## Operační systémy

# Úvod do Operačních Systémů

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

# Organizační informace

- email: petr.krajca@upol.cz
- konzultační hodiny
  - čtvrtek 13:00 – 14:30,
  - pátek 9:45 – 11:15 (termíny dle rozvrhu)
- www: `http://phoenix.inf.upol.cz/~krajca/`
- slidy budou k dispozici online

# Průběh výuky

- přednášky
- úkoly související s probíranou tématikou
- není potřeba vyřešit zadané úkoly (určené k pochopení probírané látky)
- samostudium; do příště:
  - Keprt A. *Operační systémy. 2008*. Kapitoly 1–5.3.5, tj. strany 9–48.
  - Keprt A. *Assembler. 2008*. Kapitoly 1–5.1, tj. strany 6–53.
- skripta jsou na webu katedry (sekce studium)
- dotazy a konzultace

# Literatura


- Keprt A. *Operační systémy. 2008*
- Keprt A. *Assembler. 2008*
- Silberschatz A., Galvin P.B., Gagne G. *Operating System Concepts, 7th Edition.* John Wiley & sons, 2005. ISBN 0-471-69466-5.
- Tanenbaum A.S. *Modern Operating Systems, 2nd ed.* Prentice-Hall, 2001. ISBN 0-13-031358-0.
- Stallings, W. *Operating System Internals and Design Principles, Fifth Edition.* Prentice Hall, 2004. ISBN 0-13-127837-1.
- Solomon D.A., Russinovich M. E. *Windows Internals: Covering Windows Server 2008 and Windows Vista.* Microsoft Press, 2009. ISBN 0735625301.
- Jelínek L. *Jádro systému Linux: kompletní průvodce programátora.* Brno, Computer Press, 2008.

# Architektura počítače

## John von Neumannova architektura

- CPU (ALU, **řadič**)
- paměť **společná pro program i data** (vs. harvardská architektura)
- vstup/výstup
- sběrnice (řídící, adresní, datová)
- instrukce procesoru jsou zpracovávány v řadě za sebou (není-li uvedeno jinak)

## OS jako abstrakce HW

- vyvíjet software na míru jednoho HW náročné/neefektivní (obvykle); hardware je neuvěřitelně složitý
- operační systém + jazyky vyšší úrovně poskytují potřebnou abstrakci
- operační systém – rozhraní mezi HW a SW
- v konečném důsledku několik úrovní abstrakce

# Operační systém

## Vrstvy poč. systému

1. hardware
2. operační systém (OS)
3. standardní knihovna (libc, CRT)
4. systémové nástroje
5. aplikace

## Poznámky

- hranice mezi posledními třemi vrstvami nemusí být ostré
- operační systém zajišťuje správu zdrojů (sdílený přístup k paměti, zařízením, . . . )
- z časových důvodu se omezíme jen na OS pro PC, které jsou založené na architektuře Intel x86

# CPU (1/2)

## Obecná struktura CPU

- Aritmeticko-logická jednotka (ALU) – provádí výpočty
- řídící jednotka – řídí chod CPU
- registry – slouží k uchování právě zpracovávaných dat (násobně rychlejší přístup než do paměti); speciální registry obsluhující chod CPU: IP (instruction pointer), FLAGS, IR (instruction register), SP (stack pointer)

## Instrukční sada (ISA)

- sada instrukcí ovládající procesor (specifická pro daný CPU/rodinu CPU)
- instrukce a jejich operandy jsou reprezentovány jako čísla $\Longrightarrow$ strojový kód
- každá instrukce má obvykle 0 až 3 operandy (může to být registr, konstanta nebo místo v paměti)
- pro snazší porozumění se instrukce CPU zapisují v jazyce symbolických adres (assembleru)

# Výpočet faktoriálu: Intel x86

```
00000000 <main>:
 0:   8b 4c 24 04             mov    ecx,DWORD PTR [esp+0x4]
 4:   b8 01 00 00 00          mov    eax,0x1
 9:   83 f9 00                cmp    ecx,0x0
 c:   0f 8e 0a 00 00 00       jle    1c <main+0x1c>
12:   f7 e9                   imul   ecx
14:   83 e9 01                sub    ecx,0x1
17:   e9 ed ff ff ff          jmp    9 <main+0x9>
1c:   c3                      ret
```

# Výpočet faktoriálu: Sparc V8

```
00000000 <main>:
   0:   9d e3 bf 88     save  %sp, -120, %sp
   4:   a0 10 20 01     mov  1, %l0
   8:   80 a6 00 00     cmp  %i0, %g0
   c:   04 80 00 06     ble  24 <main+0x24>
  10:   01 00 00 00     nop
  14:   a0 5c 00 18     smul  %l0, %i0, %l0
  18:   b0 26 20 01     dec  %i0
  1c:   10 bf ff fb     b  8 <main+0x8>
  20:   01 00 00 00     nop
  24:   b0 10 00 10     mov  %l0, %i0
  28:   81 c7 e0 08     ret
  2c:   81 e8 20 00     restore
```

# CPU (2/2)

- instrukce jsou zpracovávány v několika krocích:

1. načtení instrukce do CPU (Fetch)
2. dekódování instrukce (Decode)
3. výpočet adres operandů
4. přesun operandů do CPU
5. provedení operace (Execute)
6. uložení výsledku (Write-back)

- pipelining – umožňuje zvýšit efektivitu CPU
- je potřeba zajistit správné pořadí operací
- procesor může mít víc jednotek např. pro výpočty (FPU, ALU)
- problém s podmíněnými skoky (branch prediction)

| Instr. No. | Pipeline Stage | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EX | MEM | WB | | |
| 2 | | IF | ID | EX | MEM | WB | |
| 3 | | | IF | ID | EX | MEM | WB |
| 4 | | | | IF | ID | EX | MEM |
| 5 | | | | | IF | ID | EX |
| Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Intel x86: Registry (1/2)

- registry jsou 32bitové
- obecně použitelné (i když existují určité konvence, jak by se měli používat)
  - EAX (Accumulator) – střadač pro násobení a dělení, vstupně-výstupní operace
  - EBX (Base) – nepřímá adresace paměti
  - ECX (Counter) – počítadlo při cyklech, posuvech a rotacích
  - EDX (Data)
- každý registr má svou spodní 16bitovou část reprezentovanou jako regist AX, BX, CX, DX
- tyto 16 bitové registr lze rozdělit na dvě 8bitové části reprezentované jako AH, AL, BH, BL, ...

# Intel x86: Registry (2/2)

**Další registry**

- EDI (Destination Index) – adresa cíle
- ESI (Source Index) – adresa zdroje
- EBP (Base Pointer) – adresace parametrů funkcí a lokálních proměnných
- ESP (Stack Pointer) – ukazatel na vrchol zásobníku (adresa vrcholu zásobníku)
- EIP (Instruction Pointer) – ukazatel na aktuální místo v programu, adresa instrukce následující za právě prováděnou instrukcí, není možné jej přímo měnit (jen patřičnými instrukcemi)
- EF(LAGS) – příznaky nastavené právě proběhlou instrukcí

- spodních 16 bitů těchto registrů lze adresovat pomocí registrů DI, SI, BP, SP, IP, F(LAGS); další dělení není možné
- ESI a EDI jde používat jako obecně použitelné
- změny v registrech EBP, ESP by měly být uvážené

# Intel x86: Operace (1/2)

- operandy instrukcí mohou být
  - `r` – registry
  - `m` – paměť
  - `i` – hodnoty
- paměť lze v jedné instrukci adresovat pouze jednou

```
MOV r/m, r/m/i        ; op1 := op2
ADD r/m, r/m/i        ; op1 := op1 + op2
SUB r/m, r/m/i        ; op1 := op1 - op2
NEG r/m               ; op1 := - op1
MUL r/m               ; EDX:EAX := EAX * op1
IMUL r, r/m           ; op1 := op1 * op2
IMUL r, r/m, i        ; op1 := op1 * op2 * op3

OR r/m, r/m/i         ; op1 := op1 | op2
AND r/m, r/m/i        ; op1 := op1 & op2
XOR r/m, r/m/i        ; op1 := op1 ^ op2
NOT r/m               ; op1 := ~op1
```

# Příklady

```
; do registru EAX ulozi obsah EBX
mov eax, ebx

; prevrati spodnich 16 bitu v registru ECX
xor ecx, 0x0000ffff

; pricte k registru cx hodnotu registru si
add cx, si

; takto nejde -- nesedi velikosti registru
add ecx, si

; vyneguje obsah registru edx
neg edx
```

# Intel x86: Operace (2/2)

```
SHL r/m, i    ; op1 := op1 << op2 (bez znaménková operace)
SAL r/m, i    ; op1 := op1 << op2 (znaménková operace)
SHR r/m, i    ; op1 := op1 >> op2 (bez znaménková operace)
SAR r/m, i    ; op1 := op1 >> op2 (znaménková operace)
ROL r/m, i    ; rotace bitů doleva
ROR r/m, i    ; rotace bitů doprava

INC r/m       ; op1 := op1 + 1
DEC r/m       ; op1 := op1 - 1
```

- místo okamžité hodnoty (konstanty) lze použít registr CL
- jednotlivé operace nastavují hodnoty jednotlivých bitů v registru EF
- mj. obsahuje příznaky
  - SF (sign flag) – podle toho, jestli výsledek je nezáporný (0), nebo záporný (1)
  - ZF (zero flag) – výsledek byl nula
  - CF (carry flag) – pokud při operaci došlo k přenosu mezi řády
  - OF (overflow flag) – příznak přetečení

# Intel x86: Běh programu, podmíněné skoky a porovnání

- program zpracovává jednu instrukci za druhou (pokud není uvedeno jinak) $\Longrightarrow$ skok
- nepodmíněný skok
  - operace `JMP r/m/i` – ekvivalent GOTO (použití při implementaci smyček)
- není přítomná operace ekvivalentní `if`
- podmíněný skok je operace ve tvaru Jcc, provede skok na místo v programu, pokud jsou nastaveny příslušné příznaky
- např. `JZ i` (provede skok, pokud výsledek předchozí operace byl nula)
- srovnání čísel jako rozdíl (operace `CMP r/m, r/m/i`, je jako `SUB`, ale neprovádí přiřazení
- `JE` skok při rovnosti, `JNE`, při nerovnosti
- skoky po porovnání znaménkových čísel `JG` ($>$), `JGE` ($\geq$), `JL` ($<$), `JLE` ($\leq$)
- skoky po porovnání bezznaménkových čísel `JA` ($>$), `JAE` ($\geq$), `JB` ($<$), `JBE` ($\leq$)

# Příklad

```
        ; porovnej eax s hodnotou 10
        cmp eax, 10

        ; pokud je eax vetsi nez 10
        ; provede skok na adresu foo
        jg foo

        ...
        ; jinak pokracuje timto kodem
        ...
foo:
        ...
```

# Adresace paměti

- přímá adresa – ukazuje na místo v paměti
- nepřímá adresa – před přečtením hodnoty se vypočítá z hodnot registrů podle vzorce:

$$adresa = posunuti + baze + index \times factor$$

- posunutí je konstanta
- báze a index jsou registry
- factor je číslo 1, 2, 4, nebo 8
- kteroukoliv část vzorce lze vypustit
- v assembleru se čtení/zápis do paměti zapisuje pomocí `[ ... ]`
- `mov eax, [ebx + 10]`
- `mov [eax + esi * 2 + 100], bx`
- je dobré doplnit velikost zapisovaných dat
- `mov eax, dword ptr [ebx + 10]`
- `mov word ptr [eax + esi * 2 + 100], bx`
- `mov word ptr [eax + esi * 2 + 100], 42` **!!!**
- při přístupu k proměnným ve VS (jsou adresy doplněny automaticky)

# Vztah adresace paměti procesoru a jazyka C

1. **Dereference**

```
mov eax, dword ptr [ebx]        ;; eax := *ebx
```

2. **Pole**

```
short *a = malloc(sizeof(short) * 10);
_asm {
    mov ebx, a
    mov ax, [ebx + esi * 2]     ;; ax := a[esi]
}
```

3. **Strukturované hodnoty**

```
struct foo { int x; int y; int z[10]; };
struct foo *a = malloc(sizeof(struct foo));
_asm {
    mov ebx, a
    mov [ebx], ecx                ;; a->x := ecx
    mov [ebx + 4], ecx            ;; a->y := ecx
    mov [ebx + esi * 4 + 8], ecx ;; a->z[esi] := ecx
}
```

# CISC a RISC

**Complex Instruction Set Computer**

- x86 má architekturu CISC
- složitější instrukce (dovedou víc věcí, adresovat paměť, atd.), náročné na zpracování
- snazší tvořit program (je kratší)
- menší počet registrů, instrukce nejsou ortogonální

**Reduced Instruction Set Computer**

- větší počet registrů, jednoduché instrukce, schopné zajistit stejnou funkcionalitu
- samostatné instrukce pro přístup k paměti
- jedna instrukce dělá víc věcí
- příklady pro SPARC (využívá registr g0, který je vždy nula)

```
subcc %r1, %r2, %r3     ; r3 := r1 - r2
subcc %r1, %r2, %g0     ; g0 := r1 - r2   (cmp r1, r2)
or %g0, 123, %r1        ; r1 := g0 | 123  (mov r1, 123)
```