Desetinná čísla, matematický koprocesor a proměnné typu float

ISU-cv11

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole

ihusa@fit.vutbr.cz

23. dubna 2024

Desetinná čísla

Celá čísla (např. 10, datový typ int) jsou ukládána v doplňkovém kódu:

- První bit čísla určuje znaménko, a můžeme ho obrátit dvojkovým doplňkem.

Desetinná čísla (např. 10.0, datový typ float) jsou ukládána s plovoucí řádovou čárkou dle standardu IEEE 754, podle kterého se zápis čísla rozděluje na tři části:

- Znaménko (1b, S) – pro kladná čísla 0, pro záporná čísla 1.
- Exponent (8b, E) – udává mocninu základu čísla, a je uložen v kódu transformované nuly (viz. p1s40) s posunem o hodnotu 127.
- Mantisa (23b, M) – představuje desetinnou část čísla, a až na výjimky je normalizovaná – první jednička je implicitní a do mantisy se nepíše.

Hodnotu desetinného čísla uloženého jako 32b float spočítáme dle rovnice:

$$X = -1^{S} * 2^{E-127} * (1.M)$$

Na ukázku

Znaménko určuje jestli je číslo kladné nebo záporné:

- 1.0 = 0 01111111 00000000000000000000000
- -1.0 = 1 01111111 00000000000000000000000

Exponent určuje mocninu základu čísla (2^{E-127}):

- 0.5 = 0 01111110 00000000000000000000000
- 1.0 = 0 01111111 00000000000000000000000
- 2.0 = 0 10000000 00000000000000000000000

Jednotlivé bity mantisy, představují polovinu, čtvrtinu, osminu, šestnáctinu, ... :

- 1.0 = 0 01111111 00000000000000000000000
- 1.25 = 0 01111111 01000000000000000000000
- 1.5 = 0 01111111 10000000000000000000000
- 1.75 = 0 01111111 11000000000000000000000

Speciální hodnoty jsou definovány pomocí konstant:

- ± 0.0 = ? 00000000 00000000000000000000000 je nula
- $\pm$??? = ? 00000000 cokoliv kromě samých nul je denormalizované číslo
- $\pm\infty$ = ? 11111111 00000000000000000000000 je nekonečno
- qNaN = ? 11111111 1 cokoliv kromě samých nul je tiché nečíslo
- sNaN = ? 11111111 0 cokoliv kromě samých nul je signalizující nečíslo

Převod z desítkové soustavy na 32b float

Například, převod čísla -5.375 z desítkové soustavy do plovoucí řádové čárky:

- Znaménko určíme podle toho jestli je číslo kladné (S = 0) nebo záporné (S = 1).

$$-5.375 \quad \Rightarrow \quad S = 1$$

- Absolutní hodnotu čísla převedeme z desítkové do dvojkové (viz. cv01s19).

$$(5.375)_{10} = (101.011)_2$$

- Posunutím desetinné tečky číslo normalizujeme do podoby "1.mantisa".

$$(101.011)_2 \quad \Rightarrow \quad 1.01011$$

- Exponent spočítáme z posuvu dle rovnice "E-127 = posuv".

$$E\text{-}127 = 2 \quad \Rightarrow \quad E = (129)_{10} = (10000001)_2$$

- Znaménko, exponent a mantisu spojíme, a zprava doplníme nulami na 32b.

	Zn.	Exponent	Mantisa
-5.375 =	1	10000001	01011000000000000000

Převod z 32b float do desítkové soustavu

Například, převod čísla 11000000101011000000000000000000 do desítkové soustavy:

- Z exponentu spočítáme velikost posuvu, dle rovnice "E-127 = posuv".

$$E = (10000001)_2 = (129)_{10} \quad => \quad 129\text{-}127 = posuv = 2$$

- Z mantisy vytvoříme normalizované číslo ve formátu "1.mantisa".

$$1.01011$$

- Posunutím desetinné tečky číslo de-normalizujeme.

$$1.01011 \quad => \quad (101.011)_2$$

- Převodem z dvojkové do desítkové (viz. cv01s18) získáme absolutní hodnotu.

$$(101.011)_2 = (5.375)_{10}$$

- Podle znaménka určíme jestli je číslo kladné (S = 0) nebo záporné (S = 1).

Zn.	Exponent	Mantisa
1	10000001	01011000000000000000 = -5.375

Desetinné číslo převeďte z desítkové soustavy na 32b float:

- -3.0 = ???
- 2.25 = ???
- 31.0 = ???
- -0.875 = ???

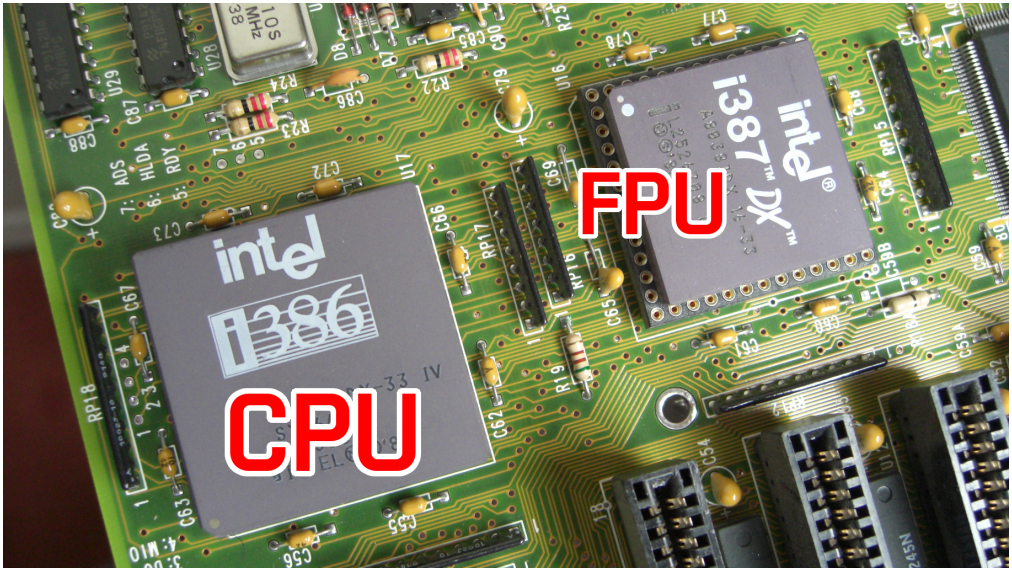
Desetinné číslo převeďte z 32b float do desítkové soustavy:

- ??? = 0 10000001 11100000000000000000000
- ??? = 0 10000101 01000000000000000000000
- ??? = 1 10000010 00000100000000000000000
- ??? = 0 01111011 00000000000000000000000

Matematický koprocesor

Počítání s desetinnými čísly zařizuje matematický koprocesor (FPU):

- Pozor – jde o jiné zařízení něž procesor (CPU) se kterým jsme pracovali doposud!



Registry koprocesoru

Pro procesor (CPU) a koprocesor (FPU) píšeme jeden společný zdrojový kód, ale oba z nich používají jiné registry a jinou instrukční sadu:

- Procesor – používá sadu Intel x86, kterou jsme používali doposud.
- Koprocesor – používá sadu Intel x87, a její instrukce vždy začínají písmenem F.

Koprocesor obsahuje osm obecných 80b registrů označovaných ST0 až ST7:

- Registry FPU pracují jako cyklický zásobník – nová položka se vždy uloží do ST0, předchozí obsah ST0 se odsune do ST1, obsah ST1 se posune do ST2, atd.
- Při odebrání položky se smaže obsah ST0, do kterého se vrátí obsah ST1, atd.
- Pozor – nijak **NESOUVISÍ** s paměťovým segmentem zásobník, ani s ESP a EBP!

Koprocesor obsahuje tři 16b řídící registry jménem FTAG, FSTAT, a FCTRL:

- FTAG (tag) – pro každý z registrů ST0 až ST7 si pamatuje jestli obsahují normální číslo, nulu, jiné speciální číslo nebo volno (smetí).
- FSTAT (status) – obsahuje ukazatel na ST0 a příznaky (ekvivalent EFLAGS).
- FCTRL (control) – obsahuje nastavení FPU (nad rámec cvičení ISU).

Načítání a vypisování hodnot

Nové položky do zásobníku FPU můžeme načítat několika způsoby:

- Ze vstupu – knihovní funkcí ReadDouble (vypisujeme WriteDouble).

```
1 call ReadDouble  ; cislo ze vstupu nacti jako novou polozku
2 call WriteDouble ; obsah registru ST0 vypis na vystup
```

- Z jiného registru FPU – instrukcí FLD (Load).

```
3 fld  st0         ; obsah ST0 nacti jako novu polozku
4 fld  st1         ; obsah ST1 nacti jako novu polozku
5 fld  st2         ; obsah ST2 nacti jako novu polozku
```

- Jako speciální konstantu – instrukcemi FLDZ, FLD1 a FLDPI.

```
6 fldz             ; konstantu 0.00000... nacti jako novu polozku
7 fld1             ; konstantu 1.00000... nacti jako novu polozku
8 fldpi            ; konstantu 3.14159... nacti jako novu polozku
```

- Pozor – koprocesor neumí načítání obecných konstant!

```
9 fld 10.0         ; CHYBA - koprocesor neumi nacitat konstanty
```

Pro základní operace používáme instrukce FADD, FSUB, FMUL a FDIV:

- Základní operace mají dva operandy – cílový a zdrojový.
- Jedním z operandů vždy musí být registr ST0.

```
1 fadd st0, st1 ; ST0 = ST0 + ST1
2 fsub st0, st1 ; ST0 = ST0 - ST1
3 fmul st1, st0 ; ST1 = ST1 * ST0 (poradi operandu muzeme vymenit)
4 fdiv st1, st2 ; CHYBA - jednim z operandu musi byt ST0
```

Pro pokročilé operace používáme instrukce FCHS, FABS, FSIN, FCOS, FSQRT, atd.:

- Pokročilé operace operandy nemají – vždy pracují s registrem ST0.

```
5 fchs          ; ST0 =  neg(ST0)  ; negace (change sign)
6 fabs          ; ST0 =  abs(ST0)  ; absolutni hodnota
7 fsin          ; ST0 =  sin(ST0)  ; sinus
8 fcos          ; ST0 =  cos(ST0)  ; cosinus
9 fsqrt         ; ST0 = sqrt(ST0)  ; odmocnina (square root)
```

Program ze vstupu načte dvě čísla (X a Y) a vypíše jejich součet:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:                                              ; ST0 ST1
5      call ReadDoubleNewLine   ; nacti prvni vstup ;   X   ?
6      call ReadDoubleNewLine   ; nacti druhy vstup ;   Y   X
7      fadd st0, st1            ; ST0 = ST0 + ST1   ; Y+X  X
8      call WriteDoubleNewLine  ; vypis ST0
9      ret
```

Vyzkoušejte si:

- Načtěte si dvě čísla (A a B) představující délku odvěsen pravoúhlého trojúhelníku.
- Spočítejte délku jeho přepony (c) a obvod (o) definované dle Pythagorovy věty.

$$c = \sqrt{a^2 + b^2} \qquad o = a + b + c$$

Například:

- 1.0, 1.0 => c = 1.41421, o = 3.41421
- 3.0, 4.0 => c = 5.00000, o = 12.00000

Přesouvání hodnot

Pro přesouvání načtených hodnot používáme instrukce FST a FXCH:

- Tyto instrukce mají jen cílový a operand – zdrojovým je vždy ST0.
- FST (Store) – obsah ST0 zkopíruje do jiného registru (FPU MOV).
- FXCH (Exchange) – obsah ST0 vymění s jiným registem (FPU XCHG).

```
1 fst  st1          ; obsah ST0 zkopiruj do ST1 (MOV)
2 fxch st1          ; obsah ST0 vymen    s  ST1 (XCHG)
```

Pozor hodnotu **NESMÍME** zkopírovat do prázdného registru:

- Hodnota by se zkopírovala, ale neaktualizoval by se ukazatel registru FSTAT!
- Hodnotu vždy musíme nejprve načíst jako novou položku do ST0, a pak ji můžeme zkopírovat nebo vyměnit s obsahem jiného registru.

```
3                                       ; ST0  ST1  ST2  ST3
4 fldpi             ; do ST0 nacti PI   ; 3.1   ?    ?    ?
5 fld  st0          ; do ST0 nacti ST0  ; 3.1  3.1   ?    ?
6 fst  st2          ; ST0 zkopiruj do ST2 ; 3.1 3.1 CHYBA  ?
```

Načtené položky ze zásobníku odstraňujeme (z ST0) instrukcí FSTP,
nebo přidáním přípony P (Pop) na konec základní aritmetické instrukce:

- FSTP má jeden operand – kam chceme obsah ST0 před smazáním zkopírovat.
- Pokud chceme položku pouze smazat, jako operand použijeme ST0.
- U aritmetické instrukce s příponou P musí být registr ST0 použit jako zdrojový.

```
1  fstp st0          ; obsah ST0 zkopiruj do ST0 a pak odstran ST0
2  fstp st1          ; obsah ST0 zkopiruj do ST1 a pak odstran ST0
3  fstp st2          ; obsah ST0 zkopiruj do ST2 a pak odstran ST0
4
5  faddp st1, st0    ; ST1 = ST1 + ST0 a odstran ST0
6  faddp st0, st1    ; CHYBA - zdrojovym operandem neni ST0
```

Počáteční hodnota registrů ST0 až ST7 je nedefinovaná:

- Pokud je návratovou hodnotu funkce desetinné číslo, vracíme ho v ST0 (ne EAX).
- Obsah registrů ST1 až ST7 na konci funkce vždy odstraňujeme ze zásobníku.
- Pokud funkce nemá návratovou hodnotu, odstraňujeme i ST0.

Sledujte jak se bude měnit obsah registrů FPU pokud na vstupu byla čísla X a Y:

```
1  %include 'rw32.inc' ;knihovna pro vstup a vystup
2  section .text       ;kodovy segment
3  main:               ; ST0  ST1  ST2  ST3  ST4  ST5  ST6  ST7
4      fld1            ; 1.0   ?    ?    ?    ?    ?    ?    ?
5      call ReadDouble ;  X   1.0   ?    ?    ?    ?    ?    ?
6      fldpi           ; 3.1   X   1.0   ?    ?    ?    ?    ?
7      call ReadDouble ;  Y   3.1   X   1.0   ?    ?    ?    ?
8      fldz            ; 0.0   Y   3.1   X   1.0   ?    ?    ?
9      fld  st0        ; 0.0  0.0   Y   3.1   X   1.0   ?    ?
10     fld  st3        ; 3.1  0.0  0.0   Y   3.1   X   1.0   ?
11     fxch st5        ;  X   0.0  0.0   Y   3.1  3.1  1.0   ?
12     fst  st2        ;  X   0.0   X    Y   3.1  3.1  1.0   ?
13     fstp st0        ; 0.0   X    Y   3.1  3.1  1.0   ?    ?
14     fstp st0        ;  X    Y   3.1  3.1  1.0   ?    ?    ?
15     fstp st0        ;  Y   3.1  3.1  1.0   ?    ?    ?    ?
16     fstp st1        ;  Y   3.1  1.0   ?    ?    ?    ?    ?
17     fstp st1        ;  Y   1.0   ?    ?    ?    ?    ?    ?
18     fstp st1        ;  Y    ?    ?    ?    ?    ?    ?    ?
19     fstp st0        ;  ?    ?    ?    ?    ?    ?    ?    ?
20     ret
```

Vyzkoušejte si:

- Ze vstupu načtěte tři čísla (A, B, C) představující délky stran trojúhelníku.
- Spočítejte poloviční obvod (s) a obsah (S) definované dle Heronova vzorce.
- Odstraňte hodnoty všech registrů FPU vyjma ST0 a vypište výsledek.
- Neplatné vstupy můžete ignorovat (viz. cv12).

$$s = \frac{a + b + c}{2} \qquad S = \sqrt{s * (s - a) * (s - b) * (s - c)}$$

Například:

- 3.0, 4.0, 5.0 => s = 6.00000, S = 6.00000
- 1.0, 1.0, 1.0 => s = 1.50000, S = 0.43301
- 1.0, 2.0, 4.0 => s = 3.50000, S = NaN

Proměnné typu float

Datové typy desetinných čísel

Koprocesor u desetinných čísel rozeznává několik datových typů různé velikosti:

Velikost		Segment			Jméno		přesnost
bity	Bajty	.bss	.data	.text	typ	velikost	
32	4	resd	dd	dword	float	double-word	single
64	8	resq	dq	qword	double	quad-word	double
80	10	rest	dt	tword	long double	ten-word	extended

Aby se číslo v paměti uložilo jako desetinné, musí obsahovat desetinnou tečku:

- Pozor – celá a desetinná čísla mají jiný rozsah platných velikostí!
- Celá – 8b, 16b, 32b, 64b.
- Desetinná – 32b, 64b, 80b.

```
1  section .data
2      A db 10      ;  8b cele       cislo jmenem A s hodnotou 10
3      B dw 20      ; 16b cele       cislo jmenem B s hodnotou 20
4      C dd 30      ; 32b cele       cislo jmenem C s hodnotou 30
5      D dq 40      ; 64b cele       cislo jmenem D s hodnotou 40
6
7      E dd 50.0    ; 32b desetinne cislo jmenem E s hodnotou 50.0
8      F dq 60.0    ; 64b desetinne cislo jmenem F s hodnotou 60.0
9      G dt 70.0    ; 80b desetinne cislo jmenem G s hodnotou 70.0
```

Načítání a ukládání desetinných čísel

Desetinná čísla načítáme a ukládáme instrukcemi FLD a FST:

- Při ukládání 80b desetinných čísel musíme používat příponu P.

```
1  fld   dword [E]    ; 32b desetinne cislo E nacti jako novu polozku
2  fld   qword [F]    ; 64b desetinne cislo F nacti jako novu polozku
3  fld   tword [G]    ; 80b desetinne cislo G nacti jako novu polozku
4
5  fst   dword [E]    ; ST0 uloz do E jako 32b desetinne cislo
6  fst   qword [F]    ; ST0 uloz do F jako 64b desetinne cislo
7  fstp  tword [G]    ; ST0 uloz do G jako 80b desetinne cislo a odstran ST0
```

Například – program do proměnné X uloží konstantu π jako 32b float:

```
8  section .bss        ;ne-inicializovany datovy segment
9      X resd 1        ; rezervujeme misto pro jednu 32b promennou
10 section .text       ;kodovy segment
11 main:
12     fldpi           ; do ST0 nacti konstantu PI
13     fst  dword [X]  ; ST0 uloz do promenne X jako 32b desetinne cislo
14     ret             ; [X] = 3.14159 = 0 10000000 10010010000111111011011
15                     ;                 S EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
```

Načítání a ukládání celých čísel

Celá čísla načítáme a ukládáme instrukcemi FILD a FIST:

- Předpona FI provede převod z doplňkového kódu na float (nebo obráceně).
- 8b celá čísla do FPU načítat ani ukládat nelze.
- Při ukládání 64b celých čísel musíme používat příponu P.

```
1  fild   word [B]     ; 16b cele cislo B nacti jako novu polozku
2  fild  dword [C]     ; 32b cele cislo C nacti jako novu polozku
3  fild  qword [D]     ; 64b cele cislo D nacti jako novu polozku
4
5  fist   word [B]     ; ST0 uloz do B jako 16b cele cislo
6  fist  dword [C]     ; ST0 uloz do C jako 32b cele cislo
7  fistp qword [D]     ; ST0 uloz do D jako 64b cele cislo a odstran ST0
```

Například – program do proměnné Y uloží konstantu π zaokrouhlenou na celé číslo:

```
8  section .bss        ;ne-inicializovany datovy segment
9      Y resd 1        ; rezervujeme misto pro jednu 32b promennou
10 section .text       ;kodovy segment
11 main:
12     fldpi           ; do ST0 nacti konstantu PI
13     fist dword [Y]  ; ST0 uloz do promenne Y jako 32b cele cislo
14     ret             ; [Y] = 3 = 00000000000000000000000000000011
```

Vyzkoušejte si:

- Vytvořte si dvě inicializované 32b celočíselné proměnné dva = 2 a pet = 5, a jednu ne-inicializovanou 32b proměnnou rez.
- Napište program který spočítá zlatý řez definovaný rovnicí:

$$\text{rez} = \frac{1+\sqrt{5}}{2} = 1.618034...$$

- Výsledek uložte do proměnné rez, její hodnotu zobrazte v debuggeru, a odstraňte hodnoty všech registrů FPU, včetně ST0.

Vyzkoušejte si:

- Vytvořte si inicializované pole několika 80b desetinných čísel.
- Napište funkci Suma definovanou podle následující hlavičky a volanou dle konvence CDECL, která spočítá a vrátí sumu prvků předaného pole.

```
1   ;long double Suma(long double* pole, int delka);
```

Například:

```
2  %include 'rw32.inc'          ;knihovna pro vstup a vystup
3  section .data                ;inicializovany datovy segment
4      arr dt 1.0, 2.0, 3.0, 4.0, 5.0 ; pole peti 80b desetinnych cisel
5  section .text                ;kodovy segment
6  main:
7      push dword 5             ; predavame delku  pole (druhy parametr)
8      push arr                 ; predavame adresu pole (prvni parametr)
9      call Suma                ; volame funkci Suma
10     add  esp, 8              ; predane parametry mazeme ze zasobniku
11     call WriteDoubleNewLine  ; vypisujeme navratovou hodnotu Sumy
12     fstp st0                 ; navratovou hodnotu Sumy mazeme z ST0
13     ret
```