Operační systémy 1

3. Cvičení

Radek Janoštík

KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Úkoly z minula – jak to šlo?

- Kolik se vám toho podařilo udělat?
- Dalo něco zabrat?
- Nějaké dotazy?

Úkoly z minula – jak to šlo?

- Kolik se vám toho podařilo udělat?
- Dalo něco zabrat?
- Nějaké dotazy?
- Prezenčka (až přijdou všichni :))

Assembler

- Možnost vytvářet program na nejnižší úrovni
  - Imperativní způsob
  - Jednotlivé instrukce procesoru
- Dříve zcela běžné, i některé větší programy a operační systémy psány v assembleru
- Dnešní překladače na vysoké úrovni
- Dobré znát i dnes
  - Pochopení principů HW a OS
  - Vytváření kompilátorů
  - Optimalizace specifických úloh pomocí specifických instrukcí (multimédia, kryptografie, ...)

Způsoby tvorby kódu

1. Externí assembler
   - Napíšeme kód v assembleru a přeložíme assemblerem
   - Funkce v objektovém souboru můžeme volat z vyššího jazyka
   - Zkusili jsme si minule (`demo.asm`, funkce `foo`)
2. Inline assembler
   - Kombinace vyššího jazyka a assembleru
   - Pomocí bloku `_asm{}`
   - MSVC podporuje pouze `i386` assembler
   - GCC jiná syntaxe a komplikovanější použití

- $\Rightarrow$ Budeme používat externí assembler s nástrojem `nasm`

Registry instrukční sady AMD64

- Registr = „paměťové buňky“ uvnitř procesoru
  - Velmi rychlý přístup
  - Omezená kapacita
  - Ukládání mezi-výpočtů, operandů pro další instrukce
- Pevně daná velikost
  - 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`
- **„rodiny registrů“ sdílí společné umístění**
  - `rax, eax, ax, ah, al` (stejné písmeno **a**)
  - Obrázek
- Specifické registry: `rsp, rbp, rip, rf(lags)`
  - Pevně daný význam
  - Některé nejdou měnit *přímo*

Základní aritmetické instrukce

- Obvyklý tvar: ⟨název instrukce⟩⟨cílový operand⟩[, ⟨další operand⟩, . . .]
- Příklad: `add eax, ebx`
- Operandy:
  - `r` – registry
  - `m` – adresa místa v paměti
  - `i` – přímá hodnoty (konstanty)
- Instrukce má pevně stanovené typy operandy. Paměť nejvýše jednou
- Některé instrukce mají pevně stanovené registry (`mul, div, idiv`
  - Nezapomenout nulovat

```
1    mov r/m, r/m/i
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17                    ; rdx := rdx:rax % op1 (neznaménkové dělení)
18   idiv r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17                    ; rdx := rdx:rax % op1 (neznaménkové dělení)
18   idiv r/m         ; eax := edx:eax / op1;
19
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17                    ; rdx := rdx:rax % op1 (neznaménkové dělení)
18   idiv r/m         ; eax := edx:eax / op1;
19                    ; edx := edx:eax % op1 (znaménkové dělení)
20   idiv r/m
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17                    ; rdx := rdx:rax % op1 (neznaménkové dělení)
18   idiv r/m         ; eax := edx:eax / op1;
19                    ; edx := edx:eax % op1 (znaménkové dělení)
20   idiv r/m         ; rax := rdx:rax / op1;
21
```

```
1    mov r/m, r/m/i   ; op1 := op2
2    add r/m, r/m/i   ; op1 := op1 + op2
3    sub r/m, r/m/i   ; op1 := op1 - op2
4    neg r/m          ; op1 := - op1
5
6    inc r/m          ; op1 := op1 + 1
7    dec r/m          ; op1 := op1 - 1
8
9    mul r/m          ; edx:eax := eax * op1
10   mul r/m          ; rdx:rax := rax * op1
11   imul r, r/m      ; op1 := op1 * op2
12   imul r, r/m, i   ; op1 := op2 * op3
13
14   div r/m          ; eax := edx:eax / op1;
15                    ; edx := edx:eax % op1 (neznaménkové dělení)
16   div r/m          ; rax := rdx:rax / op1;
17                    ; rdx := rdx:rax % op1 (neznaménkové dělení)
18   idiv r/m         ; eax := edx:eax / op1;
19                    ; edx := edx:eax % op1 (znaménkové dělení)
20   idiv r/m         ; rax := rdx:rax / op1;
21                    ; rdx := rdx:rax % op1 (znaménkové dělení)
```

Praktická práce s assemblerem

1. Pomocí direktivy global určíme, jaké funkce jsou implementovány v assembleru.
2. V sekci .text implementujeme jednotlivé funkce, které vyznačíme návěštím. Funkcí může být více.
3. Funkce vrací výsledek v registru eax (resp. rax) a je ukončena instrukcí ret.
4. V jazyce C definujeme prototypy daných funkcí a voláme je standardním způsobem.
5. Při sestavování programu sloučíme kód v C a v assembleru.

Argumenty funkcí

- Na unixových systému u funkcí s méně než sedmi argumenty pevně stanovené pořadí
  - `rdi, rsi, rdx, rcx, r8, r9`
  - tj. první argument bude vždy v `rdi`, druhý vždy v `rsi`, ...
- Obsah registrů `rbx, rsp, rbp, r12, r13, r14, r15` musí být na konci funkce stejný jako na začátku
  - Bezpečná cesta: tyto registry nepoužívat
  - Když je potřebujeme $\Rightarrow$ uložit stav na zásobník (zatím neumíme)
- Nedodržení vede k podivným chybám, které se projeví až „kdoví kde“

Ukázka použití a doporučení

- Ukázka kódu
- Ke každé funkci, návěští a řádku pište komentáře
  - Lépe se pak bude hledat chyby
- Využívejte správné registry
- Copy&Paste bývá v assembleru ještě větší zlo než v C
- Zkuste si zprovoznit debugger

Úkol (1/2)

1. Napište v assembleru funkci `int obsah_obdelnika(int a, int b)`, která spočítá obsah obdélníka.
2. Napište v assembleru funkci `int obvod_ctverce(int a)`, která spočítá obvod čtverce.
3. Napište v assembleru funkci `int obsah_ctverce(int a)`, která spočítá obsah čtverce.
4. Napište v assembleru funkci `int obvod_trojuhelnika(int a, int b, int c)`, která spočítá obvod trojúhelníka.

Úkol (2/2)

5. Napište v assembleru funkci `int obvod_trojuhelnika2(int a)`, která spočítá obvod rovnostranného trojúhelníka.

6. Napište v assembleru funkci `int obsah_trojuhelnika2(int a, int b)`, která spočítá obsah pravoúhlého trojúhelníka.

7. Napište v assembleru funkci `int objem_krychle(int a)`, která spočítá objem krychle.

8. Napište v assembleru funkci
   `unsigned int avg(unsigned int a, unsigned int b, unsigned int c)`
   pro výpočet aritmetického průměru tří čísel typu `unsigned int`.