KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Externí assembler, registry a základní aritmetické operace

KMI/OS1 Operační systémy I

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

Opakování

Praktická práce s assemblerem

- Pomocí direktivy `global` určíme, jaké funkce jsou implementovány v assembleru
- V sekci `.text` implementujeme jednotlivé funkce, které vyznačíme návěštím.
- Funkce vrací výsledek v `eax` (`rax`) a je ukončena instrukcí `ret`.
- V jazyce C definujeme prototypy daných funkcí a voláme je standardním způsobem.
- Při sestavování programu sloučíme kód v C a v assembleru.

- Uvažujeme-li unixový operační systém, pak můžeme funkcím předávat celočíselné argumenty po řade v registrech:
  `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
- **Obsah registrů:** `rbx`**,** `rsp`**,** `rbp`**,** `r12`**,** `r13`**,** `r14`**,** `r15`**, musí být na konci funkce stejný, jako na jejím začátku!!!**

Příklad

Příklad (Makefile)

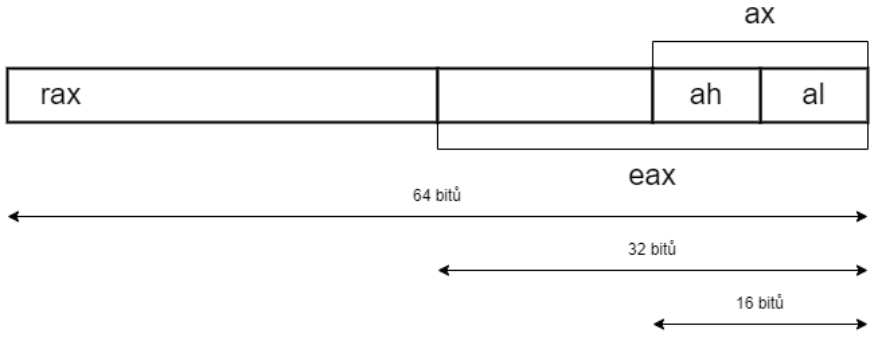
```
main: main.o funkce.o
	gcc -o main main.o funkce.o
main.o: main.c
	gcc -c main.c
funkce.o: funkce.asm
	nasm -f elf64 funkce.asm
```

Instrukční sada x86/AMD64

Registry

- **64bitové**: `rax`, `rbx`, `rcx`, `rdx`, `rsi`, `rdi`, `r8` až `r15`,
- **32bitové**: `eax`, `ebx`, `ecx`, `edx`, `esi`, `edi`, `r8d` až `r15d`,
- **16bitové**: `ax`, `bx`, `cx`, `dx`, `si`, `di`, `r8w` až `r15w`,
- **8bitové**: `ah`, `al`, `bh`, `bl`, `ch`, `cl`, `dh`, `dl`, `sil`, `dil`, `r8b` až `r15b`.

Vzájemně si odpovídající registry sdílí stejnou paměť



Instrukční sada x86/AMD64

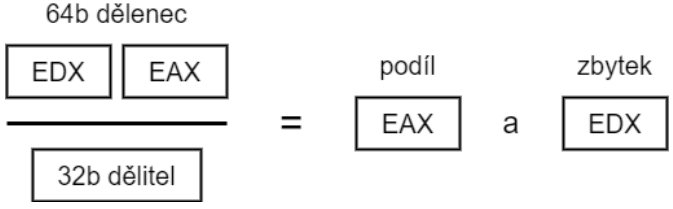
Násobení

- **Neznamenkové násobení** `edx:eax = eax * op1`:
- **Neznamenkové násobení** `rdx:rax = rax * op1`:
  - `mul r/m`
- **Znaménkové násobení** `op1 = op1 * op2`:
  - `imul r, r/m`
- **Znaménkové násobení** `op1 = op2 * op3`:
  - `imul r, r/m, i`

Instrukční sada x86/AMD64

Dělení

- **Neznaménkové dělení** `eax = edx:eax / op1; edx = edx:eax % op1`:
- **Neznaménkové dělení** `rax = rdx:rax / op1; rdx = rdx:rax % op1`:
  - `div r/m`
- **Znaménkové dělení** `eax = edx:eax / op1; edx = edx:eax % op1`:
- **Znaménkové dělení** `rax = rdx:rax / op1; rdx = rdx:rax % op1`:
  - `idiv r/m`



Řízení výpočtu

Učební text ke cvičení:
`https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/tutorial04.pdf`

Řízení běhu programu

- sekvence příkazů
- vyšší programovací jazyky nabízí:
  - cykly
  - větvení (podmínky)
- v assembleru lze řešit pomocí skoků

Řízení běhu programu
Skoky

- skok – pokračování výpočtu od zadané adresy
- Program je v paměti uložen jako data
- Každá instrukce na adrese
- Registr `rip` ukazuje na adresu instrukce, která má být provedena jako další
- Skok – změna registru `rip`
- `rip` je řídící registr, je možné ho měnit jen určitými instrukcemi
- **Typy skoků**
  - nepodmíněné
  - podmíněné

Nepodmíněné skoky

- ke skoku dojde vždy
- `jmp r/m/i`
- jeden operand = adresa skoku
- nejčastěji konstanta
- fyzické adresy neznáme, používáme **návěští** (label)
- návěští = symbolické pojmenování adresy
  `identifikator:`

Nepodmíněné skoky
Příklad

Příklad (Zkuste vysvětlit následující kód)

```
    mov eax, 0x42
foo:
    inc eax
    jmp foo
```

Nepodmíněné skoky
Příklad

Příklad

```
    mov eax, 0x42
foo:
    inc eax
    jmp foo
```

- program v nekonečné smyčce inkrementuje hodnotu registru `eax`

Nepodmíněné skoky

- adresa může být zadána relativně
- `jmp 0x12`
- skok na adresu `rip + 0x12`

Nepodmíněné skoky

Další poznámky

- Pro řešení `switch` operace `jmp qword [ebx]`
  Z adresy uložené v `ebx` přečteme adresu, kam se má skok provést a pak ho provedeme
- Můžeme skočit vlastně kamkoliv
- Vhodné používat s rozmyslem – špatně čitelný kód, obsahuje chyby

`qword` – 64-bit unsigned integer

Podmíněné skoky

- k jejich provedení dojde jen při splnění podmínky
- v jiném případě program pokračuje další instrukcí
- podmínky jsou určeny registrem `rf` (obsahující jednobitové příznaky):
  - `ZF` (zero flag) – výsledek byl nula,
  - `SF` (sign flag) – výsledek je nezáporný (0) nebo záporný (1),
  - `CF` (carry flag) – výsledek je větší nebo menší než největší/nejmenší možné číslo,
  - `OF` (owerflow flag) – příznak přetečení znaménkové hodnoty mimo daný rozsah.
- s příznaky pracujeme pomocí operací
  - je nastaven (`jz`, `js`, `jc`, `jo`)
  - není nastaven (`jnz`, `jns`, `jnc`, `jno`)

Podmíněné skoky
Příklad

Příklad (Zkuste vysvětlit následující kód)

```
    sub eax, 1
    jz foo
    inc ebx
foo:
```

Podmíněné skoky

Příklad

Příklad (Zkuste vysvětlit následující kód)

```
    sub eax, 1
    jz foo
    inc ebx
foo:
```

- odečítáme 1 od `eax`
- operace nastaví příznaky v registru `rf`
- pomocí `jz` zjišťujeme, zda je nastaven `ZF` (výsledek operace byl 0)
- pokud ano, provede se skok na `foo`
- jinak inkrementujeme `ebx`

Podmíněné skoky

- operace pro porovnání dvou hodnot
  `cmp r/m, r/m/i`
- porovnání = odečtení hodnot a nastavení `rf`

Příklad

Jak by vypadal podmíněný skok? Například chceme zjistit zda je hodnota větší než 2.

Podmíněné skoky

Příklad (Jak by vypadal podmíněný skok? Například chceme zjistit zda je hodnota větší než 2.)

```
    cmp eax, 2
    js mensi
vetsi:
    mov eax, 1
    ret
mensi:
    mov eax, -1
    ret
```

Podmíněné skoky

- operace pro **znaménkové porovnání**
  - `jg` (`jnle`) : $A > B$
    `(SF = OF) & (ZF = 0)`
  - `jge` (`jnl`) : $A \geq B$
    `(SF = OF)`
  - `jl` (`jnge`) : $A < B$
    `(SF` $\neq$ `OF)`
  - `jle` (`jng`) : $A \leq B$
    `(SF` $\neq$ `OF) nebo (ZF = 1)`

- operace pro **neznaménkové porovnání**
  - `ja` (`jnbe`) : $A > B$
    `(CF nebo ZF) = 0`
  - `jae` (`jnb`) : $A \geq B$
    `(CF = 0)`
  - `jb` (`jnae`) : $A < B$
    `(CF = 1)`
  - `jbe` (`jna`) : $A \leq B$
    `(CF nebo ZF) = 1`

`j` – jump, `l` – less, `g` – greater, `e` – equal, `a` – above, `b` – below

Podmíněné skoky
Příklad

Příklad (Výpočet absolutní hodnoty)

```
absi:
    mov eax, edi; precteni argumentu
    cmp eax, 0  ; porovnani s 0
    jge konec   ; skok na konec funkce
    neg eax     ; otoceni znameka
konec:
    ret         ; navrat z funkce
```

Podmíněné skoky
Příklad

Příklad

Napište funkci pro výpočet faktorialu.

Podmíněné skoky
Příklad

Příklad (Výpočet faktorialu)

```
fact:
    mov ecx, edi ; ecx vstupni argument (n)
    mov eax, 0x1 ; eax vysledna hodnota tj. n * (n- 1) * (n- 2) * ... * 1
fact_loop:
    cmp ecx, 0x0
    jle konec ; narazili jsme na konec, ukoncime funkci
    imul ecx  ; vynasob eax hodnotou n
    sub ecx, 0x1 ; opakuj pro n- 1
    jmp fact_loop
konec: ret
```

Podmíněné skoky
Příklad

Příklad (Výpočet faktorialu)

```
fact:
    mov ecx, edi ; ecx vstupni argument (n)
    mov eax, 0x1 ; eax vysledna hodnota tj. n * (n- 1) * (n- 2) * ... * 1
fact_loop:
    cmp ecx, 0x0
    jle konec ; narazili jsme na konec, ukoncime funkci
    imul ecx  ; vynasob eax hodnotou n
    sub ecx, 0x1 ; opakuj pro n- 1
    jmp fact_loop
konec: ret
```

Podmíněné skoky

Poznámka na závěr

- Podmíněné skoky ovlivňují rychlost zpracování programu
- Pokud možno, snažte se počet skoků minimalizovat

Příklad

Je možné následující kód přepsat tak, aby obsahovala jen jeden skok?

```
    cmp eax, 0
    jge foo
    jmp bar
foo:
    ; nejaky kod
bar:
```

Podmíněné skoky

Poznámka na závěr

Příklad (Je možné následující kód přepsat tak, aby obsahovala jen jeden skok?)

```
    cmp eax, 0
    jge foo
    jmp bar
foo:
    ; nejaky kod
bar:
```

Příklad (Řešení)

```
    cmp eax, 0
    jl bar
foo:
    ; nejaky kod
bar:
```

Řízení běhu programu

Úkoly k procvičení

1. Napište v assembleru funkci `int sgn(int i)`, která vrací hodnoty -1, 0, 1 v závislosti na tom, zda-li je hodnota `i` záporná, nulová nebo kladná.
2. Napište v assembleru funkci `char max2c(char a, char b)`, která vrací největší hodnotu. Vyzkoušejte, že funkce funguje správně pro kladné i záporné argumenty, i jejich kombinaci.
3. Napište v assembleru funkci `unsigned short min3us(unsigned short a, unsigned short b, unsigned short c)`, která vrací nejmenší hodnotu ze zadaných parametrů. Vyzkoušejte, že funkce funguje správně i pro hodnoty větší než 32768.
4. Napište v assembleru funkci `int kladne(int a, int b, int c)`, která vrací 1, pokud jsou všechny argumenty kladné, jinak 0.
5. Napište v assembleru funkci `int mocnina(int n, unsigned int m)` vracející mocninu m^n.
6. Do registrů `al`, `bl` vložte vhodné hodnoty, proveďte s nimi operace `add` a `sub` a pomocí instrukcí `jz`, `js`, `jc`, `jo` ověřte, zda byl nastavený příznak, nebo ne.