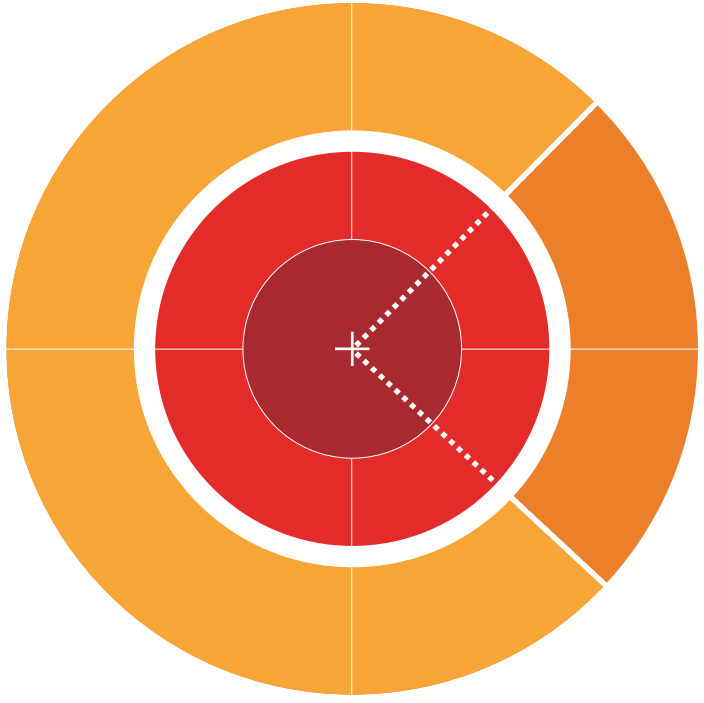
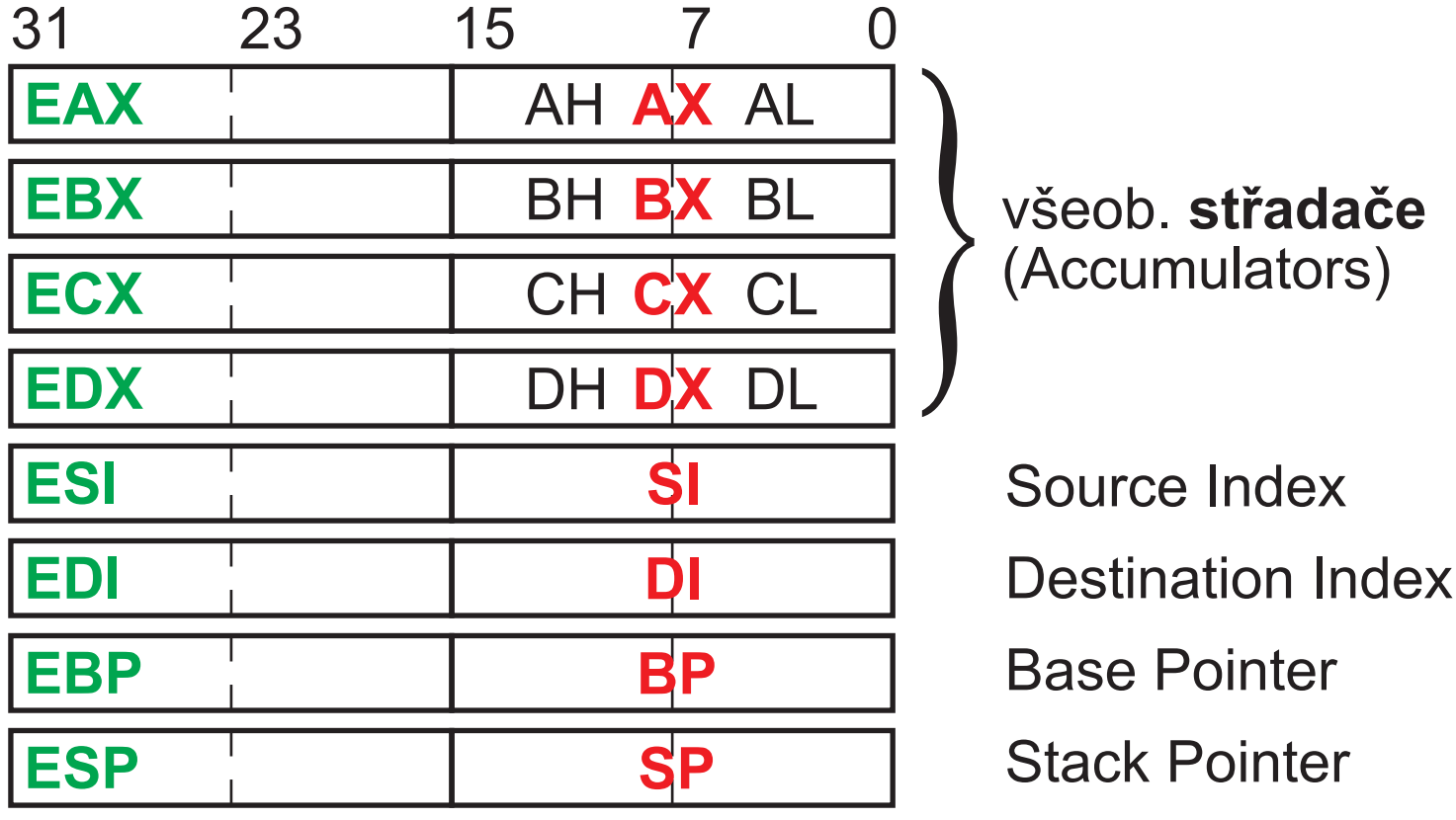


Programátorský model x86

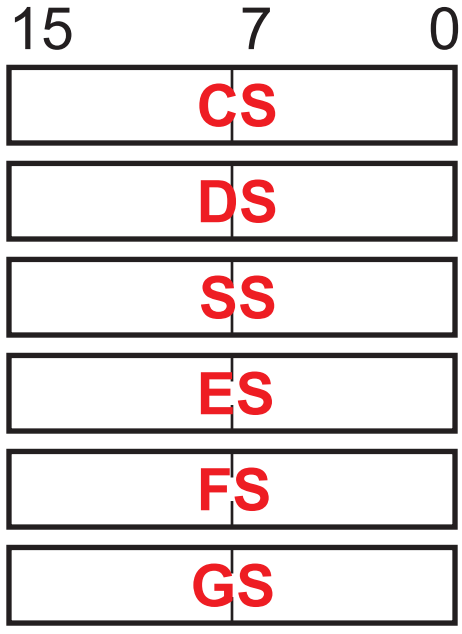
- programátorským modelem se rozumí soubor vlastností a fyzických součástí procesoru, které ovlivňují jeho programování v nízkoúrovňových jazycích
- zejména popisuje **uspořádání paměti**, využitelné **registry**, nativní **typy dat** daného procesoru, **instrukční soubor**, **přerušovací systém**, atp.

Všeobecné registry procesoru x86 (od 80386 dále)



- tyto registry jsou obecně použitelné v programu k dočasnému ukládání dat (až na ESP - s tím opatrně)
- naplňují se instrukcí **`MOV`** *reg*`,` *hodnota*, např. **`MOV BL, 5`**

Segmentové registry procesoru x86 (od 80386 dále)



Code Segment

Data Segment

Stack Segment

Extra Segment

Extra Segment

Extra Segment

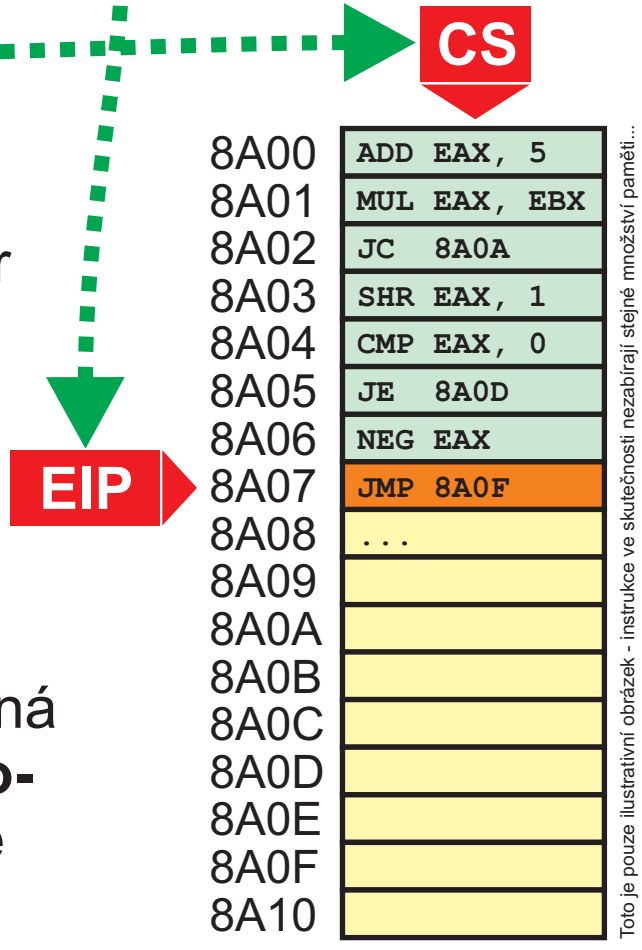
- viditelná část segmentových registrů je 16-bitová, plnit je lze instrukcí **`MOV`** nebo spec. instrukcemi **`LDS`**, **`LES`**, **`LFS`**, **`LGS`** a **`LSS`**, např. **`LDS EBX,`** *dword-ptr*, která umístí do registrového páru DS:EBX 32-bitovou adresu *dword-ptr*
- **změna obsahu CS má fatální následky** (CS obsahuje tzv. selektor kódového segmentu)

Registry se zvláštním významem

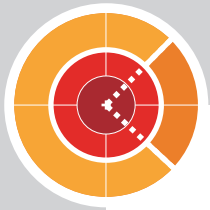
CS:**EIP** (Extended Instruction Pointer)

selektor segmentu

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



EIP ukazuje na pozici v kódovém segmentu, kde leží právě prováděná instrukce, tj. **pár CS:EIP udává pozici právě vykonávané instrukce**



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

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

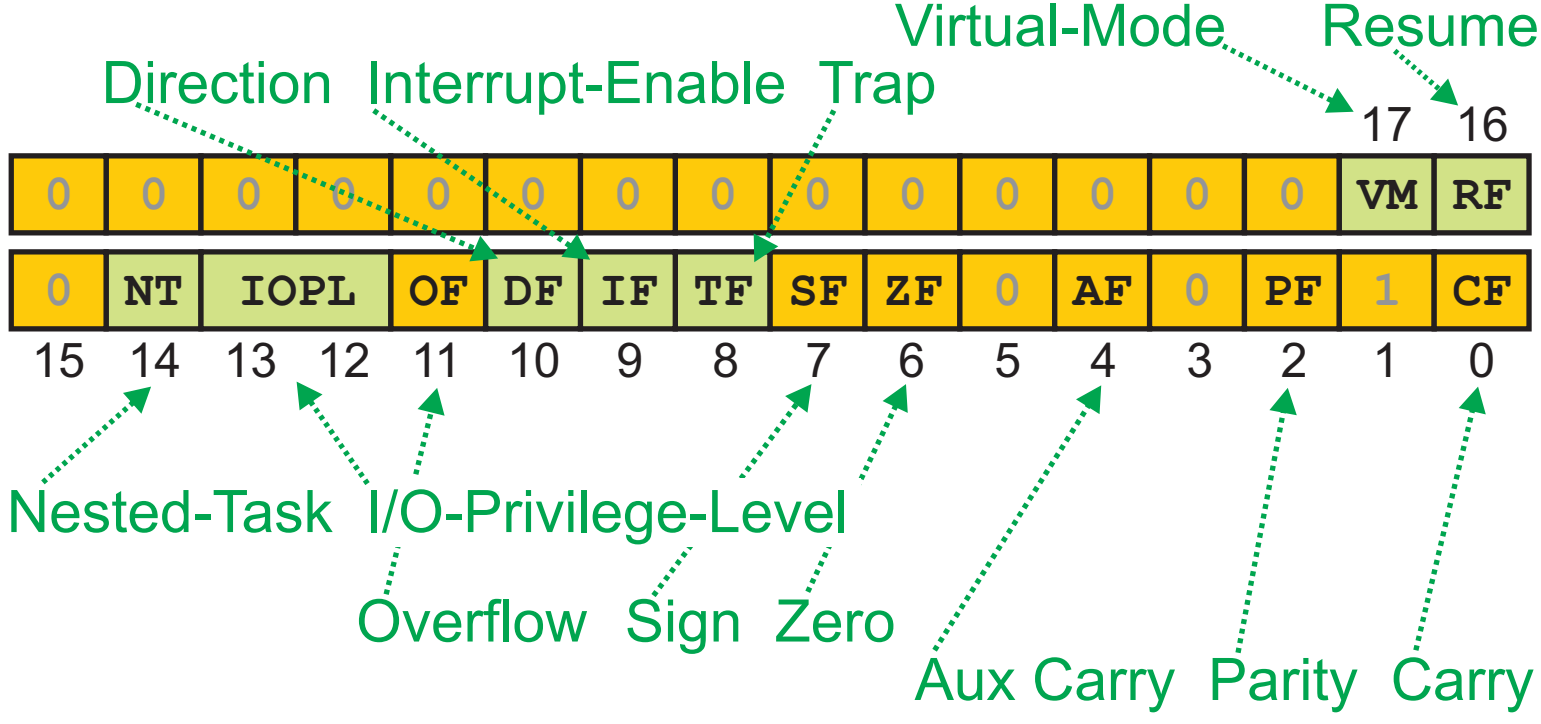
kopírování řetězce (pole bytů)

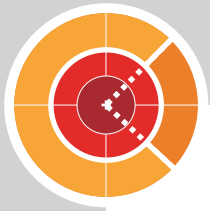
Příznakový registr EFLAGS (32-bitový)

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`).





Typy dat

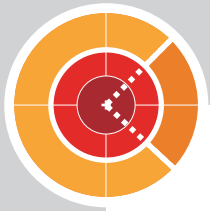
- **byte** (8 bitů), deklarace v asm instrukcí **DB** (Define Byte)
- **word** (16 bitů), deklarace **DW** (Define Word)
- **dword** (32 bitů), deklarace **DD** (Define Double-word)
- **qword** (64 bitů), deklarace **DQ** (Define Quad-word)

Jako operandy instrukcí akceptuje 80386 maximálně 32 bitů ve 32-bitových registrech (**E??**).

Některé instrukce pracují i se 64-bitovými slovy, ta se pak předávají v **registrových párech** EAX & EDX a EBX & ECX (vždy takto spolu).

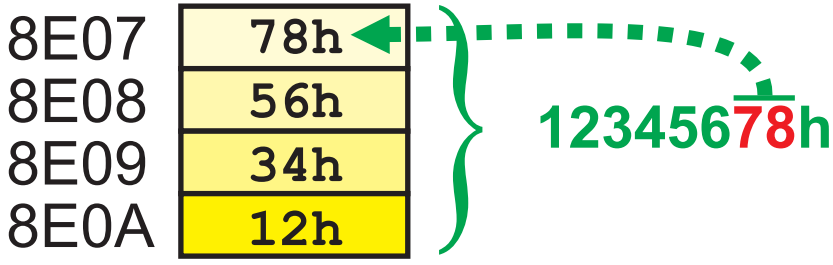
```
.DATA
     mstr db 'Hello', 0
     xp   db 100
     icnt dw ?
     ptrs dd 20 dup(0)

.CODE
     ...
```



Endian procesoru (čili uspořádání bytů ve slovech)

Procesory Intel (a jejich klony) používají **Little Endian**, tj. nižší řády jsou na nižších adresách:

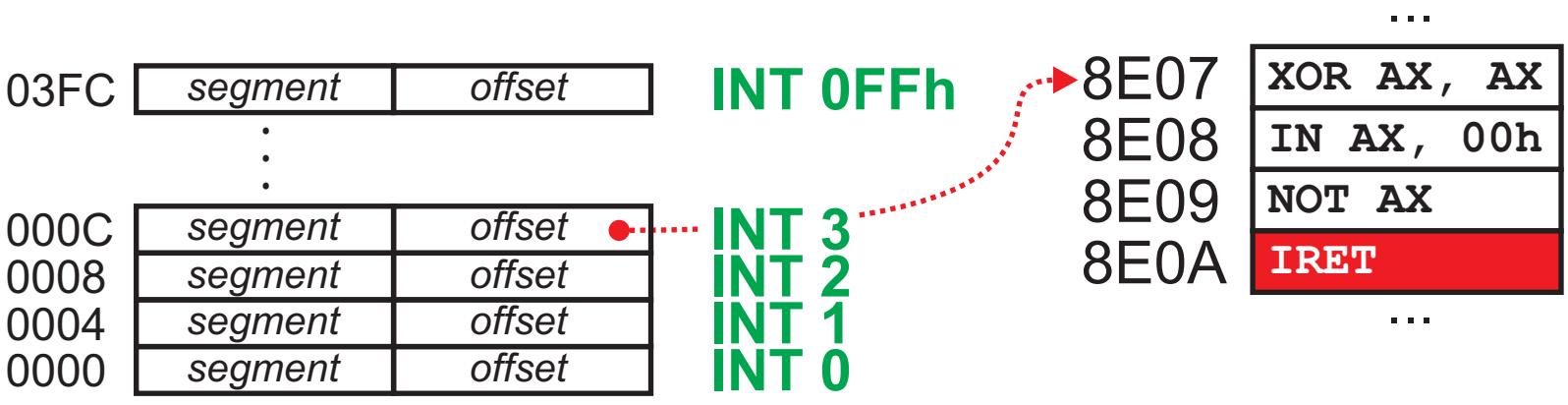


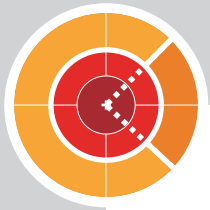
Procesory Motorola, AIM PowerPC (po G5), Sun SPARC (starší verze do V9), IBM System/370 používají **Big Endian**, tzn. nižší řády na vyšších adresách (vlastně tak, jak se číslo píše na papír).

Některé procesory umí endian podle potřeby přepínat, např. ARM, SPARC V9, MIPS, PA-RISC, IA64, DEC Alpha, některé PowerPC => tzv. **Bi-Endian**.

Přerušení (8086)

- **tabulka přerušovacích vektorů** je umístěná na fyzickém počátku paměti od adresy 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) nebo SW instrukcí **`INT n`**
- HW přerušení lze **maskovat** vynulováním příznaku **`IF`** instrukcí **`CLI`** (kromě vnitřních a NMI).





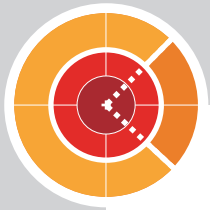
Činnost CPU při přerušení (8086)

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

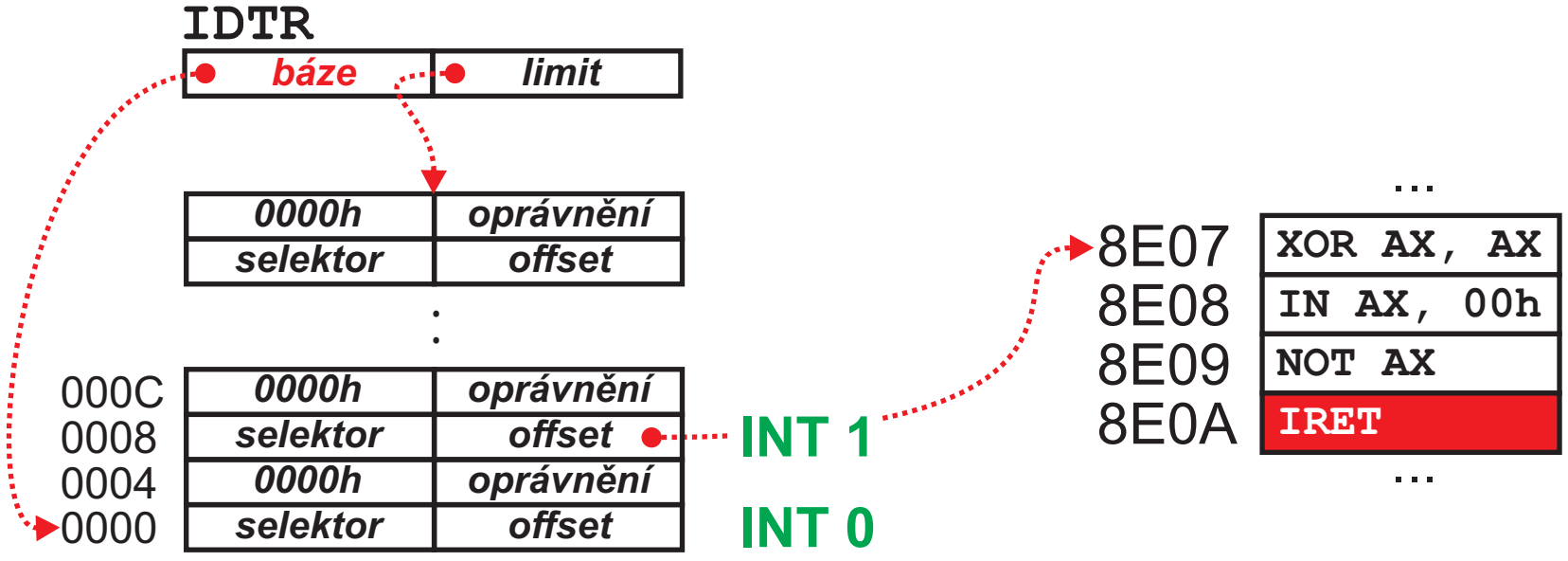
INT n	Význam
0	Dělení nulou (Divide By Zero)
1	Krokovací režim (Single-Step)
2	Nemaskovatelná přerušení (NMI)
3	Ladicí bod (Breakpoint Trap)
4	Přeplnění (Overflow Trap)

} záleží na operačním systému, jak tato přerušení obslouží



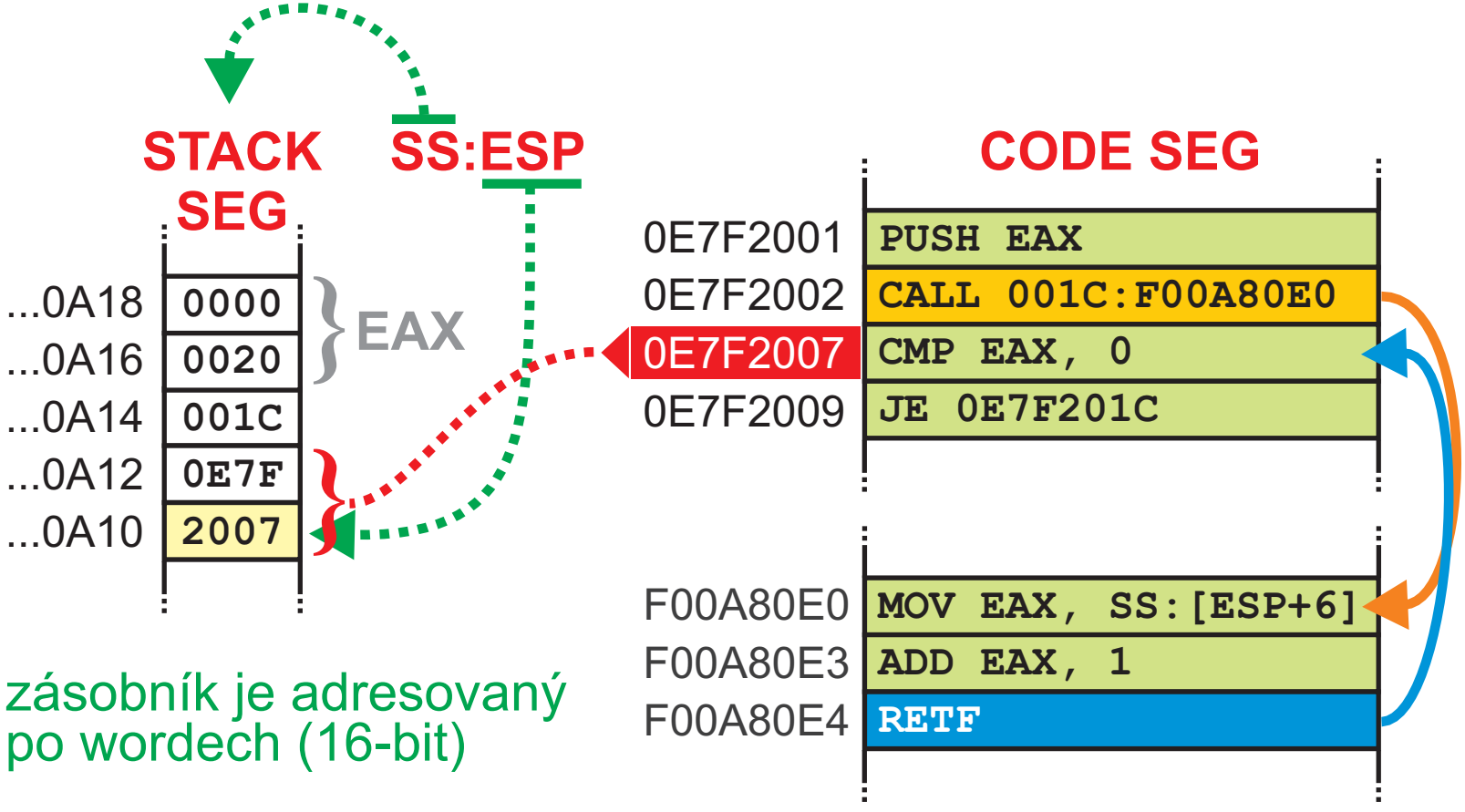
Přerušení (80386 a dále) - zjednodušeně

- **tabulka popisovačů segmentů 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)

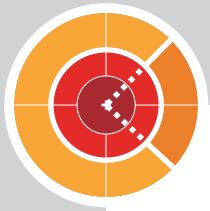


Volání podprogramů

- záleží na **paměťovém modelu**, jaká návratová adresa se ukládá do zásobníku
- předávání parametrů je na programátorovi či překladači



zásobník je adresovaný po wordech (16-bit)



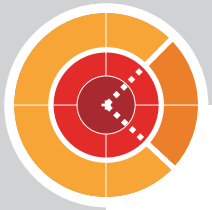
Volání podprogramů (assembler)

- pokud se externí modul v assembleru linkuje k programu přeloženému překladačem C, musí se shodovat **paměťový model** a **volací konvence** (způsob předávání p-metrů)
- různé překladače používají různé PM a VK

```
_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
```

paměťový model **FLAT**
tj. **CS** = **DS** = **ES** = **SS**

assembler **MASM-like**, překladač **Microsoft Visual C/C++ 2005**



Paměťové modely

- paměťový model určuje, jaká část programu je umístěna v jakém segmentu (kód, data, zásobník), čím jsou tedy naplněné segmentové registry a jak "velké" jsou pointery
- situace je poněkud nepřehledná, na **16-bitové platformě Intel x86** existuje 6 paměťových modelů:

TINY, **SMALL**, **MEDIUM**, **COMPACT**, **LARGE**, **HUGE**

- moderní 32- a 64-bitové platformy používají zejména model **FLAT**, což je obdoba **TINY**, tj. všechny segmentové registry jsou nastavené na stejnou hodnotu
- výše uvedený paměťový model se takto jeví z hlediska programátora, nikoliv operačního systému - ten techniku segmentace paměti využívá (oprávnění, stránkování, atd.)

- **problematika je značně rozsáhlá a komplikovaná, mimo rámec předmětu PC, zájemci** `http://www.intel.com`

Komunikace procesoru s okolními zařízeními

- děje se pomocí I/O portů, sběrnice může přenášet data buď mezi CPU a pamětí nebo ostatními zařízeními na MB
- signál M/$\overline{IO}$ určuje, za adresa nastavená na adresních vodičích A_0 - A_{15} je adresou paměti nebo I/O portu

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

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

```
@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
```