Desetinná čísla, matematický koprocesor a proměnné s plovoucí řádovou čárkou

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

20. dubna 2023

Desetinná čísla

Desetinná čísla

Celá čísla (10, int) jsou ukládána v doplňkovém kódu:

- První bit čísla určuje znaménko které můžeme obrátit dvojkovým doplňkem.

Desetinná čísla (10.0, float) jsou ukládána s plovoucí řádovou čárkou
dle standardu IEEE 754, který zápis čísla rozděluje na tři části:

- Znaménko (1b, S) – nula pro kladná čísla, jednička pro záporná.
- Exponent (8b, E) – udává mocninu čísla a jeho základní hodnota je 127.
- Mantisa (23b, M) – představuje desetinnou část čísla, až na výjimky je normalizovaná a první jednička se do ní tedy nepíše.

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

Na ukázku

Znaménko určuje jestli je číslo kladné nebo záporné:

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

Změna exponentu číslo násobí nebo dělí dvěma:

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

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

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

Speciální hodnoty jsou definovány konstantami:

- 0.0 = 0 00000000 00000000000000000000000
- ∞ = 0 11111111 00000000000000000000000
- NaN = 0 11111111 cokoliv kromě samých nul

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. cv1s19).

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

- 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 float -> desítková

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. cv1s18) 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 do plovoucí řádové čárky:

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

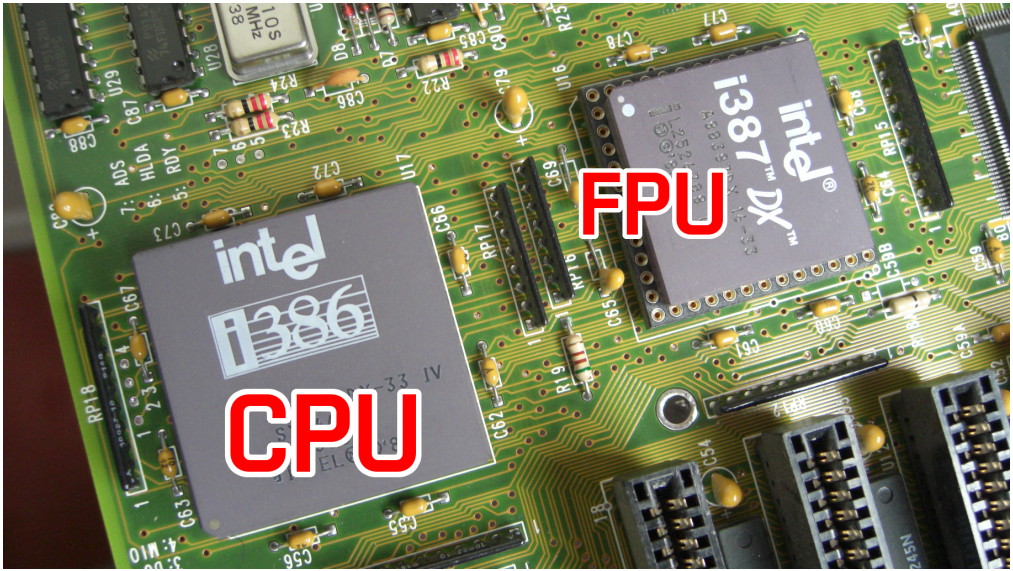
Plovoucí řádovou čárku převeďte na desetinné číslo:

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

Matematický koprocesor

Počítání s desetinnými čísly zařizuje matematický koprocesor (FPU):

- Pozor, je to jiné zařízení něž procesor (CPU) se kterým jsme pracovali doposud!



Matematický koprocesor

Procesor a koprocesor jsou řízeny jedním společným zdrojovým kódem, každý z nich ale má svoji vlastní instrukční sadu a registry:

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

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

- Registry FPU pracují jako zásobník – když něco vložíme do registru ST0, jeho předchozí obsah se automaticky přesune do registru ST1, atd.
- Pozor, tyto registry nijak **NESOUVISEJÍ** s paměťovým segmentem "zásobník" ani s registry ESP a EBP – jde o jiné fyzické zařízení!

Koprocesor také obsahuje tři řídící registry (viz. cv12):

- FTAG (tag) – Pamatuje si obsazenost obecných registrů.
- FSTAT (status) – FPU ekvivalent příznakového registru EFLAGS.
- FCTRL (control) – Nastavuje chování jednotky FPU, umožňuje přerušení.

Desetinná čísla načítáme a vypisujeme (z ST0) knihovními funkcemi:

```
1  call ReadDouble  ; do ST0 nacti 64b desetinne cislo
2  call WriteDouble ; z  ST0 vypis 64b desetinne cislo
```

Pro manipulaci s hodnotami registrů používáme instrukce FLD, FST a FXCH:

```
3  fld  st1         ; hodnotu ST1 nacti    do ST0 (jako novou hodnotu)
4  fst  st1         ; hodnotu ST0 zkopiruj do ST1 (MOV)
5  fxch st1         ; hodnotu ST0 vymen    s  ST1 (XCHG)
```

Konstanty 0.0, 1.0 a π můžeme načítat speciálními variantami instrukce FLD:

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

Hodnoty odstraňujeme přidáním přípony P (pop) na konec nějaké instrukce:

- Příponu nejčastěji přidáváme na konec instrukce FST.

```
9  fstp st0         ; ST0 zkopiruj do ST0 a odstran ST0 (odstran ST0)
10 fstp st1         ; ST0 zkopiruj do ST1 a odstran ST0 (odstran ST1)
```

Vyzkoušejte si:

- Jak se bude měnit obsah registrů FPU při provádění následujících instrukcí?

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2
3  section .text            ;kodovy segment
4  _main:
5      push ebp             ; zalohuj stare dno
6      mov  ebp, esp        ; vytvor nove dno
7
8                           ;  ST0   ST1   ST2
9      fld1                 ;  1.00  ?     ?
10     call ReadDouble      ;  XXX   1.00  ?
11     fldpi                ;  3.14  XXX   1.00
12     fxch st1             ;  XXX   3.14  1.00
13     fst  st2             ;  XXX   3.14  XXX
14     fstp st0             ;  3.14  XXX   ?
15     fstp st1             ;  3.14  ?     ?
16
17     pop  ebp             ; obnov stare dno
18     ret
```

Instrukční sada obsahuje základní aritmetické instrukce FADD, FSUB, FMUL a FDIV:

- Jedním z jejich operandů vždy musí být registr ST0.

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

Pokročilé aritmetické instrukce nemají operandy a vždy počítají s registrem ST0:

```
5  fchs          ; ST0 =  neg(ST0)  ; negace
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
```

Počáteční hodnota registrů FPU je nedefinovaná:

- Na konci programu bychom měli vyčistit všechny registry vyjma výstupního ST0.
- Hodnoty odstraňované z ST0 se jako smetí objevují v ST7, ST6, ST5, ... (viz. cv12).

Vyzkoušejte si:

- Program ze vstupu načte dvě čísla (A a B) a vypíše jejich součet.

```
1  %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2
3  section .text            ;kodovy segment
4  _main:
5      push ebp             ; zalohuj stare dno
6      mov  ebp, esp        ; vytvor nove dno
7                                                ; ST0 ST1
8      call ReadDouble      ; nacti prvni vstup ;  A   ?
9      call ReadDouble      ; nacti druhy vstup ;  B   A
10     fadd st0, st1        ; ST0 = ST0 + ST1   ; B+A  A
11     fstp st1             ; smaz ST1          ; B+A  ?
12     call WriteDouble     ; vypis vystup
13
14     pop  ebp             ; obnov stare dno
15     ret
```

Vyzkoušejte si:

- Napište program který ze vstupu načte dvě čísla (A a B) a spočítá délku přepony pravoúhlého trojúhelníku definovanou rovnicí.
- Odstraňte hodnotu všech registrů FPU s výjimkou ST0, a vypište výsledek.

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

Například:

- 1.0, 1.0 => 1.414...
- 3.0, 4.0 => 5.000...

Proměnné s plovoucí řádovou čárkou

Koprocesor rozeznává tři datové typy různé velikosti:

- Proměnné s přesností menší jak 80b se při zápisu do paměti zaokrouhlují.

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

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

```
1  section .data
2      A dw 10          ; 16b cele cislo s hodnotou 10
3      B dd 20          ; 32b cele cislo s hodnotou 20
4      C dq 30          ; 64b cele cislo s hodnotou 30
5      D dt 40          ; CHYBA - cela cisla nemohou mit velikost 80b
6
7      X dd 50.0        ; 32b desetinne cislo s hodnotou 50.0
8      Y dq 60.0        ; 64b desetinne cislo s hodnotou 60.0
9      Z dt 70.0        ; 80b desetinne cislo s hodnotou 70.0
```

Desetinná čísla načítáme a ukládáme instrukcemi FLD a FST:

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

```
1   fld   dword [X] ; nacti 32b desetinne cislo X
2   fld   qword [Y] ; nacti 64b desetinne cislo Y
3   fld   tword [Z] ; nacti 80b desetinne cislo Z
4
5   fst   dword [X] ; uloz  32b desetinne cislo X
6   fst   qword [Y] ; uloz  64b desetinne cislo Y
7   fstp  tword [Z] ; uloz  80b desetinne cislo Z a odstran ST0
```

Celá čísla načítáme a ukládáme instrukcemi FILD a FIST:

- Předpona FI provede převod z doplňkového kódu na float (nebo obráceně).
- Při ukládání 64b celých čísel musíme používat příponu P:

```
9   fild   word [A] ; nacti 16b cele cislo A
10  fild  dword [B] ; nacti 32b cele cislo B
11  fild  qword [C] ; nacti 64b cele cislo C
12
13  fist   word [A] ; uloz  16b cele cislo A
14  fist  dword [B] ; uloz  32b cele cislo B
15  fistp qword [C] ; uloz  64b cele cislo C a odstran ST0
```

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:

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

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

Vyzkoušejte si:

- Vytvořte si inicializované pole několika 80b desetinných čísel.
- Napište funkci suma definovanou dle hlavičky, která spočítá sumu prvků pole.

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

- Zavolejte funkci suma a vypište její výsledek, vrácený v registru ST0.

```
1   %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2
3   section .data            ;inicializovany segment
4       pole dt 1.0, 2.0, 3.0, 4.0, 5.0
5
6   section .text            ;kodovy segment
7   _main:
8       push dword 5         ; predej delku pole
9       push pole            ; predej adresu pole
10      call suma            ; zavolej funkci sum
11      add  esp, 8          ; odstran dva parametry
12      call WriteDouble     ; vypis vyledek
13      ret
```