KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Přístup do paměti

KMI/OS1 Operační systémy I

Mgr. Markéta Trnečková, Ph.D.
www.marketa-trneckova.cz

Příklad z minula

Příklad

Napište v assembleru funkci `int mocnina(int n, unsigned int m)` vracející mocninu m^n.

Příklad z minula
Řešení

Příklad

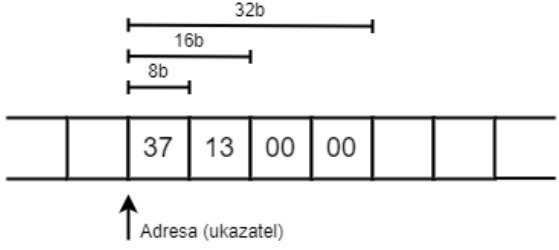
```
mocnina:
    mov rax, rdi
    mov rcx, rsi
    dec rcx
cyklus:
    cmp rcx, 0
    jle konec
    mul rdi
    dec rcx
    jmp cyklus
konec:
    ret
```

Přístup do paměti

Učební text ke cvičení:
`https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/tutorial05.pdf`

Paměť

- předpoklad: paměť je spojitý prostor
- na platformě AMD64 se skládá z 2^{64} paměťových buněk
- velikost buňky 1 byte
- adresa buňky: číslo od 0 do $2^{64} - 1$
- hodnoty jsou uloženy v po sobě jdoucích buňkách (např. 32 bitové hodnoty 4 paměťové buňky)
- adresa hodnoty – adresa první paměťové buňky
- z adresy můžeme hodnotu číst i na ní zapisovat



Adresace paměti

- většina instrukcí umožňuje, aby jeden z operandů odkazoval na místo v paměti
- přístup k paměti se zapisuje ve tvaru `velikost [adresa]`
- **velikost**:
  - `byte` (1 byte)
  - `word` (2 byte)
  - `dword` (4 byte)
  - `qword` (8 byte)
- adresa je číslo, používáme registry: `rax`, ..., `rdx`, `rsi`, `rdi`, `rbp`, `rsp`, `r8`, ..., `r15`

Adresace paměti

Příklad

```
; nacte do registru eax hodnotu z adresy 0x12345678
mov eax, dword [0x12345678]

; nacte do registru ax hodnotu z adresy, ktera je v rbx
mov ax, word [rbx]

; pricte k al hodnotu bytu na adrese rbx
add al, byte [rbx]

; nastavi byte na adrese dane registrem rbx na 0
mov byte [rbx], 0

; pricte k hodnote na adrese rbx hodnotu 2
add dword [rbx], 2
```

Adresace paměti

- pokud je z velikosti registru jasné s jak velkou hodnotou pracujeme, můžeme velikost vypustit
- to neplatí pro poslední 2 příklady
- **Proč?**

- adresu můžeme zadat i ve tvaru:
  `adresa = posunuti + baze + index` $\times$ `factor`
- `posunuti` je konstanta
- `baze`, index jsou registry
- `faktor` je číslo 1, 2, 4 nebo 8
- libovolnou část můžeme vypustit
- využití: přístupy k prvkům pole, práce se strukturami, práce s ukazateli

Použití společně s datovými typy

Ukazatele

- ukazatel = adresa v paměti (celé číslo)

Příklad

```
;; funkce zvysi hodnotu danou ukazatelem o 1
;; void incref(int *n);
incref:
   mov edx, dword [rdi] ; precteme hodnotu (rdi = ukazatel) do registru edx
   add edx, 1 ; zvysime hodnotu o 1
   mov dword [rdi], edx ; ulozime hodnotu zpet
   ret
```

- předáváme ukazatel na int (32 bitů) – ukazatel sám o sobě 64 bitů
- argument je předán registrem `rdi`, hodnotu uložíme do `edx`
- `edx` inkrementujeme a uložíme na adresu danou ukazatelem v registru `rdi`
- funkce nic nevrací, nemusíme nastavovat `eax`

Použití společně s datovými typy

Ukazatele

Příklad (Test funkce)

```
int a = 42;
int *p = &a;
printf("a = %i\n", a);
incref(p);
printf("a = %i\n", a);
```

Vyzkoušejte `funckce.asm` a `main.c`

Použití společně s datovými typy

Pole

- potřebujeme adresu prvního prvku (následující prvky jsou v paměti za ním)
- přístup do pole je pak shodný, jako v předchozím případě

Příklad (Zkuste vysvětlit)

```
;; Funkce secte count prvku v poli array.
;; int sum(int count, int *array);
sum:
    mov eax, 0 ; prubezny soucet
    mov ecx, 0 ; index aktualniho prvku
sum_loop:
    cmp edi, ecx ; test, zda jsme na konci pole
    je sum_done  ; ukonceni cyklu
    add eax, [rsi + rcx * 4] ; pricteni hodnoty do prubezneho souctu
    add ecx, 1 ; prechod na dalsi prvek
    jmp sum_loop
sum_done:
    ret ; vraceni vysledku (v eax)
```

Použití společně s datovými typy

Pole

Příklad (Test funkce)

Jak by vypadal test funkce v .c ?

Vyzkoušejte `funckce.asm` a `main.c`

Použití společně s datovými typy

Řetězce

- řetězec – pole 1 byte prvků, ukončené '\0' (0)

Příklad (Zkuste vysvětlit)

```
;; Replika funkce strcpy ze standardni knihovny.
;; Funkce prekopiruje retezec src do pameti dane ukazatelem dst.
;; void my_strcpy(char *dst, char *src);
my_strcpy:
    mov al, byte [rsi] ; precteme jeden (prvni znak) ze zdrojoveho retezce
    mov byte [rdi], al ; ulozime tento znak do ciloveho retezce
    cmp al, 0 ; pokud je to znak \0, koncime
    je done
    add rdi, 1 ; posuneme se k dalsimu znaku
    add rsi, 1
    jmp my_strcpy ; skok na zacatek cyklu
done: ret    ; konec funkce
```

Použití společně s datovými typy

Řetězce

Příklad (Test funkce)

Jak by vypadal test funkce v .c ?

Vyzkoušejte `funckce.asm` a `main.c`

Použití společně s datovými typy

Strukturované datové typy

Příklad

Jak jsou položky strukturovaných datových typů uloženy v paměti v jazyce C?

Použití společně s datovými typy

Strukturované datové typy

- položky struktury jsou v jazyce C v paměti uloženy za sebou
- adresa struktury je stejná jako adresa jejího prvního prvku (první položky)

Příklad

```
struct foo {
    int bar;
    short baz;
}

struct foo qux;
```

- 6 bytů (4 pro `bar`, 2 pro `baz`)
- přístup k prvkům v C – tečkový operátor `qux.bar`

Použití společně s datovými typy

Strukturované datové typy

Příklad

```
struct foo {
    int bar;
    short baz;
}

struct foo qux;
```

- pro efektivnější práci se strukturami – zarovnání velikostí (na násobek 4 nebo 8 bytů)

Příklad

Pomocí operátoru `sizeof()` ověřte, že výše uvedená struktura zabírá v paměti 8 bytů.

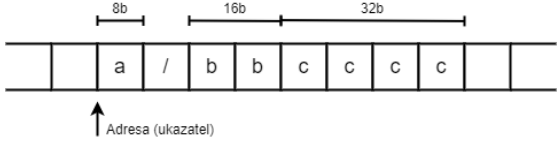
Použití společně s datovými typy

Strukturované datové typy

- k zarovnání dochází i u jednotlivých položek
  - 1 bytové hodnoty se zarovnávají na násobky 1(adresa)
  - 2 bytové hodnoty se zarovnávají na násobky 2
  - 4 bytové hodnoty se zarovnávají na násobky 4
- např. hodnoty typu `int` jsou ve struktuře uložené vždy na pozici, která je násobkem čtyř

Příklad

```
struct foo {
    char a;
    short b;
    int c;
};
```



Použití společně s datovými typy

Strukturované datové typy

Příklad

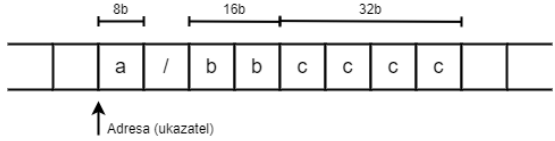
Když víme, jak jsou položky struktury uloženy v paměti, jak budeme k jednotlivým položkám přistupovat v assembleru?

Použití společně s datovými typy

Strukturované datové typy

Příklad

```
struct foo {
    char a;
    short b;
    int c;
};
```



Příklad

```
struct_foo:
    mov al, [rdi]       ; precte hodnotu clenu a do registru al
    mov cx, [rdi + 2]   ; precte hodnotu clenu b do registru cx
    mov edx, [rdi + 4]  ; precte hodnotu clenu c do registru edx
```

Přístup do paměti

Úkoly k procvičení

Všechny následující funkce naprogramujte v assembleru a voláním z jazyka C ověřte, že fungují dle očekávání.

1. Napište funkci `void swap(int *a, int *b)`, která prohodí hodnoty, které jsou dány ukazateli `a` a `b`.
2. Napište funkci `void division(unsigned int x, unsigned int y, unsigned int *result, unsigned int *remainder)`, která celočíselně vydělí hodnotu `x` hodnotou `y` a výsledek uloží na místo v paměti dané ukazatelem `result` a zbytek po dělení uloží do paměti dané ukazatelem `remainder`.
3. Napište funkci `void countdown(int *values)`, která do daného pole `values` uloží posloupnost 10, 9, 8, ..., 1 (v tomto pořadí).
4. Napište funkci `void nasobky(short *multiples, short n)`, která do pole `multiples` uloží prvních deset násobků čísla `n`.

Přístup do paměti

Úkoly k procvičení

5. Napište funkci `int minimum(int count, int *values)`, která vrací nejmenší prvek pole `values` obsahující `count` hodnot. Vyzkoušejte, že funkce funguje správně pro kladná i záporná čísla.
6. Napište funkci `unsigned int my_strlen(char *s)`, která se bude chovat jako funkce `strlen` ze standardní knihovny jazyka C.
7. Napište funkci `void my_strcat(char *dest, char *src)`, která se bude chovat jako funkce `strcat` ze standardní knihovny jazyka C.