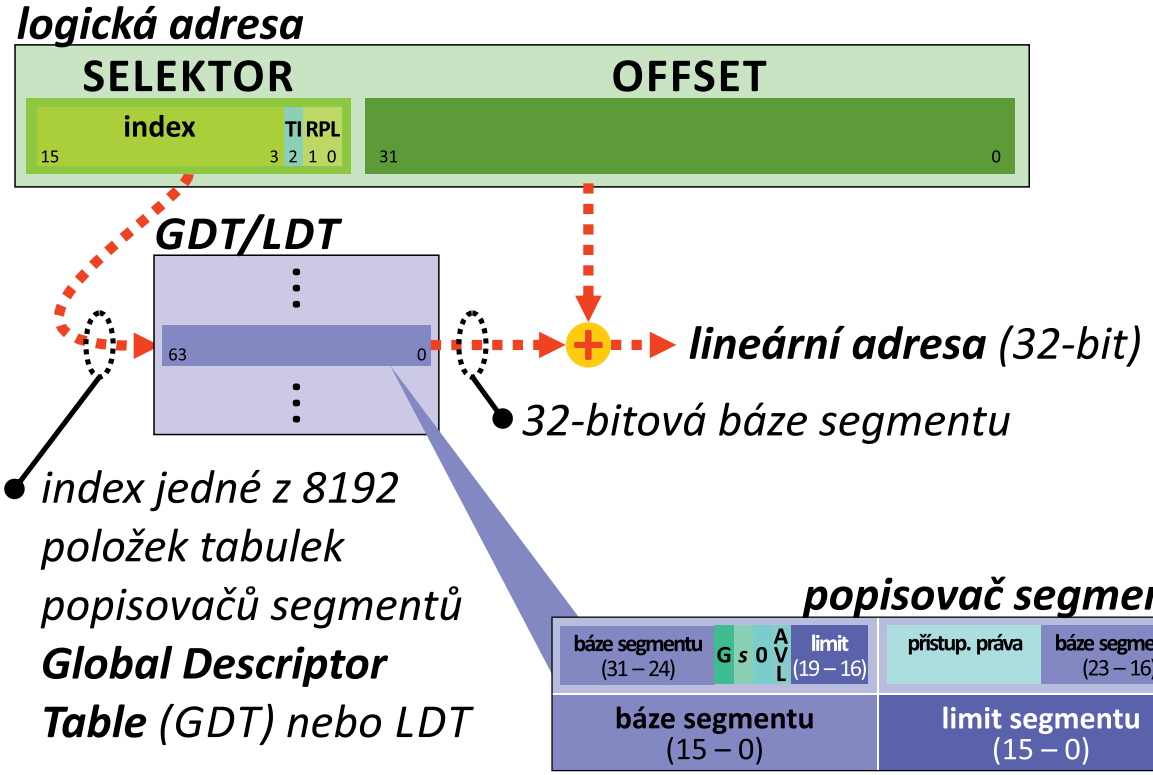
Segmentace a stránkování paměti
Smysl a účel transformace adres

- U komplexního výpočetního systému **nemůže mít adresa význam** přímo **pozice v paměťových obvodech**:
  (i) všechny konfigurace takového systému by musely mít **stejnou velikost paměti**;
  (ii) nemožné implementovat **ochranu adresního prostoru** jednotlivých běžících procesů, tj. velmi problematická implementace multitaskingu;
  (iii) nemožné používat **virtuální paměť**, tj. více paměti pro procesy, než je fyzicky instalováno v počítači;
  (iv) další, avšak mnohem méně důležité důvody.

- Proto mají procesory Intel (a klony) řady 80x86 implementované sofistikované mechanismy transformace adres.

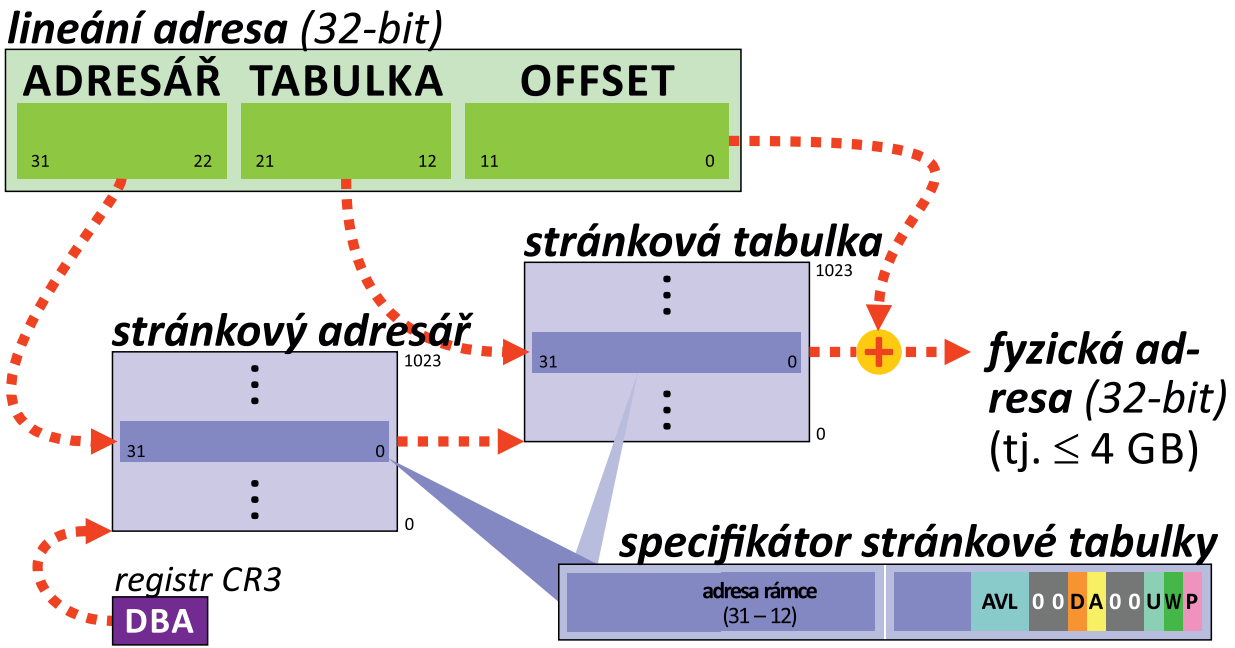
Segmentace paměti
Dvě úrovně transformace adres (úroveň 1)

- převod **logické adresy** na **lineární** via GDT/LDT



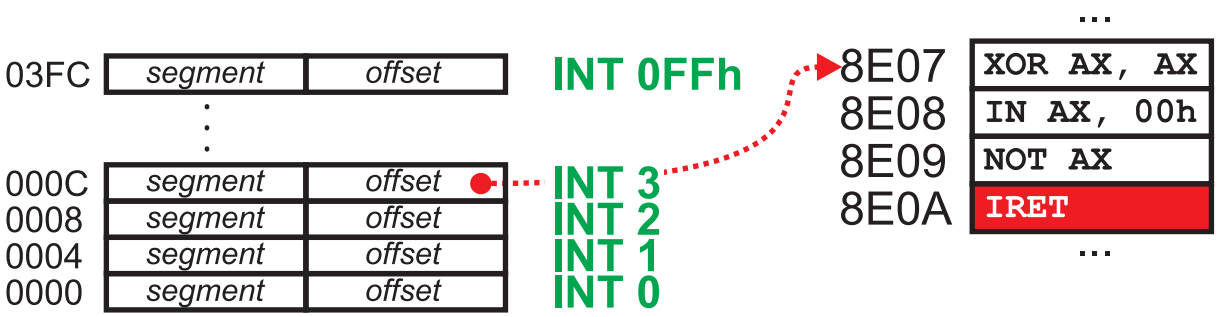
Stránkování paměti
Dvě úrovně transformace adres (úroveň 2)

- převod **lineární adresy** na **fyzickou** prostřednictvím stránkovací jednotky CPU (*Paging Unit*)



Přerušení (Interrupt)
Jednoduché µ-procesory, Intel 8086

- **tabulka přerušovacích vektorů** je umístěná na fyzickém počátku paměti od adresy 0 (i8086 0000:0000)
- adresa ukazuje na začátek obslužné rutiny přerušení (musí končit instrukcí **`IRET`**)
- přerušení může být vyvolané buď HW (z vnějšku přivedením log. úrovně na daný pin procesoru) či SW instrukcí **`INT n`**
- HW přerušení lze **maskovat** vynulováním příznaku **`IF`** instrukcí **`CLI`** (kromě vnitřních a nemaskovatelných, tzv. ***Non-Maskable Interrupt*** / NMI).



Činnost CPU při přerušení
Jednoduché µ-procesory, Intel 8086

Nastalo přerušení **n** nebo CPU dekódoval instrukci **`INT n`**:

(i) do zásobníku se uloží registr příznaků (**FLAGS**),

(ii) vynulují se příznaky **IF** a **TF**,

(iii) do zásobníku se uloží registr **CS**,

(iv) **CS** se naplní obsahem adresy **n** * 4 + 2,

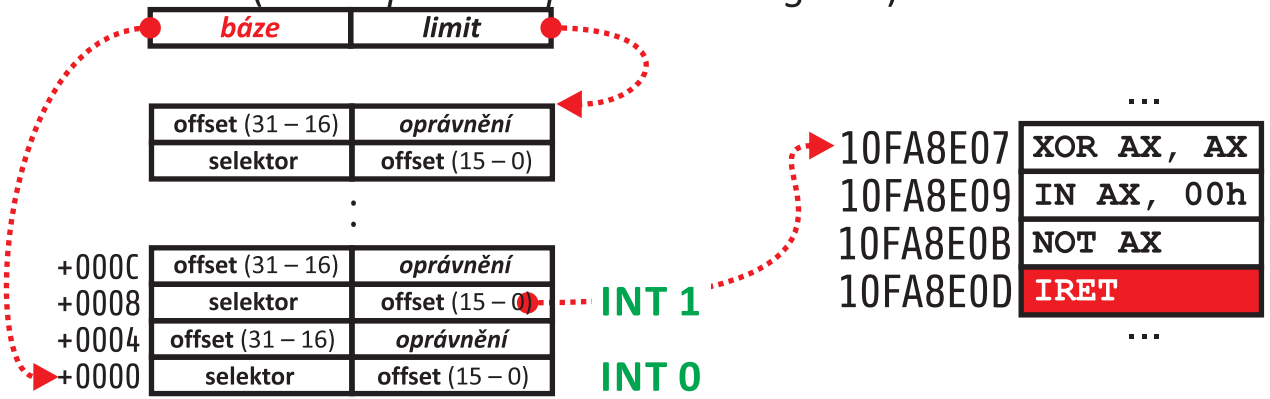
(v) do zásobníku se uloží **IP** ukazující na další **neprovedenou instrukci** přerušeného procesu,

(vi) **IP** se naplní obsahem adresy **n** * 4.

Přerušení (Interrupt)
Komplexní CPU se segmentací paměti

- **tabulka popisovačů obsluhy přerušení** (**IDT** = ***Interrupt Descriptor Table***) může být umístěna kdekoliv v paměti, adresa IDT je umístěna v registru **IDTR** (čte/plní se instrukcí **SIDT/LIDT**)
- položka IDT se nazývá **popisovač brány přerušení**, brány jsou pro (i) maskovatelná přerušení (Interrupt Gate) a (ii) nemaskovatelná přerušení (Trap Gate)

IDTR (*Interrupt Descriptor Table Register*)



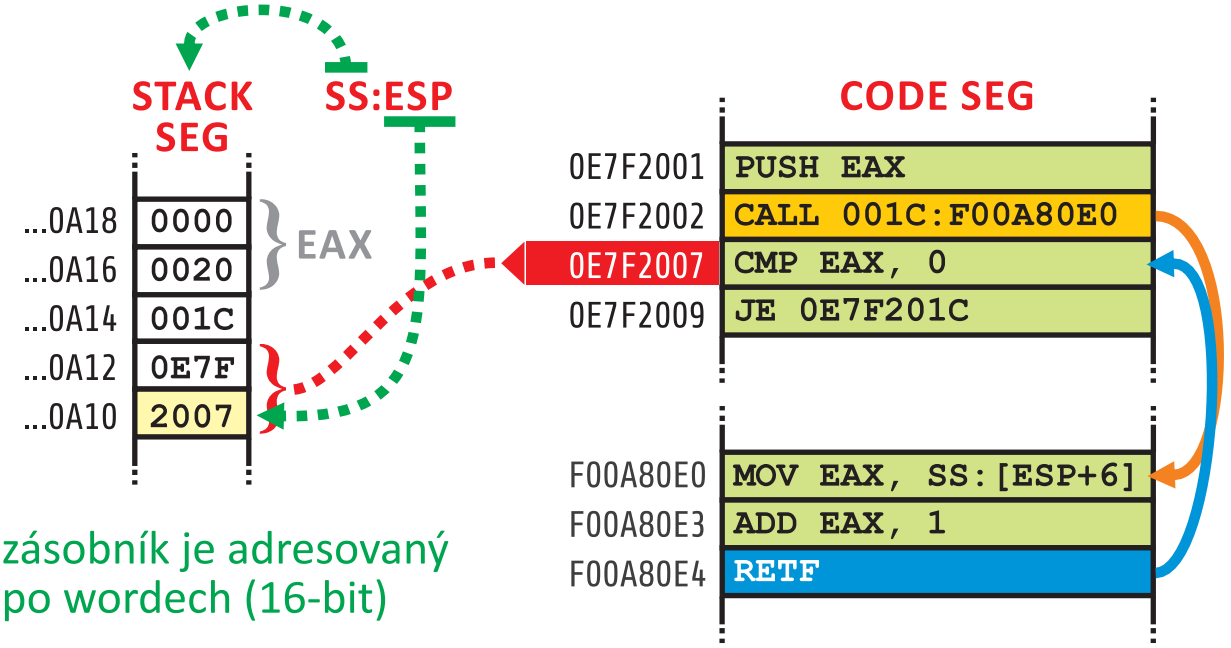
Hardwarová přerušení
Upozornění CPU na výjimečné stavy z vnějšku

- **`INT 0`** až **`INT 15`** jsou rezervována pro ošetřování výjimečných stavů generovaných hardwarem počítače

INT_NUM	Short Description PM
0x00	Division by zero
0x01	Debugger
0x02	NMI
0x03	Breakpoint
0x04	Overflow
0x05	Bounds
0x06	Invalid Opcode
0x07	Coprocessor not available
0x08	Double fault
0x09	Coprocessor Segment Overrun *(386 or earlier only)*
0x0A	Invalid Task State Segment
0x0B	Segment not present
0x0C	Stack Fault
0x0D	General protection fault
0x0E	Page fault
0x0F	*reserved*
0x10	Math Fault
0x11	Alignment Check
0x12	Machine Check
0x13	SIMD Floating-Point Exception

Volání podprogramů
Mechanismus předávání parametrů

- před odskokem do podprogramu se do zásobníku uloží ***návratová adresa*** (její tvar záleží na typu volání: **near** / **far**)
- parametry se podprogramu předávají buď přes zásobník nebo v registrech (záleží na překladači: **stdcall** / **fastcall**)



zásobník je adresovaný po wordech (16-bit)

Volání podprogramů
Realizace v assembleru

- pokud se podprogram v ASM linkuje k programu v C, musí se shodovat ***paměťový model*** (použití segmentových registrů) a ***volací konvence*** (způsob předávání parametrů)

```
_TEXT SEGMENT WORD PUBLIC 'CODE'
        public _power2
_power2 proc near
        push ebp
        mov ebp, esp
        mov eax, [ebp+4]  ; první argument
        mov ecx, [ebp+6]  ; druhý argument
        shl eax, cl       ; EAX = EAX * ( 2 ^ CL )
        pop ebp
        ret
_power2 endp
_TEXT   ends
        END
```

pod tímto názvem funkci uvidí linker, tj. v jazyce C jí lze volat jako `int power2(int, int)`

paměťový model **FLAT**, tj. **CS** = **DS** = **SS** (= ES, FS, GS) – vše je v jediném segmentu (jinak: `mov eax, ss:[ebp + 4]`)

Paměťové modely
Alokace segmentů pro potřeby procesu

- ***Paměťový model*** (na procesorech Intel 6 různých) určuje, jak jsou procesu alokovány segmenty paměti, tj. co je obsahem segmentových registrů a jaký tvar a velikost má ukazatel.
- 16-bitové programy: TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE
- 32-bitové programy: **FLAT** (= TINY), tzn. všechny segmentové registry jsou nastavené na stejnou hodnotu (selektor)
- 64-bitové programy: 7 různých paměťových modelů

- Problematika paměťových modelů (a v návaznosti na to segmentace paměti a stránkování) je velmi rozsáhlá a komplikovaná, mimo rámec předmětu KIV/PC. Vážní zájemci nechť si prostudují **Intel® 64 and IA-32 Architectures Developer's Manual**, PDF ke stažení z http://www.intel.com

Porty procesoru
Komunikace s vnějšími HW zařízeními

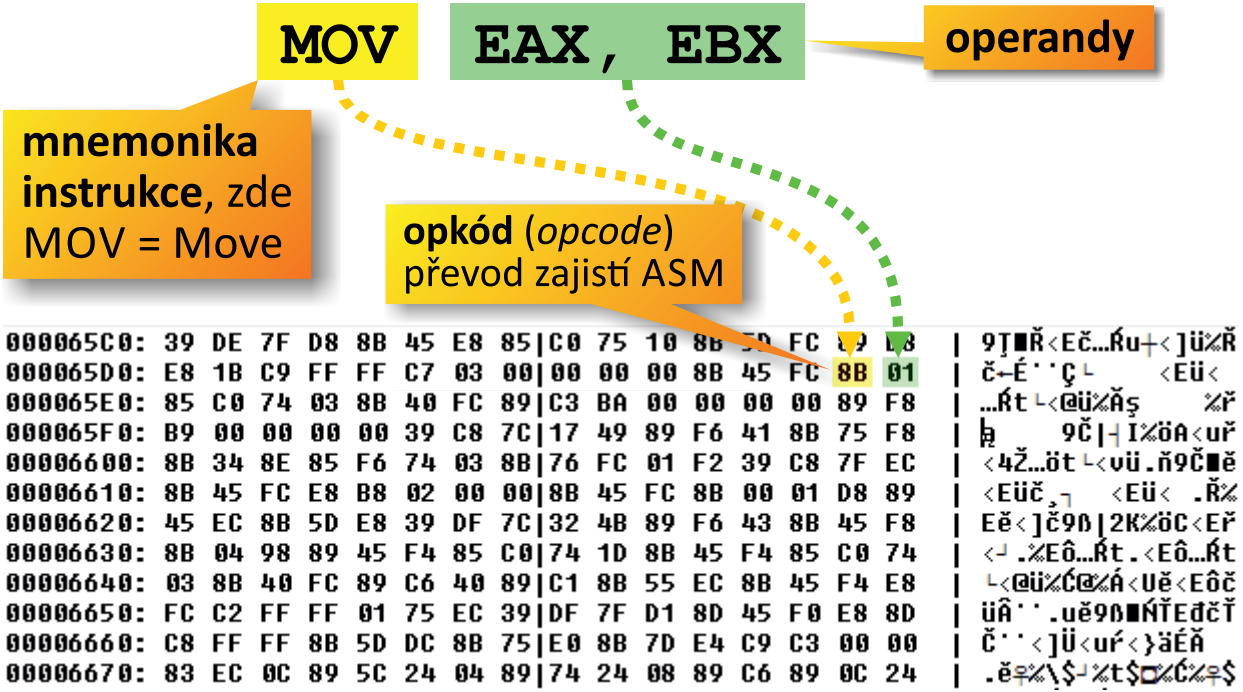
- pomocí tzv. ***I/O portů***, sběrnice může přenášet data buď mezi CPU a pamětí nebo CPU a ostatními zařízeními na MB
- signál M/$\overline{IO}$ určuje, zda adresa nastavená CPU na adresních vodičích A_0 – A_{31} je adresou v paměti nebo I/O portu

```
@L1:
   mov al, 0Ah ; 0Ah - offset 'valid'
   out 70h, al ; 70h - CMOS index port
   in al, 71h  ; 71h - CMOS data port
   test al, 10000000b
   jnz @L1     ; bit7 = 1, znovu

   xor al, al
   out 70h, al

   in al, 71h  ; čteme bajt 0 - sekundy
```

```
in  eax, 61h
in  eax, dx

out 20h, eax
out dx, eax
```

Instrukční sada
Tvar instrukcí CPU s úplnou instrukční sadou

- CPU Intel – tzv. ***úplná instrukční sada*** (***Complete Instruction Set Computer***, CISC), řádově stovky instrukcí



Instrukční sada
Instrukce pro přesun dat

- Instrukce **MOV** přenáší obsah zdrojového operandu (tj. toho, který stojí více vpravo) do cílového operandu. Obsah zdrojového operandu se nemění.

```
MOV  ESI, OFFSET retez1
MOV  EDI, OFFSET retez2
MOV  ECX, 10
REP  MOVSB
```

- **MOVSB/MOVSW/MOVSD** – byte/word/dword z adresy DS:ESI přepíše na adresu ES:EDI. Lze použít prefix **REP**, který zajistí opakování podle obsahu registru CX (při každém cyklu se sníží hodnota CX, při dosažení 0 cyklus končí).

Instrukční sada
Aritmetické instrukce

```
MOV   EAX, 0
INC   EAX
SUB   EAX, EBX
DEC   EAX, CL
MUL   EBX
NEG   EAX
DIV   EDX
```

EAX ← EAX + 1

EAX ← EAX - EBX

EAX ← EAX - CL

EAX ← EAX × EBX

EAX ← -EAX (dvojkový doplněk)

EAX ← EAX / EDX

- **MUL**: Je-li výsledek větší než 32-bitů, je umístěn do reg. páru EDX+EAX
- **DIV**: Je-li operand 32-bitový, dělí se obsah páru EDX+EAX. Do EAX je uložen celočíselný podíl, do EDX zbytek po dělení.

Instrukční sada
Logické instrukce

- **`CMP`** (= Compare): Porovnává zdrojový a cílový operand.
- **`AND`**: logický součin
- **`OR`**: logický součet
- **`XOR`**: nonekvivalence
- **`NOT`**: negace (jedničkový doplněk)
- **`TEST`**: logické porovnání pomocí konjunkce

Instrukcí jsou stovky – nemá smysl je všechny dopodrobna rozebírat (není na to v rámci KIV/PC čas).

Zájemci o programování v assembleru:

- Vlad Pirogor: *Mistrovství v jazyce Assembler*. Computer Press, Brno, 2005. ISBN 80-251-0888-0.
- Intel® 64 and IA-32 Architectures Software Developer Manuals. http://www.intel.com