

# Porovnávání desetinných čísel, parametry typu float a double

ISU-cv12

**Ing. Jakub Husa**

Vysoké Učení Technické v Brně, Fakulta informačních technologií  
Božetěchova 1/2. 612 66 Brno - Královo Pole  
[ihusa@fit.vutbr.cz](mailto:ihusa@fit.vutbr.cz)



26. dubna 2023

## Porovnávání desetinných čísel

Desetinná čísla porovnáváme instrukcí **FCOMI**:

- Instrukce nastaví příznaky **C0**, **C2** a **C3** v registru **FSTAT** (**FPU**), a namapuje je na příznaky registru **EFLAGS** (**CPU**), podle kterých pak můžeme skákat.

```
1      fcomi st1          ; ST0 porovnej s ST1 a nastav EFLAGS
```

Příznaky jsou do registru **EFLAGS** namapovány tímto způsobem:

- **st0 > st1**       => **ZF** = 0, **PF** = 0, **CF** = 0  
  **st0 = st1**       => **ZF** = 1, **PF** = 0, **CF** = 0  
  **st0 < st1**       => **ZF** = 0, **PF** = 0, **CF** = 1  
  **nedefinováno**   => **ZF** = 1, **PF** = 1, **CF** = 1

Mapování odpovídá podmíněným skokům pro **bez-znaménková** čísla **JA**, **JB**, **JE**, ... :

```
2      fcomi st1          ; ST0 porovnej s ST1
3      ja    vetsi        ; skoc pokud ST0 > ST1
4      jb    mensi        ; skoc pokud ST0 < ST1
5      je    stejne       ; skoc pokud ST0 == ST1
6      jp    chyba        ; skoc pokud ST0 a ST1 jsou neporovnatelne (NaN > 0?)
```

Vyzkoušejte si:

- Program načte dvě desetinná čísla, porovná je, a vypíše to větší z nich.

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2  section .text           ;kodovy segment
3  _main:
4      call ReadDouble      ; nacti prvni vstup
5      call ReadDouble      ; nacti druhy vstup
6
7      condition:
8          ;fcomi st1        ; ST0 porovnej s ST1 a nastav EFLAGS
9          fcom st1          ; ST0 porovnej s ST1 a nastav FSTAT
10         fstsw ax           ; FSTAT zkopiruj do AX
11         sahf              ; AH zkopiruj do EFLAGS
12
13         ja skip           ; pokud je ST0 > ST1 preskoc pristi instrukci
14         fxch st0, st1     ; vymen hodnoty ST0 a ST1
15     skip:
16         call WriteDouble  ; vypis vystup
17
18         fstp st1          ; odstran ST1
19         ret
```

Vyzkoušejte si:

- Vytvořte si inicializované pole několika 32b desetinných čísel.
- Vytvořte si dvě 32b proměnné s hodnotami **mínus nekonečno** a **Not-a-Number**.
- Zavolejte funkci **maximum** a vypište její výsledek, vrácený v registru ST0.

```
1 ; float maximum(float* pole, int delka)
```

- Jako počáteční hodnotu maxima použijte hodnotu **mínus nekonečno**.
- Pokud délka pole není kladná, jako návratovou hodnotu použijte **Not-a-Number**.

```
2 %include "rw32-2018.inc" ;knihovna pro vstup a vystup
3 section .data           ;inicializovany segment
4     pole dd 1.0, 4.0, 2.0, 5.0, 3.0
5     ninf dd ; ???       ; minus nekonecno
6     nan  dd ; ???       ; Not-a-Number
7 section .text           ;kodovy segment
8 _main:
9     push dword 5         ; predej delku pole
10    push pole            ; predej adresu pole
11    call maximum         ; zavolej funkci maximum
12    add esp, 8           ; odstran dva parametry
13    call WriteDouble     ; vypis vyledek
14    ret
```

**Parametry typu float**

Desetinná čísla typu **float** (32b) jsou načítána a vypisována z registru **EAX**:

```
1      call ReadFloat          ; do EAX nacti 32b float
2      call WriteFloat         ; z EAX vypis 32b float
```

Abychom hodnotu z **EAX** dostali do **ST0** můžeme použít zásobník:

```
3      call ReadFloat          ; EAX = vstup
4      push eax                ; EAX vlož na zásobník
5      fld dword [esp]         ; hodnotu z vrcholu zásobníku nacti do ST0
6      add esp, 4              ; odstran hodnotu z vrcholu zásobníku
7      call WriteDouble        ; vypis hodnotu ST0
```

Pro uložení konstanty typu **float** na paměťový segment **zásobník**, nebo do celočíselných registrů (**EAX**, **EBX**, ...), potřebujeme **makro** překladače **NASM**:

```
8      mov  eax, __float32__(0.5) ; do EAX dej 32b float
9      call WriteFloat            ; z EAX vypis 32b float

10
11     push __float32__(0.5)       ; na zásobník vlož 32b konstantu 0.5
12     push dword 0.5              ; CHYBA - bez makra float nejde vložit
```

Registry koprocessoru **FPU** nejde vložit na zásobník:

```
1      push ST0                ; CHYBA - instrukce CPU, registr FPU
```

Problém můžeme obejít použitím **lokální proměnné**:

```
2  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
3
4  section .text            ;kodovy segment
5  _main:
6      push ebp              ; vytvor nove dno
7      mov  ebp, esp         ; zalohuj stare dno
8      sub  esp, 4           ; alokuj jednu lokalni promennou
9
10     call ReadDouble        ; ze vstupu nacti 64b desetinne cislo
11     fst  dword [ebp-4]     ; do lokalni promenne uloz 32b cislo z ST0
12     mov  eax, [ebp-4]      ; do EAX nahraj hodnotu z lokalni promenne
13     call WriteFloat        ; z EAX vypis 32b float
14
15     mov  ebp, esp         ; uvolni loklani promenne "add esp, 4"
16     pop  ebp              ; obnov stare dno
17     ret
```



Program zavolá funkci která sečte dvě čísla typu **float** předaná přes zásobník:

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2  section .text           ;kodovy segment
3  _main:
4      push __float32__(1.1); predej druhy parametr (Y)
5      push __float32__(2.2); predej prvni parametr (X)
6      call soucet          ; zavolej funkci soucet
7      add esp, 8           ; odstran predane parametry
8      call WriteFloat      ; vypis vystup z EAX
9      ret
10
11  soucet:
12      push ebp             ; zalohuj stare dno
13      mov ebp, esp         ; vytvor nove dno
14      sub esp, 4           ; alokuj jednu lokalni promennou
15
16      fld dword [ebp +12]  ; ST0 = Y
17      fadd dword [ebp + 8] ; ST0 = X+Y
18      fst dword [ebp - 4]  ; ST0 uloz do lokalni promenne
19      mov eax, [ebp - 4]   ; lokalni promennou uloz do EAX
20      fstp st0            ; odstran hodnotu ST0
21
22      mov esp, ebp        ; uvolni vsechny lokalni promenne
23      pop ebp             ; obnov stare dno
24      ret
```

Vyzkoušejte si:

- Napište a zavolejte funkci **dostrel** která spočítá dostřel (**d**) definovaný rovnicí:

$$d = \frac{v^2}{9.80665} \sin(2\alpha)$$

- Platnost vstupních hodnot nemusíte ověřovat.
- Na konci odstraňte hodnoty všech registrů **FPU** vyjma výstupního **ST0**.

Například:

- $V = 10.0, \alpha = 0.2 \Rightarrow d = 3.970962$
- $V = 5.0, \alpha = 0.8 \Rightarrow d = 2.548203$

```
1  %include "rw32-2018.inc" ; knihovna
2  section .text           ; kodovy segment
3  _main:
4      call ReadFloat      ; nacti float (V)
5      push eax            ; predej parametr
6      call ReadFloat      ; nacti float (a)
7      push eax            ; predej parametr
8      call dostrel        ; zavolej funkci
9      add esp, 8          ; odstran parametry
10     call WriteDouble     ; vypis vysledek
11     ret
```

**Parametry typu double**

Desetinná čísla typu **double** (64b) na zásobníku předáváme po polovinách:

- Procesor proměnné v paměti ukládá ve formátu **little-endian** a **méně významné BAJTY** (ne **bity**) jsou tedy na **nižší adrese**.

```

1 section .data
2     X db 0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF ;pole osmi bytu
3     Y dq 0x0123456789ABCDEF ;jeden qword
    
```

adresa	[X+0]	[X+1]	[X+2]	[X+3]	[X+4]	[X+5]	[X+6]	[X+7]
hodnota	0x01	0x23	0x45	0x67	0x89	0xAB	0xCD	0xEF

  

hodnota	0xEF	0xCD	0xAB	0x89	0x67	0x45	0x23	0x01
adresa	[Y+0]	[Y+1]	[Y+2]	[Y+3]	[Y+4]	[Y+5]	[Y+6]	[Y+7]

Instrukcí **PUSH** na zásobník proto nejprve předáváme **horní** polovinu proměnné:

- Návrátové hodnoty typu **double** vrátíme v registru **ST0** (konvence **CDECL**).

```

4 push dword [Y + 4] ;predej hornich 32b promenne typu double
5 push dword [Y + 0] ;predej dolnich 32b promenne typu double
    
```

Funkce sečte dvě 64b desetinná čísla předaná hodnotou.

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2  section .data           ;inicializovany segment
3      X dq 1.1            ; 64b promenna s hodnotou 1.1
4      Y dq 2.2            ; 64b promenna s hodnotou 2.2
5  section .text           ;kodovy segment
6  _main:
7      push dword [Y + 4]   ; predej hornich 32b promenne Y
8      push dword [Y + 0]   ; predej dolnich 32b promenne Y
9      push dword [X + 4]   ; predej hornich 32b promenne X
10     push dword [X + 0]   ; predej dolnich 32b promenne X
11     call soucet          ; zavolej funkci soucet
12     add esp, 16          ; odstran predane parametry
13     call WriteDouble     ; vypis vystup z ST0
14     ret
15
16 soucet:                  ; double soucet(double X, double Y)
17     push ebp             ; zalohuj stare dno
18     mov ebp, esp         ; vytvor nove dno
19     fld qword [ebp + 16] ; do ST0 nahraj Y
20     fld qword [ebp + 8]  ; do ST0 nahraj X (ST1 = Y)
21     faddp st1, st0       ; k ST1 pricti ST0 a odeber ho
22     pop ebp              ; obnov stare dno
23     ret
```

Funkce sečte dvě 64b desetinná čísla předaná odkazem.

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2  section .data           ;inicializovany segment
3      X dq 1.1            ; 64b promenna s hodnotou 1.1
4      Y dq 2.2            ; 64b promenna s hodnotou 2.2
5  section .text           ;kodovy segment
6  _main:
7      push dword Y         ; predej adresu promenne Y
8      push dword X         ; predej adresu promenne X
9      call soucet          ; zavolej funkci soucet
10     add esp, 8           ; odstran predane parametry
11     call WriteDouble     ; vypis vystup z ST0
12     ret
13
14  soucet:                 ; double soucet(double* X, double* Y)
15     push ebp             ; zalohuj stare dno
16     mov ebp, esp         ; vytvor nove dno
17     mov esi, [ebp + 12]  ; do ESI nahraj adresu promenne Y
18     fld qword [esi]      ; do ST0 nahraj hodnotu z adresy ESI
19     mov esi, [ebp + 8]   ; do ESI nahraj adresu promenne X
20     fld qword [esi]      ; do ST0 nahraj hodnotu z adresy ESI
21     faddp st1, st0       ; k ST1 pricti ST0 a odeber ho
22     pop ebp             ; obnov stare dno
23     ret
```

Vyzkoušejte si:

- Implementujte a zavolejte funkci **odmocnina**, definovanou dle hlavičky.

1 

```
; double odmocnina(double X)
```

- Funke spočítá odmocninu **bez** použití instrukce **FSQRT**, provedením **deseti** iterací **babylonské metody** definované iterační rovnicí:

$$Y_{i+1} = \frac{1}{2} \left( \frac{X}{Y_i} + Y_i \right)$$

- Pro jednoduchost předpokládejte že **X** vždy bude kladné číslo.
- Jako počáteční hodnotu odmocniny **Y<sub>0</sub>** použijte vstupní hodnotu **X**.
- Na konci funkce odstraňte hodnoty všech registrů **FPU** vyjma výstupního **ST0**.

Například:

- 4.0** => **2.000000**
- 10.0** => **3.162278**
- 100.0** => **10.000000**