

# B4B350SY: Operační systémy

## Lekce 1. Úvod do operačních systémů

Petr Štěpán

stepan@fel.cvut.cz



September 18, 2017

# Outline

- 1 Úvod
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS osobního počítače

# Outline

- 1 Úvod
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS osobního počítače

# B4B350SY - Operační systémy

Přednášející:

Michal Sojka K13135/CIIRC

`sojkam1@fel.cvut.cz`

Petr Štěpán, K13133

`stepan@fel.cvut.cz`

# Materiály

- SILBERSCHATZ A., Galvin P. B., Gagne G.: Operating System Concepts  
<http://codex.cs.yale.edu/avi/os-book/OS7/os7c/index.html>
- TANENBAUM, Andrew S a Albert S WOODHULL. Operating systems design and implementation. 3rd ed. Upper Saddle River: Prentice-Hall, c2006, xvii, 1054 s. ISBN 0131429388
- <http://wiki.osdev.org/>
- <http://hypervisor.org/>
- YouTube lectures (anglicky):
  - CS 162 – UC Berkeley
  - OS-SP06 – Surendar Chandra – UC Berkeley
  - MIT 6.004

# Organizace předmětu

- Souhrná podrobná literatura v češtině není
- Tyto prezentace (stránka předmětu <https://cw.fel.cvut.cz/wiki/courses/b4b35osy>)
- Cvičení částečně seminární, více samostatná práce, nutná domácí příprava
- Zkouška:
  - Body ze cvičení
  - Písemná zkouška
  - Ústní část - dobrovolná (nutná pro A)

# Cíle předmětu

Podle Hospodářských novin se Informatika vyučuje nejlépe na FEL, ČVUT (22.1.2015)

- OS patří k základům informatiky
- Poznat úkoly OS a principy práce OS
- Využívat OS efektivně a bezpečně

Co NENÍ cílem tohoto předmětu

- Naučit Vás jak napsat aplikaci pod (X)Windows
- Naučit triky pro konkrétní OS
- Vytvořit OS – na to je málo času

# Proč studovat OS

- Pravděpodobně nikdo z vás nebude psát celý nový OS
- Proč tedy OS studovat?
  - Každý ho používá a jen málokdo ví jak pracuje
  - Jde o nejrozsáhlejší a nejsložitější IT systémy
  - Uplatňují se v nich mnohé různorodé oblasti
    - softwarové inženýrství,
    - netradiční struktury dat,
    - sítě, algoritmy, ...
  - Čas od času je potřeba OS upravit
    - pak je potřeba operačním systémům rozumět
    - psaní ovladačů, ...
  - Techniky užívané v OS lze uplatnit i v jiných oblastech
    - neobvyklé struktury dat, krizové rozhodování, problémy souběžnosti, správa zdrojů, ...
    - mnohdy aplikace technik z jiných disciplín (např. operační výzkum)
    - naopak techniky vyvinuté pro OS se uplatňují v jiných oblastech (např. při plánování aktivit v průmyslu)



# Outline

- 1 Úvod
- 2 Malý návod na použití školy**
- 3 Co je operační systém
- 4 OS osobního počítače

# Cíle vzdělávání

- V obecné rovině
  - Naučit kriticky myslet
  - Naučit hledat zákonitosti
  
- V konkrétní rovině
  - Předat nějaké konkrétní znalosti (co je posix, cache, sběrnice)
  - Předat nějaké konkrétní dovednosti (jak se programuje, jak efektivně vést projekt)

# O dobrém a špatném učení

- Povrchní přístup k učení
  - Úkoly dělám, abych splnil jejich zadání a dostal body
  - Výsledkem je zpravidla memorování
  
- Hlubkový přístup k učení
  - Úkoly dělám, abych splnil jejich účel
  - Výsledkem je zpravidla porozumění
  - Navíc je nutné najít účel úloh

# Proč porozumět a ne memorovat

- Schopnost spojit nové a dřívější znalosti
  - Pomáhá v chápání nových znalostí
  - Pomáhá odstranit chybné znalosti
- Schopnost použít znalosti
  - Znalosti lze spojit s každodenní zkušeností
- Schopnost uchovat znalosti
  - Dobře spojené a pochopené znalosti se pamatují déle
- Volba je na Vás !

# Co bude na přednáškách

- Výkladu se nedá uniknout
  - Náplň je většinou předem k dispozici
- Je to jako kino, ne? Návštěva kina je:
  - Pasivní zážitek s občas zajímavým příběhem
  - Nemusíte příliš přemýšlet
  - Desítky miliónů \$ vynaložené na udržení Vaší pozornosti
- Aktivní učení
  - Charles Lin: Active Learning in the Classroom
  - <http://www.cs.umd.edu/class/sum2003/cmcs311/Notes/Learn/active.html>

# Co bude na přednáškách

- Když chodíte na přednášky
  - očekává se, že se něco naučíte
- V čem je problém
  - Látka je složitá, ale při poslouchání to člověku nepříjde
  - Většina věcí se jeví logická – myšlenkové zkratky
  - Pro zvládnutí je nutné se jí nějakou dobu věnovat i po přednášce
    - ACM/IEEE CS Curriculum: na 1 hodinu přednášky v bakalářském studiu připadají 2-3 hodiny domácí přípravy

# Jak se něco na přednášce naučit

- Neusnout
  - Bez ohledu na to, jak těžké to může být
  - Kdo spí, ten se nic nenaučí a přichází o souvislosti
- Chodit pravidelně
  - Nová látka staví na předchozích základech
  - Naučíte se lépe rozumět přednášejícímu
  - Když jsem minule nebyl, alespoň si přečíst přednášky
- Aktivně poslouchat
  - Nejlépe se nové věci naučíte při hledání vlastního vysvětlení, jak věci fungují
  - Dává smysl to co slyšíte?
  - Byli byste schopni to vysvětlit někomu, kdo na přednášce nebyl?
- Pokud něco nedává smysl
  - Zapište si co Vám nedává smysl
  - Zkuste vymyslet otázku, jejíž zodpovědění by věci vyjasnilo a položte ji přednášejícímu

# Kdy a jak se ptát

- Když Vaše představa neodpovídá tomu co slyšíte
  - Nebo když Vám chybí část „skládanky“
  - Na konci přednášky byste měli být schopni položit několik otázek, alespoň upřesňujících
    - „Myslím si, že říkáte ..(vlastními slovy).., je to tak?“
- Než se zeptáte, zkuste si odpovědět
  - Pokud nevíte, nebo si nejste jisti, zeptejte se
- Při hledání otázek začnete pozorněji poslouchat
  - Začnete poslouchat s cílem se něco naučit
  - Naučíte se klást užitečné dotazy



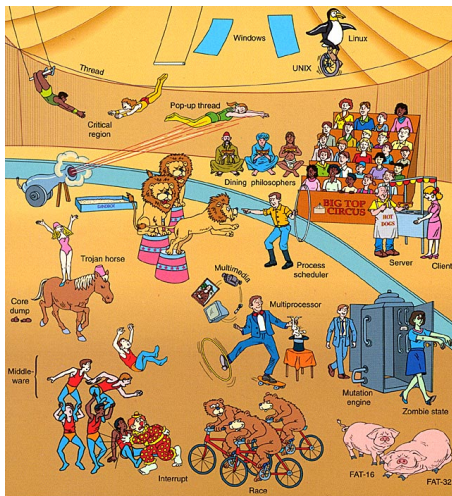
# Outline

- 1 Úvod
- 2 Malý návod na použití školy
- 3 Co je operační systém**
- 4 OS osobního počítače

# Co je operační systém

## Úkoly OS:

- Spouštět a dohlížet uživatelské programy
- Efektivní využití HW
- Usnadnit řešení uživatelských problémů
- Učinit počítač (snáze) použitelný
- Umíte použít počítač bez OS?



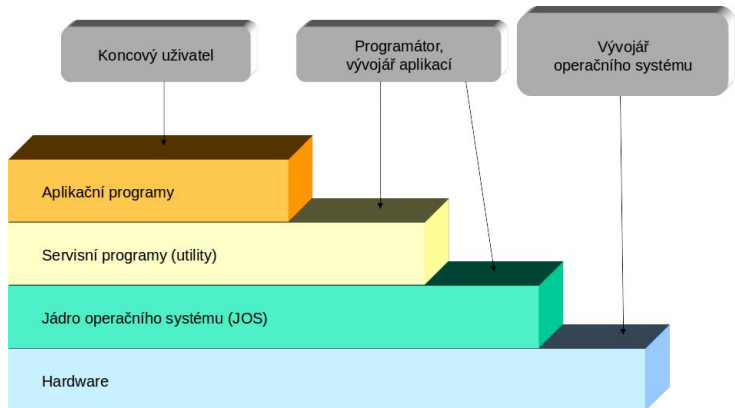
# Co je operační systém

- Neexistuje žádná obecně platná definice
- Několik koncepcí pojmu OS
  - systémové (jen jádro a s ním související nadstavby)
  - „obchodní“ (to, co si koupíme pod označením OS)
  - organizační (včetně pravidel pro hladký chod systému)
- OS jako rozšíření počítače
  - Zakrývá komplikované detaily hardware
  - Poskytuje uživateli „virtuální stroj“, který má se snáze ovládá a programuje
- OS jako správce systémových prostředků
  - Každý program dostává prostředky v čase
  - Každý program dostává potřebný prostor na potřebných prostředcích

# Co je operační systém

V této přednášce budeme brát operační systém jako jádro operačního systému

- ostatní (tzv. systémové) programy lze chápat jako nadstavbu jádra
- GUI – Windows je grafická nadstavba systémových programů

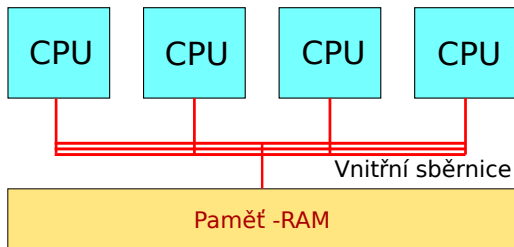


# R;znorodost OS

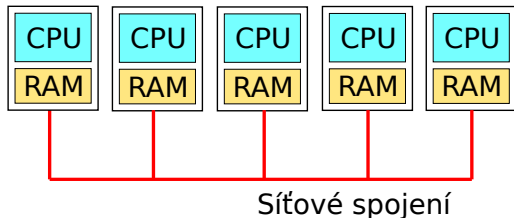
- OS „střediskových“ (mainframe) počítačů – dnes již historický pojem
- OS superpočítačů (3,120,000 jader, 54,902TFlops, 17,8MW příkon)
- OS datových a síťových serverů
- OS osobních počítačů a pracovních stanic
- OS reálného času (Real-time OS – řízení letadel, vlaků, raket, družic, apod.)
- OS přenosných zařízení – telefony, tablety
- Vestavěné OS (tiskárna, pračka, telefon, ...)
- OS čipových karet (smart card OS)
- ... a mnoho dalších specializovaných systémů

# Paralelní a distribuované systémy

Těsně vázaný  
multiprocesorový  
systém



Distribuovaný  
systém typu  
klient-server



# Systémy reálného času - RT

- Nejčastěji řídicí zařízení v dedikovaných (vestavěných) aplikacích:
  - vědecký přístroj, diagnostický zobrazovací systém, systém řízení průmyslového procesu, monitorovací systémy
  - obvykle dobře definované pevné časové limity
  - někdy také subsystém univerzálního OS
- Klasifikace:
  - striktní RT systémy – Hard real-time systems
    - omezená nebo žádná vnější paměť, data se pamatují krátkodobě v RAM paměti
    - protipól univerzálních OS nepodporují striktní RT systémy
    - plánování musí respektovat požadavek ukončení kritického úkolu v rámci požadovaného časového intervalu
  - tolerantní RT systémy – Soft real-time systems
    - použití např. v průmyslovém řízení, v robotice
    - použitelné v aplikacích požadujících dostupnost některých vlastností obecných OS (multimedia, virtual reality, video-on-demand)
    - kritické úkoly mají přednost „před méně šťastnými“

# Více úloh současně - Multitasking

- Zdánlivé spuštění více procesů současně je nejčastěji implementováno metodou sdílení času tzv. Time-Sharing Systems (TSS)
- Multitasking vznikl jako nástroj pro efektivní řešení dávkového zpracování
- TSS rozšiřuje plánovací pravidla
  - o rychlé (spravedlivé, cyklické ) přepínání mezi procesy řešícími zakázky interaktivních uživatelů
- Podpora on-line komunikace mezi uživatelem a OS
  - původně v konfiguraci počítač – terminál
  - v současnosti v síťovém prostředí
- Systém je uživatelům dostupný on-line jak pro zpřístupňování dat tak i programů

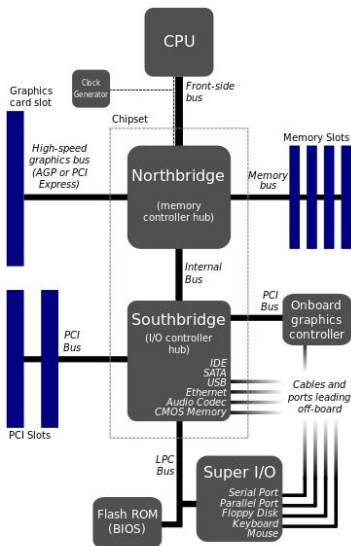


# Outline

- 1 Úvod
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS osobního počítače**

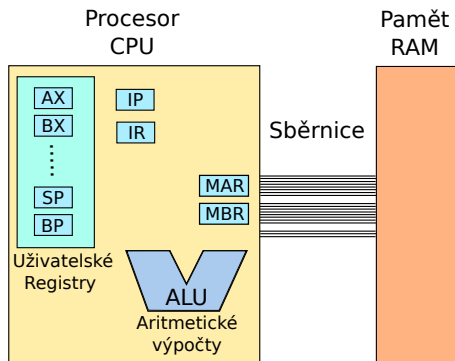
# Osobní počítač

- Základem počítače je procesor – CPU
- Procesor je připojen sběrnicemi k ostatním periferiím počítače – paměti, grafickému výstupu, disku, klávesnici, myši, síťovému rozhraní, atd.
- Činnost sběrnice řídí arbiter sběrnice



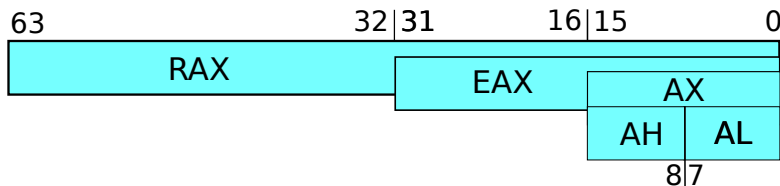
# Processor - CPU

- Základní vlastnosti:
- šířka datové a adresové sběrnice
- počet vnitřních registrů
- rychlost řídicího signálu – hodiny
- instrukční sada



# Processor - x86/AMD64

- Přehledný popis - [https://en.wikibooks.org/wiki/X86\\_Assembly](https://en.wikibooks.org/wiki/X86_Assembly)
- Všechny registry vzhledem ke zpětné kompatibilitě jsou 64/32/16/8 bitové



## Řídící a stavové registry

- EIP/RIP – instruction pointer – adresa zpracovávané instrukce
- EIR/RIR – instruction registr – kód zpracovávané instrukce
- EFLAGS/RFLAGS – stav procesoru povoleno/zakázáno přerušení, system/user mód, výsledek operace – přetečení, podtečení, rovnost 0, apod.

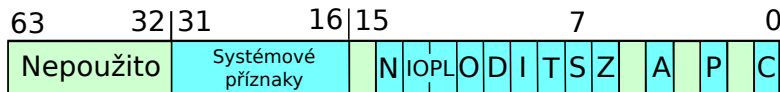
# Processor - x86/AMD64

## Uživatelské registry

- programově dostupné registry pro ukládání hodnot programu eax, ebx, ecx, edx
- registry umožňující uchovat hodnotu, nebo ukazatel do paměti esi, edi, ebp
- esp – stack pointer - ukazatel zásobníku, pro ukládání lokálních proměnných a návratových adres funkcí, používán při funkcích push, pop
- AMD64/X86-64 přidává 8 dalších registrů r8-r15, ve formě r8b nejnižší bajt, r8w nejnižší slovo (16 bitů), r8d – nižších 32 bitů, r8 – 64 bitový registr

# Registr FLAGS

FLAGS registr



C - Carry flag

P - Parity flag

Z - Zero flag

S - Sign flag

O - Overflow flag

I - Interrupt enable

T - Trap flag

IOPL - I/O privilege level

A - Adjust flag

# Režimy práce procesoru

## FLAGS registr

- Dva režimy práce procesoru IOPL - základ hardwarových ochran
  - Systémový CPL0 = privilegovaný režim
    - procesor může vše, čeho je schopen
  - Uživatelský CPL3 = aplikační (ochranný) režim
    - privilegované operace jsou zakázány
- Privilegované operace
  - ovlivnění stavu celého systému (halt, reset, Interrupt Enable/Disable, modifikace Flags, modifikace registrů MMU )
  - instrukce pro vstup/výstup (in, out)
- Přechody mezi režimy
  - Po zapnutí stroje systémový režim
  - Přechod do uživatelského – modifikace Flags (popf nebo reti)
  - Přechod do systémového – pouze přerušení vč. programového

## Instrukce – x86/AMD64

## Ulož hodnotu

## AT&amp;T

`movq zdroj 64b, cíl`

`movl zdroj 32b, cíl`

`movw zdroj 16b, cíl`

`movb zdroj 8b, cíl`

registry se značí %ax

hodnoty \$, hex 0x

`movl $0xff, %ebx`

## Intel

`mov cíl, zdroj`

pouze ax

číslo, hex postfix h

`mov ebx, 0ffh`



## Instrukce – x86/AMD64

Ulož hodnotu - odkaz do paměti

AT&T

```
movl (%ecx),%eax
movl 3(%ebx), %eax
movl (%ebx, %ecx, 0x2), %eax
movl -0x20(%ebx, %ecx, 0x4), %eax
```

Intel

```
mov eax, [ecx]
mov eax, [ebx+3]
mov eax, [ebx+ecx*2h]
mov eax, [ebx+ecx*4h-20h]
```

odkaz má 4 složky základ+index\*velikost+posun

pole struktur o velikosti velikost, základ je ukazatel na první prvek,

index říká, který prvek chceme a posun, kterou položku uvnitř struktury potřebujeme.

není potřeba použít všechny 4 složky

## Instrukce – x86/AMD64

## Aritmetika - AT&amp;T syntax

operace co, k čemu

addq \$0x05,%rax	rax = rax + 5
subl -4(%ebp), %eax	eax = eax - mem(ebp-4)
subl %eax, -4(%ebp)	mem(ebp-4) = mem(ebp-4)-eax
andX	bitový and - argumenty typu X - b, w, l, q
orX	bitový or
xorX	bitový xor (nejrychlejší vynulování registru)
mulX	násobení čísel bez znamének
divX	dělení čísel bez znamének
imulX	násobení čísel se znaménky
idivX	dělení čísel se znaménky

## Instrukce – x86/AMD64

## Aritmetika s jedním operandem - AT&amp;T syntax

operace s cím

<code>incl %eax</code>	<code>eax = eax + 1</code>
<code>decw (%ebx)</code>	<code>mem(ebx) = mem(ebx)-1</code>
<code>shlb \$3, %al</code>	<code>al = al«3</code>
<code>shrb \$1, %bl</code>	<code>bl=11000000, po bl=01100000</code>
<code>sarb \$1, %bl</code>	<code>bl=11000000, po bl=11100000</code>
<code>rorx, rolx</code>	bitová rotace doprava a doleva
<code>rcrx, rcl</code>	bitova rotace – pres C - carry flag

## Práce se zásobníkem

<code>pushl %eax</code>	ulož na zásobník obsah eax 32 bitů
<code>popw %bx</code>	vyber ze zásobníku 2 bajty do bx
<code>pushf/popf</code>	ulož/vyber register EFLAGS
<code>pusha/popa</code>	ulož/vyber všechny už. registry

Při každém pop se zvětšuje registr SP o velikost operandu, při push se SP zmenšuje.

## Instrukce – x86/AMD64

## Podmíněné skoky

test a1, a2    tmp = a1 AND a2, Z tmp=0, C tmp<0

cmp a1, a2    tmp = a1-a2, Z tmp=0, C tmp<0

pak lze použít

## následující skoky

jmp kam        nepodmíněný skok, vlastně %eip=kam

je kam         jmp equal - skoč při rovnosti

jne kam        jmp not equal - skoč při nerovnosti

jg/ja kam      jmp greater – skoč pokud je a1 > a2 (sign/unsig)

jge/jae kam    skoč pokud je a1 >= a2 (sign/unsig)

jl/jb kam      jmp less – skoč pokud je a1 < a2 (sign/unsig)

jle/jbe kam    skoč pokud je a1 <= a2 (sign/unsig)

jz/jnz kam     skoč pokud je Z=1/0

jo/jno kam     skoč pokud je O (overflow) = 1/0

## Instrukce – x86/AMD64

## Volání funkce

```
call adr   vlastně push %eip, jmp adr
ret       vlastně pop %eip
```

## Lokální proměnné ve funkci – příklad implementace

```
push %ebp      ; Ulozime hodnotu EBP do zasobniku
mov %esp, %ebp ; Zkopirujeme hodnotu registru ESP to EBP
sub $12, %esp ; Snizime ukazatel zasobniku o 3x4 bajty
```

První proměnná bude na adrese  $-4(\%ebp)$ , druhá  $-8(\%ebp)$

První parametr bude na adrese  $8(\%ebp)$ , další  $12(\%ebp)$

```
mov %ebp, %esp ; Vratime ukazatel zpet na puvodni pozici.
pop %ebp      ; Obnovime puvodni hodnotu registru EBP
ret           ; Navrat z funkce
```

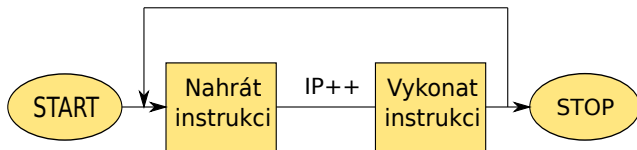
# Instrukce – x86/AMD64

## Složitost assembleru

- Algoritmus se dá přeložit různými způsoby do assembleru
- Různé způsoby pracují různě rychle a jsou rozdílně dlouhé a rozdílně přehledné
- `xor %ebx, %ebx` je to samé jako `mov $0, %ebx`
- `lea adresa, registr` – load effective address – nastaví hodnotu ukazatele do zadaného registru
- `lea -12(%esp), %esp` je to samé jako `sub $12, %esp`
- `lea` je výhodnější vzhledem k předzpracování instrukcí, nezatěžuje ALU jednotku (ovšem třeba Atom má zpracování adr. pomalejší než ALU).

# Pracovní krok procesoru

- Procesor pracuje v krocích.
- Jeden krok obsahuje fáze:
  - Přípravná fáze (fetch cycle)
    - nahrává do procesoru instrukci podle IP a umístí její kód do IR
    - na jejím konci se inkrementuje IP
  - Výkonná fáze (execute cycle)
    - vlastní provedení instrukce
    - může se dále obracet (i několikrát) k paměti



```

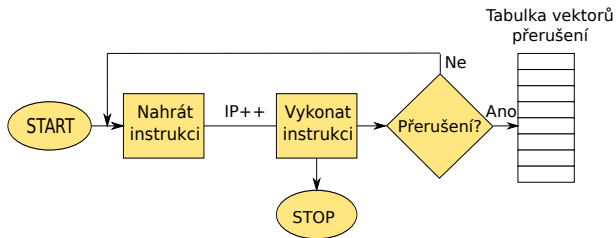
loop: FETCH;      /* z adresy IP nahraj data do IR */
Increment(IP);
EXECUTE;        /* provede operaci uloženou v IR */
end loop
  
```

# Přerušení

- Přerušení normální posloupnosti provádění instrukcí
  - cílem je zlepšení účinnosti práce systému
  - je potřeba provést jinou posloupnost příkazů jako reakci na nějakou „neobvyklou“ externí událost
  - přerušující událost způsobí, že se pozastaví běh procesu v CPU takovým způsobem, aby ho bylo možné později znovu obnovit, aniž by to přerušovaný proces „poznal“
- Souběh I/O operace
  - přerušení umožní, aby CPU prováděla jiné akce než instrukce programu čekajícího na konec I/O operace
  - činnost CPU se později přeruší iniciativou „I/O modulu“
  - CPU předá řízení na obslužnou rutinu přerušení (Interrupt Service Routine) – standardní součást OS
- CPU testuje nutnost věnovat se obsluze přerušení alespoň po dokončení každé instrukci
  - existují výjimky (např. „blokové instrukce“ Intel)



# Pracovní krok s přerušením



```

INTF=False; /* vymaz preruseni */
loop: FETCH;
    Increment(IP);
    EXECUTE;
    IF povoleno preruseni && INTF then
        Uloz FLAGS na zasobnik
        Uloz IP na zasobnik
        FLAGS nastav CPL0 a zakaz preruseni
        IP = vektoru preruseni
    end loop
  
```

# Druhy přerušení

- Každé přerušení má své číslo odkazující do tabulky přerušení, kde je tzv. vektor přerušení
- Když přerušení nastane vektor přerušení obsahuje adresu programu, který se vykoná
- Příklady přerušení
  - Chyba dělení (dělení nulou) - 0
  - Nemaskovatelné přerušení 2
  - Program break 3
  - Chybná instrukce 6
  - Chybějící segment 11
  - Chyba segmentu zásobníku 12
  - Chyba ochrany 13
  - Chyba stránky 14
  - uživatelské přerušení 32-255
  - časovač 32

# Druhy přerušení

- Přerušení je speciálním případem výjimečné situace.
  - Synchronní přerušení (s během programu) - periférie, maskovatelné a nemaskovatelné
  - Programové (naprogramované) - speciální instrukce (INT, TRAP)
- Generované kontrolními obvody počítače:
  - dělení nulou, pokus o vykonání nelegální či neznámé instrukce
  - neoprávněný pokus o přístup k paměťové lokaci (narušení ochrany paměti, virtuální paměť)
- Asynchronní (přicházející zvenčí – klasické přerušení)
  - I/O, časovač, hardwarové problémy (např. výpadek napájení ...)
- Kdy se na výjimečné situace reaguje?
  - Standardní přerušení: po dokončení instrukce během níž vznikl požadavek
  - Výjimka vysoké úrovně: během provádění instrukce (po dokončení některé fáze provádění instrukce) – instrukci nelze dokončit – neznámá instrukce, dělení nulou
  - Kritická výjimka: nelze dokončit ani cyklus přenosu dat a je nutno reagovat okamžitě – narušení ochrany paměti

# Obsluha přerušení

- Žádost se vyhodnotí na přípustnost (priority přerušení)
- Procesor přejde do zvláštního cyklu
  - FLAGS se uloží na zásobník (Výsledek aritmetických operací se mění téměř každou instrukcí).
  - Na zásobník se uloží i čítač instrukcí IP (návrátová hodnota z přerušení).
  - Do FLAGS se vygeneruje nové stavové slovo s nastaveným CPL0. Nyní je CPU v privilegovaném režimu
  - IP se nahradí hodnotou z vektoru přerušení - skok na obsluhu přerušení
- Procesor přechází do normálního režimu práce a zpracovává obslužnou rutinu přerušení v privilegovaném módu
  - Obslužná rutina musí být transparentní, tj. programově se musí uložit všechny registry CPU, které obslužná rutina použije, a před návratem z přerušení se opět vše musí obnovit tak, aby přerušená posloupnost instrukcí nepoznala, že byla přerušena.
  - Obslužnou rutinu končí instrukce „návrat z přerušení“ IRET mající opačný efekt: z vrcholu zásobníku vezme položky, které umístí zpět do IP a FLAGS

# Více násobné přerušování

- Sekvenční zpracování
  - během obsluhy jednoho přerušování se další požadavky nepřijímají (pozdržují se)
  - jednoduché, ale nevhodné pro časově kritické akce
- Vnořené zpracování
  - prioritní mechanismus
  - přijímají se přerušování s prioritou striktně vyšší, než je priorita obsluhovaného přerušování
- Odložené zpracování
  - V přerušování se provede pouze nejnutnější obsluha zařízení, zbytek se provede v rámci OS
  - Neblokují se zbytečně další přerušování