Matematický koprocesor, desetinné konstanty a parametry

ISU-cv11

Ing. Jakub Husa Ph.D.

Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

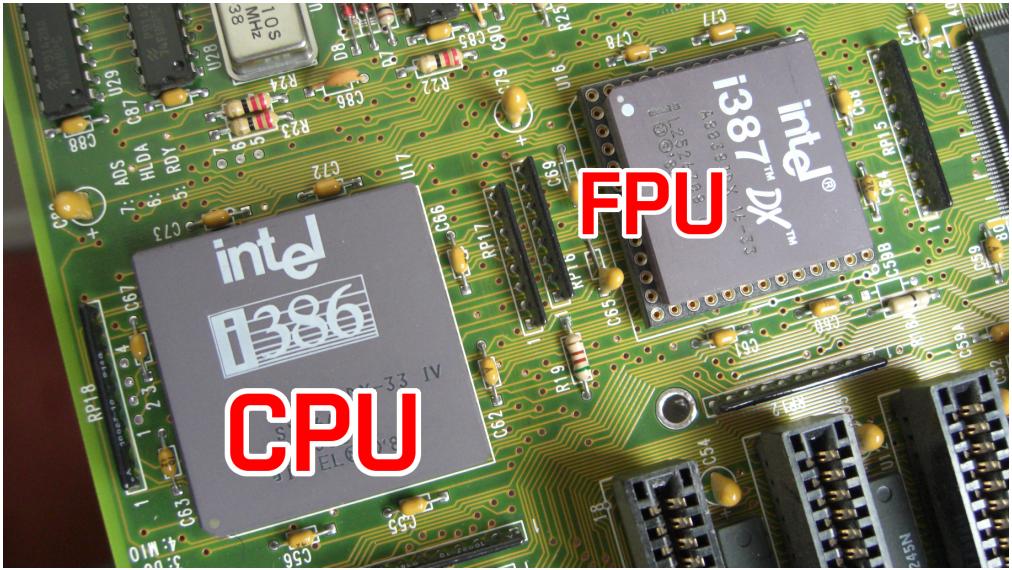
25. dubna 2025

Matematický koprocesor

Matematický koprocesor

Pro počítání s desetinnými čísly používáme 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 obě zařízení používají zcela 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 pro aritmetické výpočty
označované jako ST0 až ST7 a které se chovají jako zásobník:

- Při načtení nové položky se hodnota vždy uloží do registru ST0,
  předchozí obsah ST0 se posune do ST1, obsah ST1 se posune do ST2, atd.
- Při smazání položky se odstraní ST0, do kterého se vrátí ST1, do něj ST2, atd.
- Pozor – registry koprocesoru ST0 až ST7 **NEMAJÍ** žádnou souvislost s
  paměťovým segmentem zásobník (viz. cv08), ani s registry 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, jinou speciální hodnotu (viz. cv09) nebo volno (smetí).
- FSTAT (status) – obsahuje pozici ST0 a příznaky (ekvivalent EFLAGS v CPU).
- FCTRL (control) – obsahuje nastavení koprocesoru (nad rámec cvičení ISU).

Debugger - registry koprocesoru

80b zápis obsahuje znaménko (1b), exponent (15b), normalizaci (1b) a mantisu (63b):

⊖ X87 Registers (FPU)

H b [R2] ST0 = Infinity, 2 (SPECIAL)
0111111111111111100000000000000000000000000000000000000000000000000000000000000000
H b [R3] ST1 = NaN, 2 (SPECIAL)
1111111111111111100000000000000000000000000000000000000000000000000000000000000000
H b [R4] ST2 = 0.00000, 1 (ZERO)
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
H b [R5] ST3 = 1.00000, 0 (VALID)
0011111111111111100000000000000000000000000000000000000000000000000000000000000000
H b [R6] ST4 = 2.00000, 0 (VALID)
0100000000000000100000000000000000000000000000000000000000000000000000000000000000
H b [R7] ST5 = 3.00000, 0 (VALID)
0100000000000000110000000000000000000000000000000000000000000000000000000000000000
H b [R0] ST6 = NaN 3 (EMPTY)
H b [R1] ST7 = NaN 3 (EMPTY)

H b FSTAT = 0001000000000000

C3=0 | TOP = 2 | C2=0 | C1=0 | C0=0 | ES=0 | SF=0 | PE=0 | UE=0 | OE=0 | ZE=0 | DE=0 | IE=0

H b FTAG = 0000000110101111

R7 = 0 | R6 = 0 | R5 = 0 | R4 = 1 | R3 = 2 | R2 = 2 | R1 = 3 | R0 = 3

H b FCTRL = 0000001101111111

RC = nearest | PC = extended | PM=1 | UM=1 | OM=1 | ZM=1 | DM=1 | IM=1

Načítání hodnot

Pro načítání desetinných čísel používáme instrukci FLD (Float LoaD):

- Instrukcí i FLD můžeme čísla načítat jen z paměti a z registrů FPU.
- Při načítání z paměti musíme použít pseudo-instrukci pro velikost (viz. cv03).

```
1  %include 'rw32.inc' ;knihovna pro vstup a vystup
2  section .data       ;inicializovany segment
3      X dd 10.0       ; 32b desetinne cislo X
4      Y dd 20.0       ; 32b desetinne cislo Y
5  section .text       ;kodovy segment
6  main:               ;  ST0   ST1   ST2   ST3
7      fld dword [X]   ; 10.0    ?     ?     ?  ;do ST0 nacti 32b promennou X
8      fld dword [Y]   ; 20.0  10.0    ?     ?  ;do ST0 nacti 32b promennou Y
9      fld st1         ; 10.0  20.0  10.0    ?  ;do ST0 nacti registr ST1
10     ret
```

Pro čtení do a vypisování z ST0 používáme funkce ReadDouble a WriteDouble:

- ReadFloat a WriteFloat nepoužíváme, protože nepracují s ST0 ale s EAX.

```
11 call ReadDoubleNewLine  ; do ST0 nacti 64b desetinne cislo ze vstupu
12 call WriteDoubleNewLine ; ST0 vypis na vystup
```

Základní aritmetické operace

Základní aritmetické operace spočítáme instrukcemi FADD, FSUB, FMUL a FDIV:

- Základní aritmetické operace mohou mít jeden nebo dva operandy.
- Pokud jsou operandy dva, první z nich cílový, druhý z nich je zdrojový.

```
1  fadd st0, st1        ; ST0 = ST0 + ST1   ;Float ADDition       (scitani)
2  fsub st0, st1        ; ST0 = ST0 - ST1   ;Float SUBtraction    (odcitani)
3  fmul st0, st1        ; ST0 = ST0 * ST1   ;Float MULtiplication (nasobeni)
4  fdiv st0, st1        ; ST0 = ST0 / ST1   ;Float DIVision       (deleni)
```

- Pokud jsou operandy dva, oba musejí být registry a alespoň jeden musí být ST0.

```
5  fmul st0, st1        ; ST0 = ST0 * ST1
6  fmul st1, st0        ; ST1 = ST1 * ST0
7  fmul st1, st1        ; CHYBA - ani jeden z operandu neni ST0
```

- Pokud je operand jen jeden, tak cílovým operandem je implicitně registr ST0, a jako zdrojový operand můžeme použít registr nebo paměť.

```
8  fmul st1             ; ST0 = ST0 * ST1
9  fmul dword [X]       ; ST0 = ST0 * [X]
10 fmul st0, dword [X]  ; CHYBA - dva operandy - oba museji byt registr
```

Program spočítá a vypíše součet dvou desetinných čísel ze vstupu:

```
%include 'rw32.inc'           ;knihovna pro vstup a vystup
section .text                 ;kodovy segment
main:                         ;   ST0    ST1    ST2
    call ReadDoubleNewLine    ;    X      ?      ?    ;nacti prvni cislo
    call ReadDoubleNewLine    ;    Y      X      ?    ;nacti druhe cislo
    fadd st0, st1             ;   Y+X     X      ?    ;ST0 = ST0 + ST1
    call WriteDoubleNewLine   ;   Y+X     X      ?    ;vypis ST0
    ret
```

Program spočítá a vypíše součet dvou desetinných čísel z paměti:

```
%include 'rw32.inc'           ;knihovna pro vstup a vystup
section .data                 ;inicializovany segment
    X dd 10.0                 ; 32b desetinne cislo X
    Y dd 20.0                 ; 32b desetinne cislo Y
section .text                 ;kodovy segment
main:                         ;   ST0    ST1    ST2
    fld  dword [X]            ; 10.0     ?      ?    ;do ST0 nacti X
    fadd dword [Y]            ; 30.0     ?      ?    ;ST0 = ST0 + [Y]
    call WriteDoubleNewLine   ; 30.0     ?      ?    ;vypis ST0
    ret
```

Pokročilé aritmetické operace

Pokročilé operace spočítáme instrukcemi FCHS, FABS, FSIN, FCOS, FSQRT, atd.:

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

```
1 fchs      ; ST0 =  neg(ST0)  ;Float CHange Sign     (negace)
2 fabs      ; ST0 =  abs(ST0)  ;Float ABSolute value (absolutni hodnota)
3 fsin      ; ST0 =  sin(ST0)  ;Float SINe           (sinus)
4 fcos      ; ST0 =  cos(ST0)  ;Float COSine         (kosinus)
5 fsqrt     ; ST0 = sqrt(ST0)  ;Float SQare RooT     (odmocnina)
```

Vyzkoušejte si:

- Program spočítá a vypíše sinus z hodnoty 1.0 (radián).

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .data                ;inicializovany segment
    X dd 1.0                 ; 32b desetinne cislo X
section .text                ;kodovy segment
main:                        ;  ST0   ST1   ST2
    fld dword [X]            ;  1.0    ?     ?  ;nacti 32b promennou X
    fsin                     ;  0.84   ?     ?  ;ST0 = sin(ST0)
    call WriteDoubleNewLine  ;  0.84   ?     ?  ;vypis ST0
    ret
```

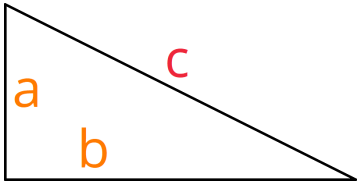
Vyzkoušejte si:

- Ze vstupu načtěte dvě čísla představující délku odvěsen pravoúhlého trojúhelníku.
- Pomocí Pythagorovy věty spočítejte délku jeho přepony (c) a jeho obvod (o).

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

Například:

- 3.0, 4.0 => c = 5.00000, o = 12.00000
- 2.5, 6.0 => c = 6.50000, o = 15.00000
- 4.8, 5.5 => c = 7.30000, o = 17.60000



Nápověda:

- Mocninu spočítáme násobením FMUL (dva operandy).
- Odmocninu spočítáme instrukcí FSQRT (bez operandů).
- Kopii hodnoty vytvoříme instrukcí FLD (jeden operand).

Pro kopírování desetinných čísel používáme instrukci FST (Float STore):

- Instrukce má jen jeden (cílový) operand – zdrojovým je vždy ST0.
- Cílovým operandem může být pameť nebo **NEPRÁZDNÝ** registr.
- Při ukládání do paměti vždy musíme použít pseudo-instrukci pro velikost.

```
1 fst  st1          ; ST1 = ST0      ;ST0 uloz do registru ST1
2 fst  dword [X]    ; [X] = ST0      ;ST0 uloz do 32b promenne X
3 fst  [X]          ; CHYBA - operation size not specified
```

Pro výměnu desetinných čísel používáme instrukci FXCH (Float eXCHange):

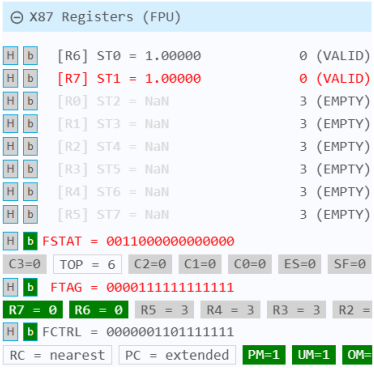
- Instrukce má jen jeden (cílový) operand – zdrojovým je vždy ST0.
- Cílovým operandem může být pouze **NEPRÁZDNÝ** registr.
- Při ukládání do paměti vždy musíme použít pseudo-instrukci pro velikost.

```
4 fxch st1          ; swap(ST0,ST1) ;vymen ST0 a ST1
5 fxch dword [X]    ; CHYBA - vymenovat muzeme pouze registry
```

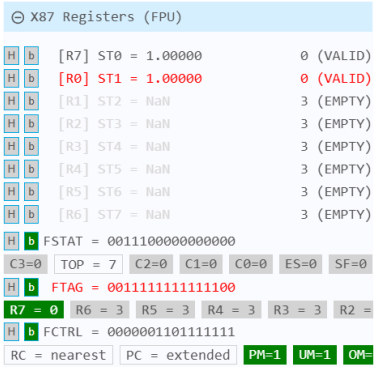
Pozor, instrukcí FST hodnotu **NESMÍME** zkopírovat do prázdného registru:

- Instrukce hodnotu označí jako platnou v registru FTAG, ale neaktualizuje FSTAT!
- Kopii vždy **MUSÍME** vytvořit znovu-načtením hodnoty instrukcí FLD.

```
1  fld  dword [X]  ; do ST0 nacti X
2  fld  st0        ; do ST0 nacti ST0
3                  ; FSTAT je ok
```



```
4  fld  dword [X]  ; do ST0 nacti X
5  fst  st1        ; ST1 = ST0
6                  ; CHYBA v FSTAT
```



Program vymění dvě desetinná čísla v paměti:

```
%include 'rw32.inc'      ;knihovna pro vstup a vystup

section .data            ;inicializovany segment
    X dd 10.0            ; 32b desetinne cislo X
    Y dd 20.0            ; 32b desetinne cislo Y

section .text            ;kodovy segment
main:                    ;  ST0   ST1   ST2   ST3
    fld dword [X]        ; 10.0    ?     ?     ?    ;nacti 32b promennou X
    fld dword [Y]        ; 20.0  10.0    ?     ?    ;nacti 32b promennou Y

    fst dword [X]        ; 20.0  10.0    ?     ?    ; X = ST0
    fxch st1             ; 10.0  20.0    ?     ?    ; swap(ST0,ST1)
    fst dword [Y]        ; 10.0  20.0    ?     ?    ; Y = ST0

    ret
```

Pro mazání desetinných čísel používáme instrukci FSTP (Float STore Pop):

- Přidání přípony P (Pop) za instrukci FST (Float STore), po provedení instrukce (zkopírování do cílového registru) smaže obsah registru ST0.

```
1 fstp st0         ; nejprve ST0 = ST0      , pak odstran ST0
2 fstp st1         ; nejprve ST1 = ST0      , pak odstran ST0
3 fstp dword [X]   ; nejprve [X] = ST0      , pak odstran ST0
```

Příponu P (Pop) můžeme přidat i za základní aritmetické instrukce (viz. přednáška):

- Instrukce musejí mít dva operandy, první musí být registr a druhý musí být ST0.

```
4 faddp st1, st0   ; nejprve ST1 = ST1+ST0 , pak odstran ST0
5 fsubp st1, st0   ; nejprve ST1 = ST1-ST0 , pak odstran ST0
6 fmulp st1, st0   ; nejprve ST1 = ST1*st0 , pak odstran ST0
7 fdivp st1, st0   ; nejprve ST1 = ST1/st0 , pak odstran ST0
```

Jak bude vypadat obsah registrů po provedení následujících instrukcí?

```
%include 'rw32.inc'      ;knihovna pro vstup a vystup

section .data            ;inicializovany segment
    X dd 10.0            ; 32b desetinne cislo X
    Y dd 20.0            ; 32b desetinne cislo Y
    Z dd 30.0            ; 32b desetinne cislo Z

section .text            ;kodovy segment
main:                    ;  ST0   ST1   ST2   ST3
    fld dword [X]        ; 10.0    ?     ?     ?    ;nacti 32b promennou X
    fld dword [Y]        ; 20.0  10.0    ?     ?    ;nacti 32b promennou Y
    fld dword [Z]        ; 30.0  20.0  10.0    ?    ;nacti 32b promennou Z

    fstp st0             ; 20.0  10.0    ?     ?    ;ST0 = ST0, smaz ST0
    fstp st1             ; 20.0    ?     ?     ?    ;ST1 = ST0, smaz ST0
    fstp st0             ;   ?     ?     ?     ?    ;ST0 = ST0, smaz ST0

    ret
```

Vyzkoušejte si:

- Zkopírujte si šablonu a spočítejte diskriminant a kořeny ryze kvadratické rovnice.
- Všechny výsledky uložte do paměti a zkontrolujte si je v debuggeru.

$$D = -4AC \qquad X_1 = \frac{+\sqrt{D}}{2A} \qquad X_2 = \frac{-\sqrt{D}}{2A}$$

Například:

- A = 2.0, C = -4.5 => X_1 = 1.50000, X_2 = -1.50000

Nápověda:

- Násobení dvěma implementujeme součtem FADD.
- Negaci implementujeme aritmetickou operací FCHS.
- Adresování položek pole je stejné jako pro celá čísla (viz. cv03).

```
%include 'rw32.inc'      ;knihovna pro vstup a vystup

section .data            ;datovy segment
    A dd 2.0             ; 32b desetinne cislo A
    C dd -4.5            ; 32b desetinne cislo C
    D dd 0.0             ; 32b desetinne cislo D
    X dd 0.0, 0.0        ; pole dvou 32b desetinnych cisel X

section .text            ;kodovy segment
main:

    ;???

    ret
```

Desetinné konstanty

Desetinné konstanty

Instrukce FLD neumí načítat obecné konstanty ani hodnoty z registrů CPU:

```
1 fld  10.0      ; CHYBA - nacitani obecne konstanty
2 fld  eax       ; CHYBA - nacitani z EAX (registr CPU)
```

Instrukce FLD umí načíst tři specílní konstanty 0.0 (FLDZ), 1.0 (FLD1) a π (FLDPI):

```
3 fldz           ; Float LoaD Zero     ;nacti konstantu 0.0
4 fld1           ; Float LoaD 1        ;nacti konstantu 1.0
5 fldpi          ; Float LoaD PI       ;nacti konstantu 3.14159...
```

Obecnou konstantu v vytvoříme pomocí a lokálních proměnných (viz. cv09):

- Na začátku funkce vytvoříme rámec, a alokujeme lokální proměnnou.
- Pomocí makra __float32__() do proměnné uložíme obecnou konstantu.
- Instrukcí FLD lokální proměnnou načteme do ST0.
- Na konci funkce lokální proměnnou uvolníme a zrušíme zásobníkový rámec.

```
6 mov  dword [ebp-4], __float32__(0.5) ; do 32b lokalni promenne
7                                      ; uloz desetinne cislo 0.5
8 fld  dword [ebp-4]                   ; nacti hodnotu z lokalni promenne
```

Program vytvoří a načte několik různých konstant:

- Při ukládání do paměti musíme použít pseudo-instrukci pro velikost.

```
%include 'rw32.inc'      ;knihovna pro vstup a vystup
section .text            ;kodovy segment
main:
    push ebp             ; zalohuj stare dno
    mov  ebp, esp        ; vytvor nove dno
    sub  esp, 4          ; alokuj misto pro jednu 32b lokalni promennou

    mov  dword [ebp-4], __float32__(0.5) ; do 32b lokalni promenne
                                         ; uloz desetinne cislo 0.5

                         ; ST0   ST1   ST2   ST3
    fldz                 ; 0.0    ?     ?     ?   ;nacti 0.0
    fld1                 ; 1.0   0.0    ?     ?   ;nacti 1.0
    fldpi                ; 3.141 1.0   0.0    ?   ;nacti PI
    fld  dword [ebp-4]   ; 0.5   3.141 1.0   0.0   ;nacti lokalni promennou

    mov  esp, ebp        ; uvolni vsechny lokalni promenne
    pop  ebp             ; obnov stare dno
    ret                  ; konec funkce
```

Vyzkoušejte si:

- Zkopírujte si šablonu, a bez vytváření dalších globálních proměnných spočítejte hodnotu zlatého řezu definovaného následující rovnicí.
- Výsledek uložte do proměnné GR, smažte obsah všech registrů FPU a její hodnotu si pak zobrazte v debuggeru.

$$GR = \frac{1.0 + \sqrt{5.0}}{2.0} = 1.61803...$$

Nápověda:

- První lokální proměnná je na adrese [EBP-4], druhá na adrese [EBP-8] (viz. cv09).
- Pro konstantu 1.0 místo lokální proměnné můžeme použít instrukci FLD1.
- U proměnné GR musíme v debuggeru nastavit správný datový typ.

```
%include 'rw32.inc'      ;knihovna pro vstup a vystup

section .bss             ;neinicializovany segment
    GR resd 1            ; neinicializovane 32b cislo GR

section .text            ;kodovy segment
main:
    push ebp             ; zalohujeme stare dno
    mov  ebp, esp        ; vytvarime nove dno
    sub  esp, 8          ; alokujeme misto pro dve 32b lokalni promenne

    ; ???

    mov  esp, ebp        ;uvolnujeme vsechny lokalni promenne
    pop  ebp             ;obnovujeme stare dno
    ret
```

Desetinné parametry

Matematický koprocesor (FPU) neumí pracovat s registry procesoru (CPU):

- Koprocesor neumí provádět skoky (JMP, JE, ...) ani cykly (LOOP).
- Koprocesor neumí pracovat se zásobníkem (PUSH, POP).

Při psaní programu proto musíme kombinovat instrukce obou zařízení:

- Cykly a podmínky a předávání parametrů implementujeme na procesoru.
- Výpočty s desetinnými čísly implementujeme pomocí koprocesoru.

Desetinná čísla mohou být předána přes zásobník jako parametr funkce:

- Obsah registrů ST1 až ST7 na konci funkce mažeme vždy.
- Pokud návratovou hodnotu je desetinné číslo, vracíme ho v ST0 (ne v EAX).
- Pokud návratovou hodnotu není desetinné číslo tak mažeme i ST0.

Funkce sečte dvě 32b desetinná čísla (X a Y) předaná přes zásobník:

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .text                ;kodovy segment
main:
    push dword __float32__(20.0) ; predej druhy parametr (Y)
    push dword __float32__(10.0) ; predej prvni parametr (X)
    call soucet              ; zavolej funkci soucet
    add  esp, 8              ; odstran predane parametry (CDECL)
    call WriteDouble         ; vypis vystup z ST0 (navratova hodnota)
    fstp st0                 ; smaz navratovou hodnotu
    ret

soucet:
    push ebp                 ; zalohuj stare dno
    mov  ebp, esp            ; vytvor nove dno
                             ; ST0  ST1
    fld  dword [ebp +12]     ;  Y    ?
    fld  dword [ebp + 8]     ;  X    Y
    fadd st0, st1            ; X+Y   Y
    fstp st1                 ; X+Y   ?
    pop  ebp                 ; obnov stare dno
    ret
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementuje funkci SumaCisel, která jako parametry dostane adresu pole desetinných čísel a jeho délku, a spočítá jejich sumu.
- Parametry funkce jsou předané dle konvence CDECL, návratovou hodnotu funkce vrací v registru ST0.

Například:

- arr1, 3 => 11.1
- arr2, 10 => 5.5

Nápověda:

- Adresa a délka pole jsou celá čísla, položky pole jsou desetinná čísla.
- Cyklus implementujeme instrukcí LOOP, před začátkem cyklu si musíme nastavit adresu, počet opakování, a počáteční hodnotu sumy.
- Pro načítání hodnot nemůžeme používat řetězové instrukce, adresu položek pole si musíme počítat sami (viz. cv07).

```
%include 'rw32.inc'           ;knihovna pro vstup a vystup
section .data                 ;datovy segment
    arr1 dd 0.1, 1.0, 10.0    ; pole tri 32b desetinnych cisel
    arr2 dd 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

section .text                 ;kodovy segment
main:
    push dword 3              ; predavame delku  pole (druhy parametr)
    push arr1                 ; predavame adresu pole (prvni parametr)
    call SumaCisel            ; volame funkci SumaCisel
    add  esp, 8               ; predane parametry mazeme ze zasobniku
    call WriteDoubleNewLine   ; vypisujeme navratovou hodnotu
    fstp st0                  ; navratovou hodnotu mazeme z ST0

    push dword 10             ; predavame jinou delku
    push arr2                 ; predavame jine pole
    call SumaCisel            ; volame funkci SumaCisel
    add  esp, 8               ; predane parametry mazeme ze zasobniku
    call WriteDoubleNewLine   ; vypisujeme navratovou hodnotu
    fstp st0                  ; navratovou hodnotu mazeme z ST0
    ret
```

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci Odmocnina, která provedením deseti iterací babylónské metody spočítá odmocninu **BEZ** instrukce FSQRT.
- Vstupem funkce je 32b desetinné číslo X, předané dle konvence STDCALL, návratovou hodnotou je Y_{10} předané v registru ST0, které se rovná $\sqrt{X}$.
- Pro jednoduchost předpokládejte že X je kladné číslo.
- Babylonská metoda je definována iterační rovnicí:

$$Y_0 = X \qquad Y_{i+1} = \frac{1.0}{2.0}\left(\frac{X}{Y_i} + Y_i\right)$$

Například:

- 4.0 => 2.00000
- 2.0 => 1.41421
- 10.0 => 3.16228

Nápověda:

- Před začátkem cyklu do FPU nahrajeme všechny potřebné konstanty a Y_0.
- Na konci každé iterace nám v FPU musejí zůstat pouze konstanty a Y_{i+1}.
- Počet načítaných (FLD) a mazaných (FSTP) hodnot uvnitř cyklu se musí shodovat.

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup

section .text                ;kodovy segment
main:
    push __float32__(4.0)    ; predej parametr (4.0)
    call Odmocnina           ; zavolej funkci
    call WriteDoubleNewLine  ; vypis vystup z ST0  (2.0)
    fstp st0                 ; odstran ST0

    push __float32__(2.0)    ; predej parametr (2.0)
    call Odmocnina           ; zavolej funkci
    call WriteDoubleNewLine  ; vypis vystup z ST0 (1.41421)
    fstp st0                 ; odstran ST0

    push __float32__(10.0)   ; predej parametr (10.0)
    call Odmocnina           ; zavolej funkci
    call WriteDoubleNewLine  ; vypis vystup z ST0 (3.16228)
    fstp st0                 ; odstran ST0

    ret
```

$$
\begin{aligned}
X &= 4.0 \\
Y_0 &= X = 4.0 \\
Y_1 &= \frac{1.0}{2.0}\left(\frac{X}{Y_0} + Y_0\right) = \frac{1.0}{2.0}\left(\frac{4.0}{4.0} + 4.0\right) = 2.5 \\
Y_2 &= \frac{1.0}{2.0}\left(\frac{X}{Y_1} + Y_1\right) = \frac{1.0}{2.0}\left(\frac{4.0}{2.5} + 2.5\right) = 2.05 \\
Y_3 &= \frac{1.0}{2.0}\left(\frac{X}{Y_2} + Y_2\right) = \frac{1.0}{2.0}\left(\frac{4.0}{2.05} + 2.05\right) = 2.00061 \\
&\vdots \\
Y_{10} &= \frac{1.0}{2.0}\left(\frac{X}{Y_9} + Y_9\right) = \frac{1.0}{2.0}\left(\frac{4.0}{2.0} + 2.0\right) = 2.0 = \sqrt{X}
\end{aligned}
$$