

Operační systémy 1

# Řízení výpočtu a přístup k paměti

Petr Krajča



Katedra informatiky  
Univerzita Palackého v Olomouci

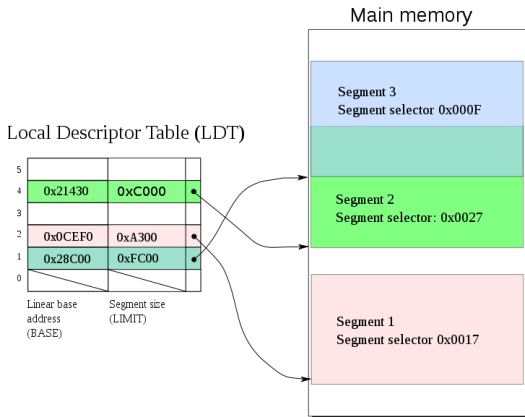
- lineární struktura s pevnou délkou a náhodným přístupem
- přímá adresa – ukazuje na pevně dané místo v paměti
- nepřímá adresa – před přečtením hodnoty se vypočítá z hodnot registrů podle vzorce:

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

- 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 ve tvaru:  
velikost PTR [ ... ]
- kde *velikost* může být (podle velikosti): BYTE, WORD, DWORD  
`mov dword ptr [ebx], eax`  
`add ax, word ptr [ebx + esi * 2 + 10]`
- pokud lze odvodit velikost dat z použitých registrů, je možné vypustit velikost PTR  
`mov [eax], ebx`  
`add ax, [ebx + esi * 2 + 10]`
- **Pozor!!!** `mov word ptr [eax + esi * 2 + 100], 42`
- při přístupu k proměnným ve VS jsou adresy doplněny automaticky
- `mov eax, a` je ve skutečnosti:  
`mov eax, dword ptr [ebp - n]`

- i386 má ve skutečnosti 48bitové adresy: selector (16 b) + offset (32 b)
- selector je určen pomocí segmentových registrů (CS, DS, SS, ES, FS, GS)
- segmentový registr většinou určen implicitně  $\Rightarrow$  pracuje se jen s offsetem
- lineární adresa =  $DT[\text{selector}] + \text{offset}$  (kde DT je LDT nebo GDT)





## Dereference

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

## Pole

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

## 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  
}
```

- adresa paměti `mem` je zarovnaná na `n` bytů, pokud je `mem` násobkem `n`
- z paměti procesor čte celé slovo (např. 32 bitů)  $\implies$  vhodné, aby čtená hodnota ležela na zarovnané paměti (rychlejší přístup, snazší implementace CPU)
- některé CPU neumožňují číst data z nezarovnané adresy (RISC), jiné penalizují zpomalením výpočtu
- hodnoty jsou zarovnávány na svou velikost, např.
  - `char` na 1B,
  - `short` na 2B,
  - `int` na 4B, atd.
- tzn. hodnoty typu `short` jsou v paměti vždy na adresách, které jsou násobky 2, hodnoty `int` na násobcích 4, atd.
- velikost struktur se obvykle zakrouhluje na 4 nebo 8B (směrem nahoru)



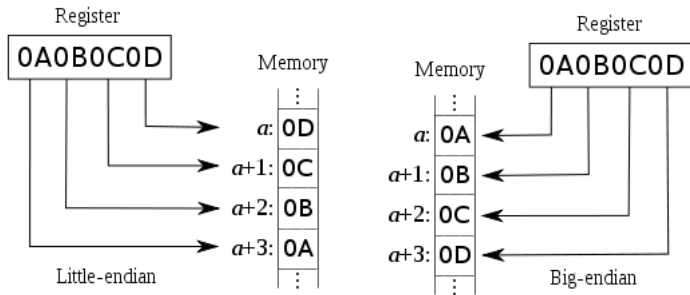
- Příklad:

```
struct foo {  
    char a;  
    /* mezera 3B */  
    int b;  
    char c;  
    /* mezera 1B */  
    short d;  
};
```

- toto je chování překladače; lze jej změnit (je-li to nutné)



- liší se mezi procesory  $\Rightarrow$  potřeba brát v úvahu při návrhu datových formátů a protokolů
- **little-endian**: hodnoty jsou zapisovány od nejméně významného bytu (x86, Amd64, Alpha, ...)
- **big-endian**: hodnoty jsou zapisovány od nejvýznamějšího bytu (SPARC, IBM POWER, Motorola 68000, ...)
- **bi-endian**: ARM, PowerPC, SparcV9, IA-64, ... (za určitých okolností lze přepínat)



- čísla jsou v doplňkovém kódu (zápornou hodnotu dostaneme tak, že provedeme inverzi bitů a přičteme 1)  $\implies$  snadná manipulace
- znaménkové a neznaménkové typy (`unsigned int` vs. `int`)!!!
- pokud se hodnota nevejde do rozsahu typu  $\implies$  přetečení/podtečení

```
char a = 127 + 1;           // => -128
unsigned char c = 255 + 1;  // => 0
char b = -10 - 120;         // => 126
```

## BCD (Binary Coded Decimal)

- čísla v desítkové soustavě 4b na cifru

## ASCII (American Standard Code for Information Interchange)

- způsob kódování znaků
- původně použité 7bitové hodnoty (později rozšířeny na 8 bitů)
- řídicí znaky (CR, LF, BELL, TAB, backspace, atd.)
- národní abecedy – horní polovina tabulky, kódování ISO-8859-X, Windows-125X, atd.

## Unicode

- znaková sada (definuje vazbu číslo  $\Leftrightarrow$  znak)
- několik tzv. rovin po 65535 znacích (v současnosti 110.000+ znaků)
- první rovina se nazývá základní (Basic Multilingual Plane, BMP) – znaky západních jazyků

## UCS (Universal Character Set)

- způsob kódování znaků Unicode
- pevně daná velikost
- UCS-2 – 16 bitů na znak, odpovídá základní rovině UNICODE
- UCS-4 – 32 bitů na znak, všechny znaky UNICODE

## UTF-8 (Unicode Transformation Format)

- kódování znaků s proměnlivou délkou
- zpětně kompatibilní s ASCII

bits	rozsah UNICODE	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
7	0000–007f	0xxxxxxx					
11	0080–07ff	110xxxxx	10xxxxxx				
16	0800–ffff	1110xxxx	10xxxxxx	10xxxxxx			
21	10000–1fffff	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	200000–3ffffff	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	4000000–7fffffff	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

## UTF-16

- proměnlivá délka kódování
- rozšiřuje UCS-2
- varianty UTF-16BE or UTF-16LE
- Byte Order Mark (BOM) – umožňuje určit typ kódování (0xffef nebo 0xfeff)

- jednotlivé operace nastavují hodnoty bitů v registru EF
- záleží na operaci, které příznaky nastavuje
- **příznaky pro řízení výpočtu**
  - SF (sign flag) – podle toho, jestli výsledek je nezáporný (0) nebo záporný (1)
  - ZF (zero flag) – výsledek byl nula
  - CF (carry flag) – výsledek je větší nebo menší než největší/nejmenší možné číslo
  - OF (overflow flag) – příznak přetečení znaménkové hodnoty mimo daný rozsah
- **další příznaky**
  - AF (auxiliary carry flag) – přenos ze čtvrtého do pátého bitu (BCD čísla)
  - PF (parity flag) – nastaven na jedna při sudé paritě (pouze dolních 8 bitů)
- **řídící příznaky**
  - TF (trap flag) – slouží ke krokování
  - DF (direction flag) – ovlivňuje chování instrukcí blokového přesunu
  - IOPL (I/O privilege level) – úroveň oprávnění (2 bity, pouze jádro)
  - IF (Interrupt enable flag) – možnost zablokovat některá přerušení (pouze jádro)

- program zpracovává jednu instrukci za druhou (pokud není uvedeno jinak)  $\implies$  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), dále `JNZ`, `JS`, `JNS`, ...

## Porovnávání čísel

- srovnání čísel jako rozdíl (operace `CMP r/m, r/m/i`, je jako `SUB`, ale neprovádí přiřazení)
- `JE` skok při rovnosti, `JNE`, při nerovnosti (v podstatě operace `JZ` a `JNZ`)
- a další operace

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	

- příklad použití
- podmíněné skoky po porovnání neznaménkových hodnot

instrukce	alt. jméno	příznaky	podmínka
JA	JNBE	$(CF \text{ and } ZF) = 0$	$A > B$
JAE	JNB	$CF = 0$	$A \geq B$
JB	JNAE	$CF = 1$	$A < B$
JBE	JNA	$(CF \text{ or } ZF) = 1$	$A \leq B$

- podmíněné skoky po porovnání znaménkových hodnot

instrukce	alt. jméno	příznaky	podmínka
JG	JNLE	$(SF = OF) \text{ \& } ZF = 0$	$A > B$
JGE	JNL	$(SF = OF)$	$A \geq B$
JL	JNLE	$(SF \neq OF)$	$A < B$
JLE	JNL	$(SF \neq OF) \text{ nebo } ZF = 1$	$A \leq B$

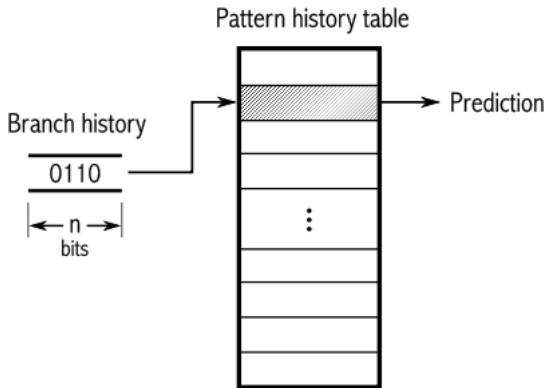


- pro snadnější implementaci cyklů byly zavedeny speciální operace
- JECXZ, JCXZ – provede skok pokud registr ECX/CX je nulový (není potřeba explicitně testovat ECX)
- LOOP – odečte jedničku od ECX a pokud v registru ECX není nula provede skok

## Poznámky

- uvádí se, že složené operace jsou pomalejší než jednotlivé kroky
- (obecně) podmíněné skoky zpomalují běh programu  $\implies$  zrušení výpočtu v pipeline
- procesory implementují různé heuristiky pro odhad jestli daný skok bude proveden
  - statický přístup (např. u skoků zpět se předpokládá, že budou provedeny)
  - dynamický přístup (na základě historie skoků se rozhodne)
  - nápověda poskytnutá programátorem (příznak v kódu)

- procesory používají kombinace výše zmíněných metod (hlavně dynamický odhad); různé metody
- čtyřstavové počítadlo:
- při každém průchodu procesor ukládá do Branch Prediction Buffer (2b příznak jestli byl skok proveden nebo ne) a postupně přechází mezi čtyřmi stavy:
  - 11 – strongly taken
  - 10 – weakly taken
  - 01 – weakly not taken
  - 00 – strongly not taken
- až na stav 00 předpokládá, že skok bude proveden
- velikost BPB a počáteční stav počítadla se mezi procesory liší
- problém: pravidelné střídání úspěšnosti  $\implies$  dvouúrovňový odhad (vzor chování)



- pro každý vzor existuje odhad založený na výše zmíněném přístupu
- velikost vzoru závisí na procesoru
- globální vs. lokální tabulka

- procesor má vyčleněný úsek paměti pro zásobník (LIFO)  $\implies$  mezivýpočty, návratové adresy, lokální proměnné, ...
- procesory i386 mají jeden zásobník, který roste shora dolů
- registr ESP ukazuje na vrchol zásobníku (`mov eax, [esp]` načte hodnotu na vrcholu zásobníku)
- uložení/odebrání hodnot pomocí operací:

```
PUSH r/m/i          ;; sub esp, 4
                     ;; mov [esp], op1
```

```
POP r/m              ;; mov op, [esp]
                     ;; add esp, 4
```

- k volání podprogramu se používá operace CALL  $r/m/i \implies$  uloží na zásobník hodnotu registru IP a provede skok

```
push eip                ;; tato operace neexistuje  
jmp <addr>
```

- k návratu z funkce se používá operace RET  $\implies$  odebere hodnotu ze zásobníku a provede skok na adresu danou touto hodnotou
- použití zásobníku umožňuje rekurzi

## Volání funkcí

- předání parametrů
- vytvoření lokálních proměnných
- provedení funkce
- odstranění informací ze zásobníku
- návrat z funkce, předání výsledku

- způsob jakým jsou předávány argumenty funkcím jsou jen konvence (specifické pro překladač, i když často součástí specifikace ABI OS)
- předávání pomocí registrů (dohodnou se urč. registry), příp. zbývající argumenty se uloží na zásobník
- předávání argumentů čistě přes zásobník
- kdo odstraní předané argumenty ze zásobníku? (volaná funkce nebo volající?)
- Konvence C (cdecl)
  - argumenty jsou předané čistě přes zásobník
  - zprava doleva
  - argumenty ze zásobníku odstraňuje volající
  - umožňuje funkce s proměnlivým počtem parametrů
- Konvence Pascal (pascal)
  - argumenty jsou předané čistě přes zásobník
  - zleva doprava
  - argumenty ze zásobníku odstraňuje volaný
  - neumožňuje funkce s proměnlivým počtem parametrů



- Konvence fastcall (fastcall, msfastcall)
  - první dva parametry jsou předány pomocí ECX, EDX
  - zbylé argumenty jsou na zásobníku zprava doleva
  - argumenty ze zásobníku odstraňuje volaný
  - mírně komplikuje funkce s proměnlivým počtem parametrů
  - pod tímto jménem mohou existovat různé konvence
- návratová hodnota se na i386 obvykle předává pomocí registru EAX, příp. EDX:EAX