B4B35OSY: Operační systémy

Lekce 1. Úvod do operačních systémů

Petr Štěpán
stepan@fel.cvut.cz

4. října, 2018

Outline

Obsah

B4B35OSY – Operační systémy

Přednášející:

Michal Sojka, CIIRC
Michal.Sojka@cvut.cz

Petr Štěpán, FEL, katedra kybernetiky
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
- Hodnocení:
  - Body ze cvičení
    - Úlohy celkem až 50 bodů
    - Aktivita při hodině max 10 bodů
  - Písemná zkouška max 30 bodů
  - Ústní část max 10 bodů – 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, . . .
    - Mnoho programátorských problémů lze na nižší úrovni vyřešit snadněji a efektivněji
  - Techniky užívané v OS lze uplatnit i v jiných oblastech
    - neobvyklé struktury dat, krizové rozhodování, problémy souběžnosti, správa zdrojů, ...
    - mnohdy aplikace technik z jiných disciplin (např. operační výzkum)
    - naopak techniky vyvinuté pro OS se uplatňují v jiných oblastech (např. při plánování aktivit v průmyslu)

Naučit se lépe programovat

- Programování má různé podoby/úrovně (měli byste se seznámit se všemi):
  - Integrace high-level knihoven (mnohé webové a mobilní aplikace)
  - Aplikační programování (AI, počítačové hry, . . .) – obsahují vlastní algoritmy
  - Nízko-úrovňové programování (OS, embedded systémy, . . .) – pomezí SW a HW

- *"You might not think that programmers are artists, but programming is an extremely creative profession. It's logic-based creativity."*

  —John Romero

Naučit se přehledně programovat

- *"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*

  —Martin Fowler

  - V jednoduchosti je krása
    - řešení úlohy továrna z loňského roku – na 200 ale i 2000 řádek kódu
  - V dnešní době je program způsob záznamu informací/znalostí
    - V jedné pekárně se porouchal stroj a museli povolat pekaře-důchodce, protože nikdo jiný neznal recept na chleba ani nerozuměl programu stroje.
  - Nepište kód pro počítač, ale pro lidi, aby byl srozumitelný a znalosti tam byly na první pohled viditelné

- *"Programming is the art of algorithm design and the craft of debugging errant code."*

  —Ellen Ullman

Naučit se komentovat program

- Používejte vhodně pojmenované proměnné a funkce
- Komentujte co má funkce dělat a naznačte jak to dělá
- Komentujte jen to, co nelze vyjádřit programovacím jazykem
  - i = 1; // do proměnné i přiřadíme hodnotu 1 – NE!

- *“The cleaner and nicer the program, the faster it’s going to run. And if it doesn’t, it’ll be easy to make it fast.”*

  —Joshua Bloch

- Opět – Nepište kód pro počítač, ale pro lidi, aby byl srozumitelný a znalosti tam byly na první pohled viditelné

Obsah

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
- Hloubkový 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/cmsc311/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 nepřijde
  - 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

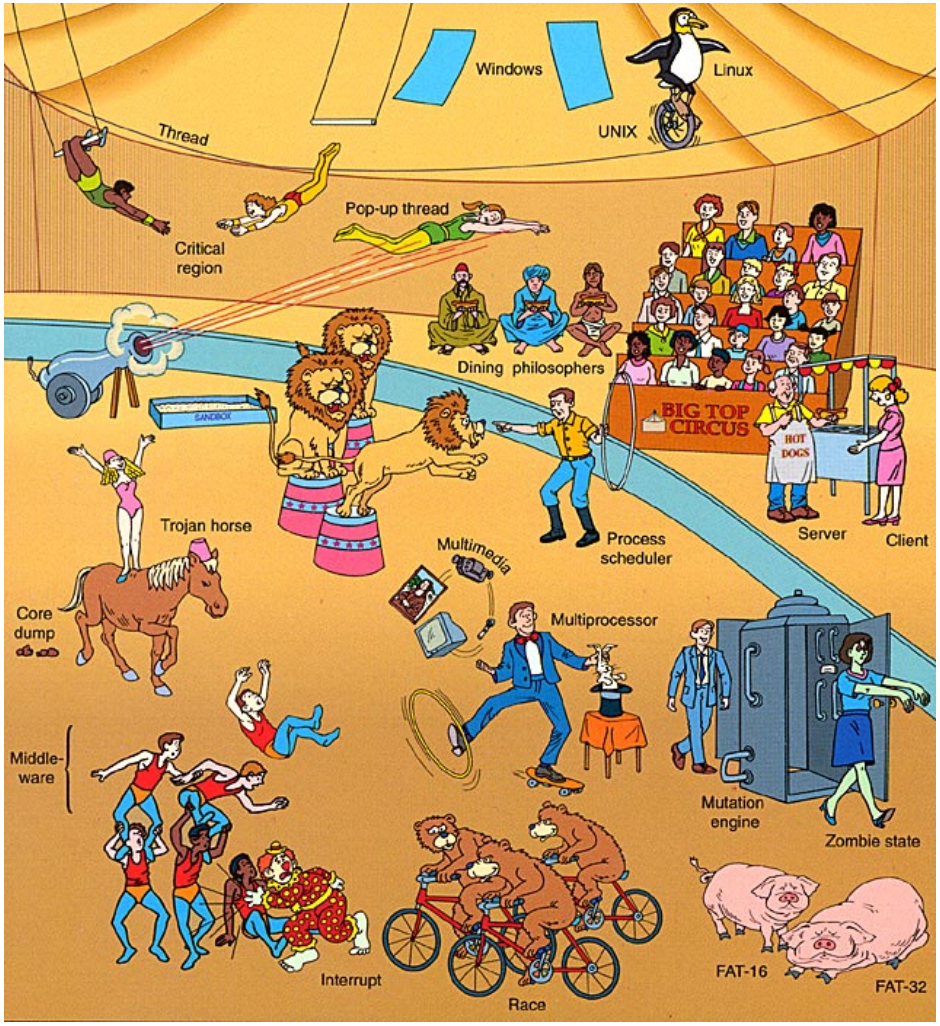
- Když Vaše představa neodpovídá tomu co slyšíte
  - Nebo když Vám chybí část „skládanky“
  - Na konci přednášky byste měli být schopni položit několik otázek, alespoň upřesňujících
    - „Myslím si, že říkáte ..(vlastními slovy).., je to tak?“
- Než se zeptáte, zkuste si odpovědět
  - Pokud nevíte,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

Obsah

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?



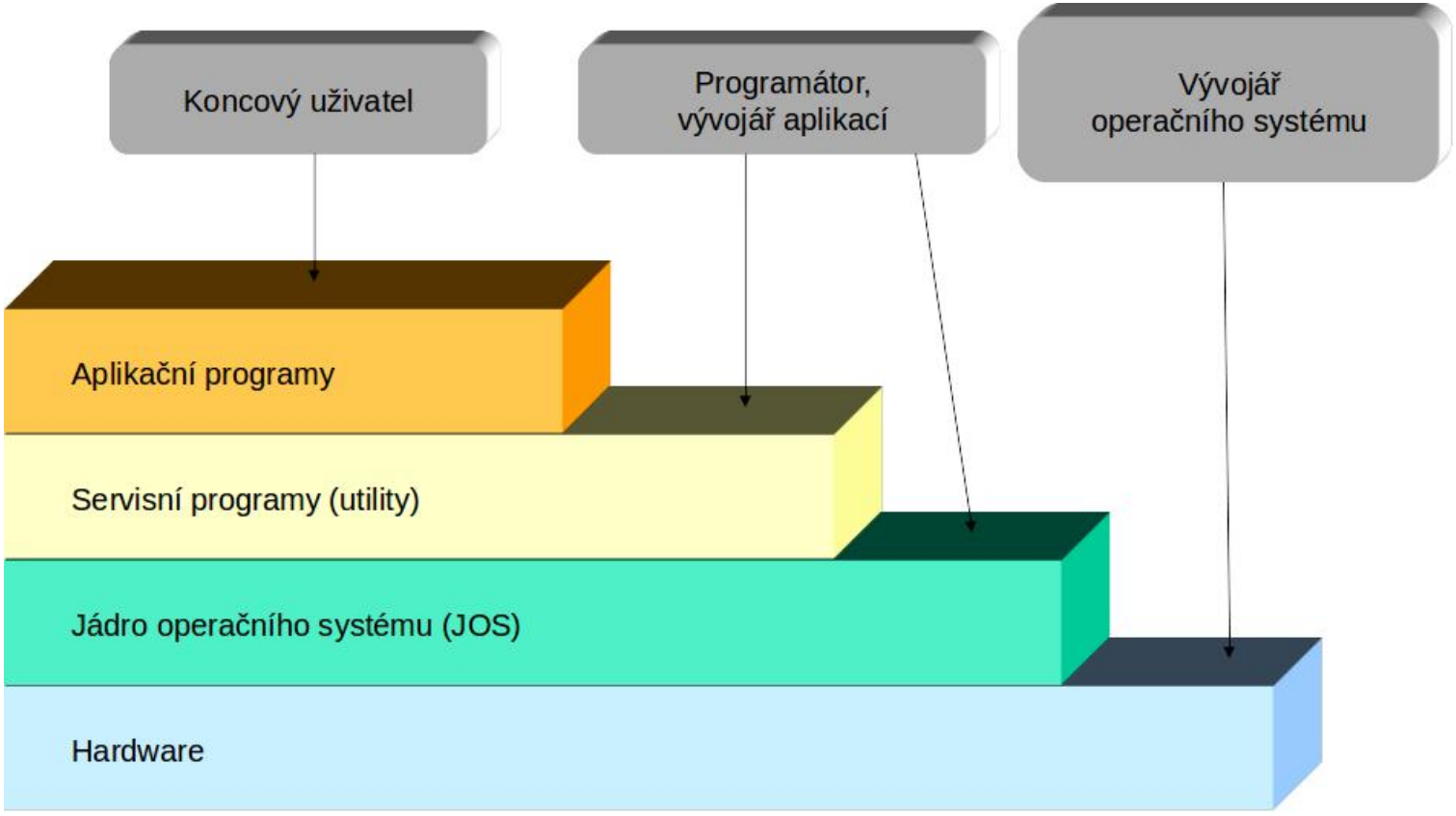
Co je operační systém

- 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 se snáze ovládá a programuje
- OS jako správce systémových prostředků
  - Každý program dostává prostředky v čase
  - Každý program dostává potřebný prostor na potřebných prostředcích
  - Prostředky jsou CPU, paměť, periférie

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

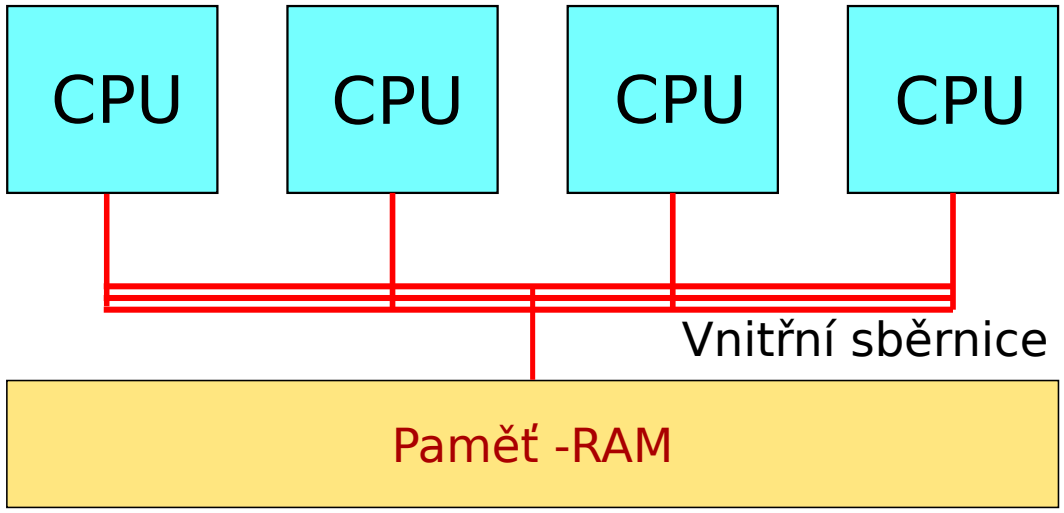


Různorodost OS

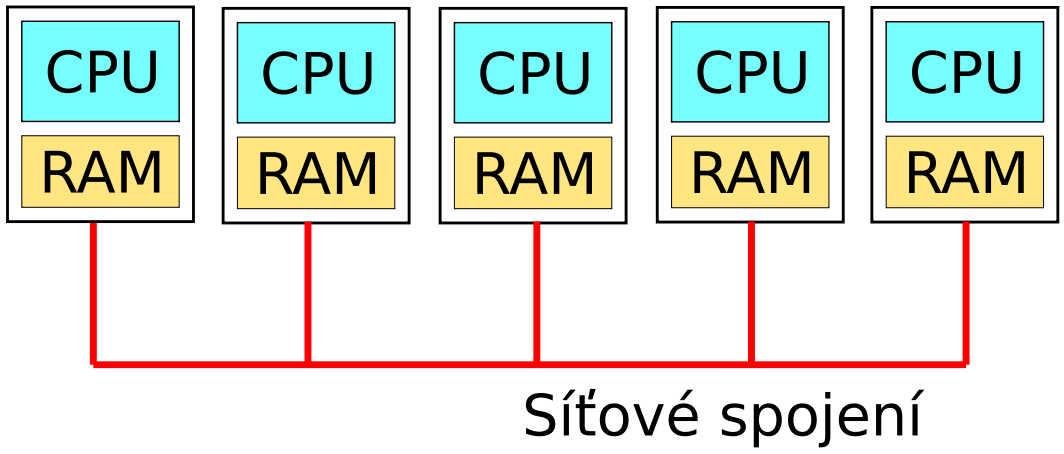
- OS „střediskových“ (mainframe) počítačů – dnes již historický pojem
- OS superpočítačů (5 mil. jader, 200 PFlops, 13 MW příkon)
- OS datových a síťových serverů
- OS osobních počítačů a pracovních stanic
- OS reálného času (Real-time OS – řízení letadel, vlaků, raket, družic, apod.)
- OS přenosných 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”

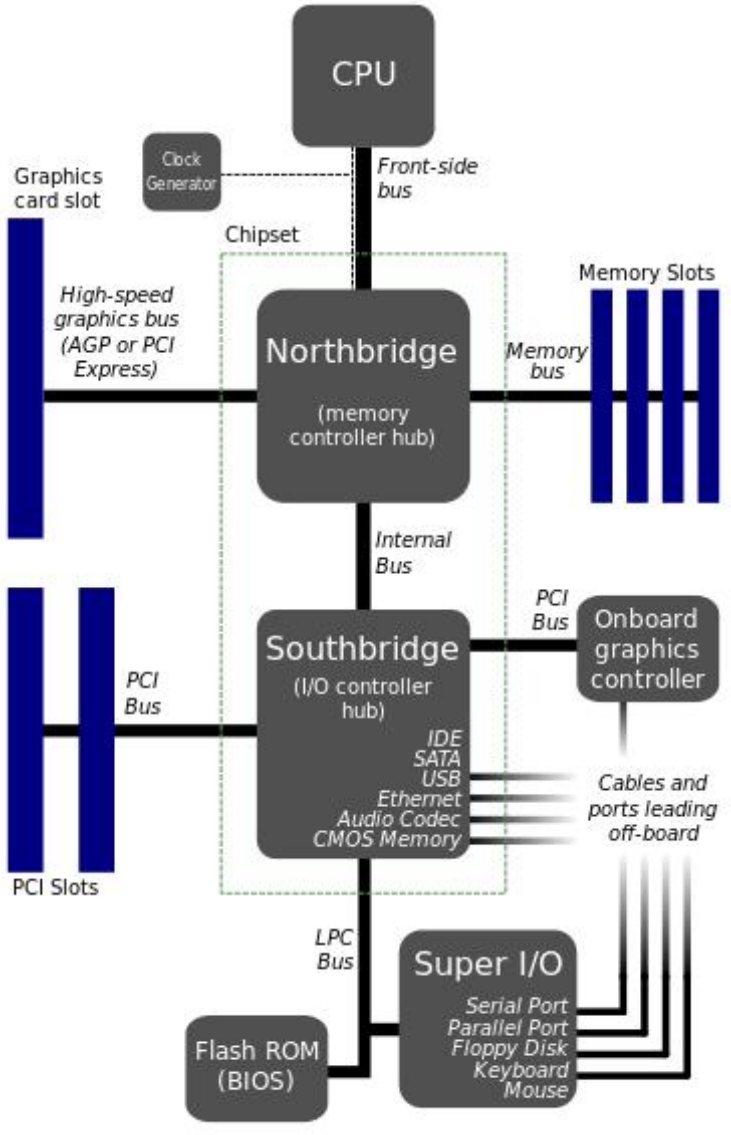
Více úloh současně – Multitasking

- 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

Obsah

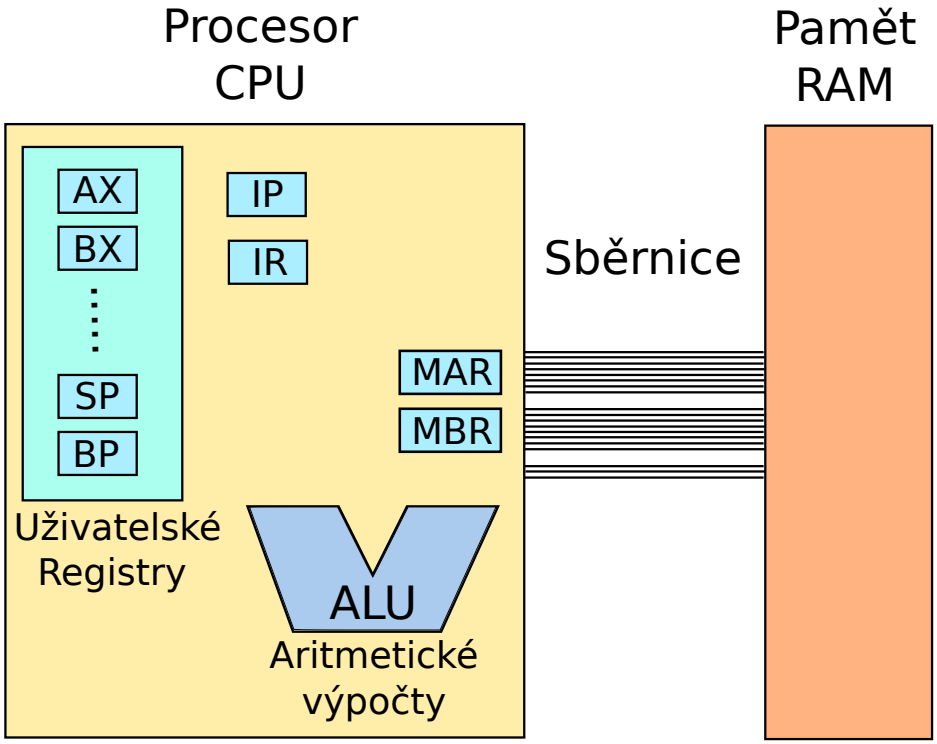
Osobní počítač

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



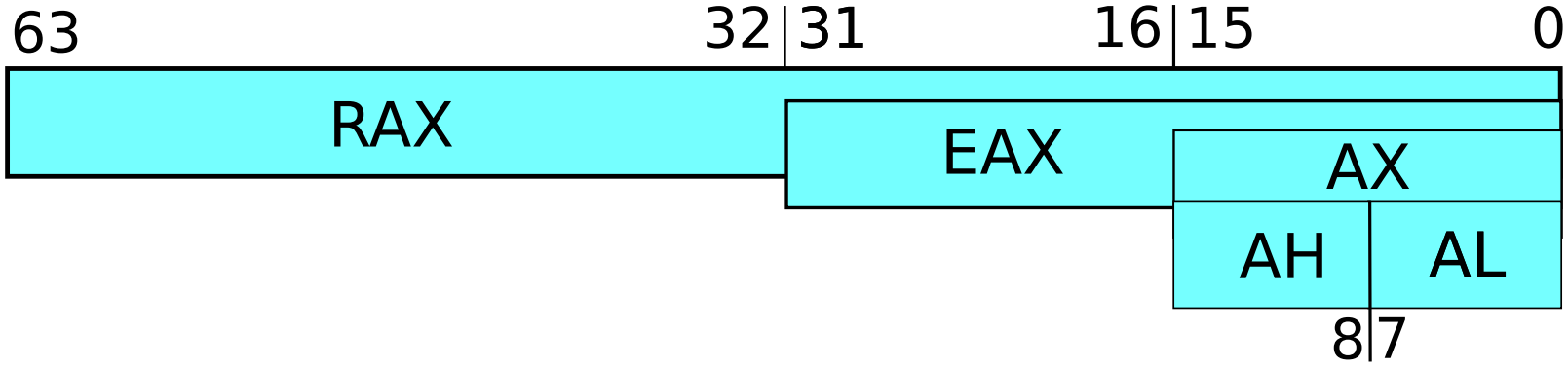
Procesor – CPU

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



Procesor – x86/AMD64

- Přehledný popis – `https://en.wikibooks.org/wiki/X86_Assembly`
- Všechny registry vzhledem ke zpětné kompatibilitě jsou 64/32/16/8 bitové

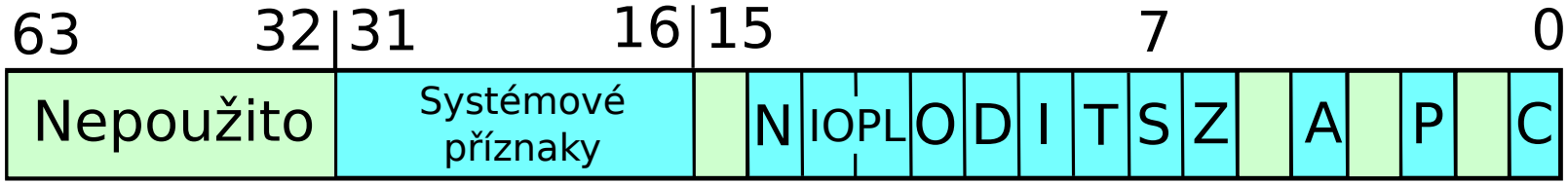


Řídicí a stavové registry

- 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.

Registr FLAGS

RFLAGS registr



C – Carry flag
P – Parity flag
Z – Zero flag
S – Sign flag
O – Overflow flag

I – Intterupt 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
  - CPL0[1] = privilegovaný (systémový) režim
    - procesor může vše, čeho je schopen
  - CPL3 = uživatelský (aplikační) 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

[1] Current privilege level

Procesor – 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 - detailněji dále
- 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

Instrukce – x86/AMD64

Instrukce “ulož hodnotu”
(běžně se používají dvě různé syntaxe pro zápis assembleru)

AT&T	**Intel**
`movq` zdroj 64b, cíl	`mov` cíl, zdroj
`movl` zdroj 32b, cíl	
`movw` zdroj 16b, cíl	
`movb` zdroj 8b, cíl	
registry se značí `%ax`	pouze `ax`
hodnoty $, hex 0x	číslo, hex postfix h
`movl $0xff, %ebx`	`mov ebx, 0ffh`

Instrukce – x86/AMD64

Ulož hodnotu na adresu (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&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&T syntax

operace s cim

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

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

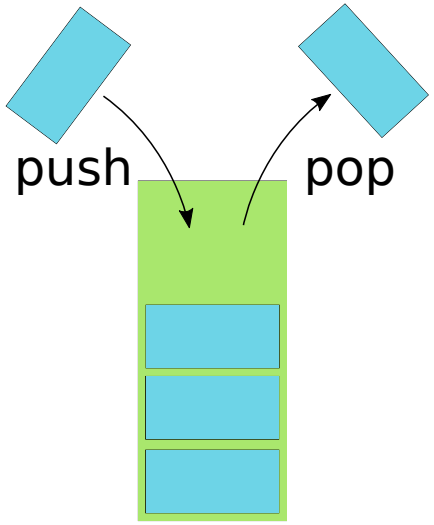
pak lze použít následující skoky

`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

Zásobník

Zásobník:

- obecná struktura LIFO
- operace push vloží data do zásobníku
- operace pop vybere data ze zásobníku



Implemetace:

- implementace registrem *SP* - ukazuje na vrchol zásobníku
- konvence - při každém pop se zvětšuje registr *SP* o velikost operandu, při push se *SP* zmenšuje.

`pushl %eax`	ulož eax na zásobník
`popw %bx`	vyber ze zásobníku 2 bajty do *bx*
`pushf/popf`	ulož/vyber register EFLAGS
`pusha/popa`	ulož/vyber všechny uživatelské registry

Funkce zásobníku

Zásobník:

- návratová hodnota po ukončení funkce
- lokální proměnné funkce
  - zásobník je vetšinou malý
  - omezená velikost lokálních proměnných
  - pozor při rekurzi - lépe se rekurzi vyhnout

Instrukce – x86/AMD64

Volání funkce

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

`ret` vlastně `pop %eip`

`leave` vlastně `mov %ebp, %esp`, `pop %ebp`

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)`, druha `-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
```

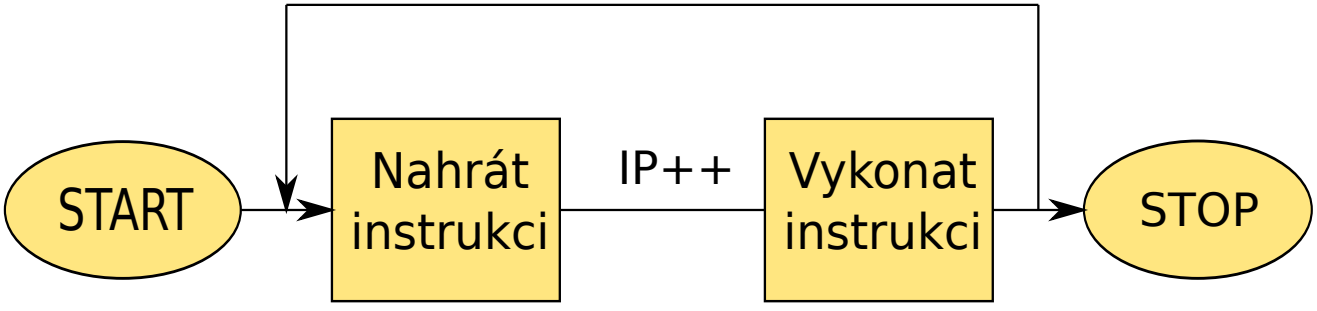
Instrukce – x86/AMD64

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

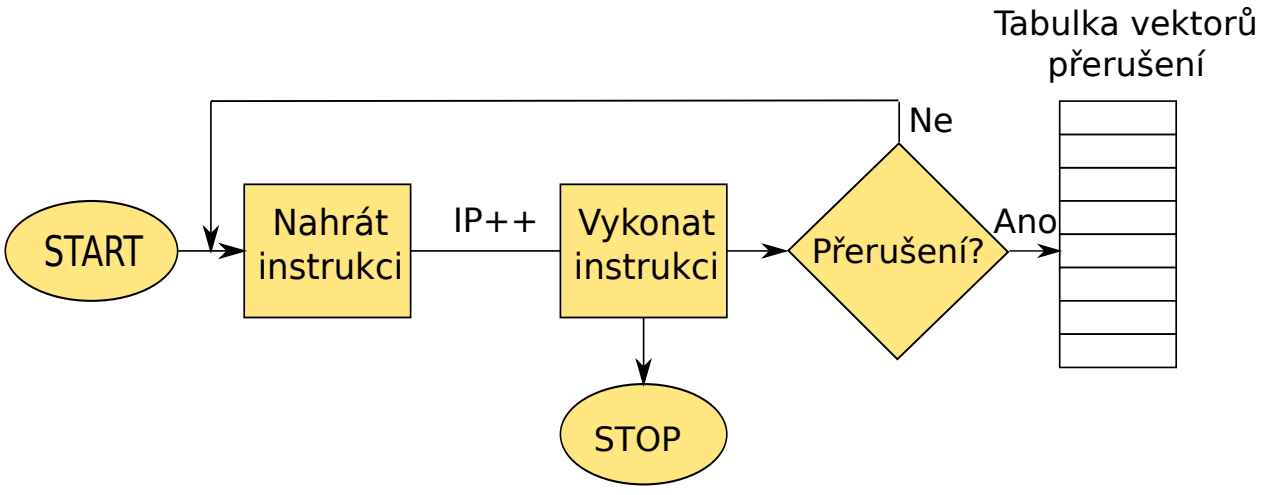



```
loop: FETCH;        /* z adresy IP nahraj data do IR */
      Increment(IP);
      EXECUTE;      /* provede operaci ulozenou v IR */
      end loop
```

Přerušení (výjimky)

- 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 aktuálně vykonávaného programu v CPU takovým způsobem, aby ho bylo možné později znovu obnovit, aniž by to přerušený program „poznal“
- Souběh I/O operace
  - přerušení umožní, aby po začátku přenosu dat z/do periférie CPU prováděla jiné akce než čekání 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 instrukce
  - 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
```

Obsluha přerušení

- Žádost se vyhodnotí na přípustnost (priority přerušení)
- Procesor přejde do zvláštního cyklu
  - FLAGS se uloží na zásobník (registr FLAGS se mění již při vstupu do přerušení a také většina instrukcí mění hodnotu FLAGS; je tedy nutné ho uložit co nejdříve).
  - Na zásobník se uloží i hodnota čítače instrukcí IP (návratová 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

Druhy přerušení (x86)

- Každé přerušení má své číslo odkazující do tabulky přerušení, kde je tzv. vektor přerušení
- Vektor přerušení obsahuje adresu programu, od které se začné vykonávat kód při výskytu daného přerušení
- Přerušení se dělí vzhledem k vykonávanému programu na synchronní a asynchronní

Synchronní přerušení

- Chyba dělení (dělení nulou) 0
- Program break 3
- Chybná instrukce 6
- Chybějící segment 11
- Chyba segmentu zásobníku 12
- Chyba ochrany 13
- Chyba stránky 14

Asynchronní přerušení

- Nemaskovatelné přerušení 2
- časovač 32
- uživatelské přerušení 32–255 (síťová karta, klávesnice, . . .)

Zdroje přerušení

- Vnitřní přerušení – problém při zpracování strojové instrukce
  - instrukce nebo data nejsou v paměti - chyba stránky, chyba segmentu
  - instrukci nelze provést - dělení nulou, ochrana paměti, nelegální instrukce
  - nutno reagovat okamžitě, nelze dokončit instrukci, někdy nelze ani načíst instrukci
- Vnější přerušení – vstupně/výstupní zařízení
  - asynchroní s během procesoru
  - signalizace potřeby reagovat na vstup/výstup
  - reakce po dokončení vykonávané instrukce
- Programové přerušení – strojová instrukce proveď přerušení
  - využívá se k ochraně jádra OS
  - obsluha přerušení může používat privilegované instrukce
  - lze spustit pouze kód připravený OS

Vícenásobné přerušení

- Sekvenční zpracování
  - během obsluhy jednoho přerušení se další požadavky nepřijímají (pozdržují se, IF bit v registru FLAGS)
  - jednoduché, ale nevhodné pro časově kritické akce
- Vnořené zpracování
  - prioritní mechanismus
  - přijímají se přerušení s prioritou striktně vyšší, než je priorita obsluhovaného přerušení
- Odložené zpracování
  - V přerušení se provede pouze nejnutnější obsluha zařízení, zbytek se provede později mimo přerušení (deffered jobs, workqueues, . . .)
  - Neblokují se zbytečně další přerušení