Floating Point Instructions

Ned Nedialkov

McMaster University
Canada

SE 3F03
March 2013

Outline

Storing data

- **fld** `src`
  - pushes `src` onto the FPU stack
  - decrements TOS
  - `src` can be register or memory
  - if `src` is in memory, single or double, it is extended to 80 bits
- **fild** `src`
  - 16 or 32 bit integer in memory is converted to extended precision and stored at ST0

- **fst** `dest`
  - stores ST0 at `dest`
  - `dest` can be a FP register or memory
  - does not remove value from the stack
- `fstp dest` same as **fst** but pops from the stack after copying
- **fist** `dest` stores ST0 as a signed integer
- **fistp** `dest` stores ST0 as a signed integers and pops after storing

Addition

- **fadd** `src`
  - `ST0 = ST0 + src`
- **fadd** `dest, src`
  - `dest = dest+src`
  - `dest, src` must be FP registers
- **faddp** `dest, src`
  - `dest = dest+src`
  - `dest, src` must be FP registers
  - pops the stack

Subtraction

- **fsub** `src`
  - `ST0 = ST0 - src`
- **fsubr** `src`
  - `ST0 = src - ST0`
  - reverse subtraction
- **fsub** `dest, src`
  - `dest = dest-src`
  - `dest, src` must be FP registers
- **fsubp** `dest, src`
  - `dest = dest+src`
  - `dest, src` must be FP registers
  - pops the stack

Multiplication

- **fmul** `src`
  - `ST0 = ST0*src`
- **fmul** `dest, src`
  - `dest = dest*src`
- **fmulp** `dest, src`
  - `dest = dest*src`
  - pops the stack
- **fmulp**
  - `ST0 = ST0*ST1`
  - pops the stack

Division

- **fdiv** `src`
  - `ST0 = ST0/src`
- **fdiv** `dest, src`
  - `dest = dest/src`
- **fdivr** `src`
  - `ST0 = src/ST0`
- Similarly, there are division “pop” instruction.

Comparison instructions

- **fcom** `src`
  - compares `ST0` and `src`

	C3	C2	C0
ST0>src	0	0	0
ST0=src	1	0	0
ST0<src	0	0	1
not comparable	1	1	1

- **fcom**
  - compares `ST0` and `ST1`
- **ftst**
  - compares `ST0` and `0.0`

Some more instructions

- **fchs**
  - changes the sign of `ST0`
- `abs`
  - `ST0 = |ST0|`
- **fldcw** `src`
  - loads 16 bit from memory into the FPU control register
- **fstcw** `dest`
  - store the control register in memory

Example

Compute the real roots of $ax^2 + bx + c = 0$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

From http://www.drpaulcarter.com/pcasm/

```
;   int quadratic( double a, double b, double c,
;                  double * root1, double *root2 )
;   a, b, c - coefficients of powers of quadratic equation (see above)
;   root1   - pointer to double to store first root in
;   root2   - pointer to double to store second root in
; Return value:
;   returns 1 if real roots found, else 0
%define a               qword [ebp+8]
%define b               qword [ebp+16]
%define c               qword [ebp+24]
%define root1           dword [ebp+32]
%define root2           dword [ebp+36]
%define disc            qword [ebp-8]
%define one_over_2a     qword [ebp-16]
segment .data
MinusFour       dw      -4
segment .bss
segment .text
        global  quadratic
quadratic:
        push    ebp
```

```
mov     ebp, esp
sub     esp, 16            ; allocate for disc & one_over_2a
push    ebx                ; must save original ebx
fild    word [MinusFour]; stack -4
fld     a                  ; stack: a, -4
fld     c                  ; stack: c, a, -4
fmulp   st1                ; stack: a*c, -4
fmulp   st1                ; stack: -4*a*c
fld     b
fld     b                  ; stack: b, b, -4*a*c
fmulp   st1                ; stack: b*b, -4*a*c
faddp   st1                ; stack: b*b - 4*a*c
ftst                       ; test with 0
fstsw   ax
sahf
jb      no_real_solutions ; if disc < 0, no real solutions
fsqrt                      ; stack: sqrt(b*b - 4*a*c)
fstp    disc               ; store and pop stack
fld1                       ; stack: 1.0
fld     a                  ; stack: a, 1.0
fscale                     ; stack: a * 2^(1.0) = 2*a, 1
fdivp   st1                ; stack: 1/(2*a)
fst     one_over_2a        ; stack: 1/(2*a)
```

```
        fld     b                   ; stack: b, 1/(2*a)
        fld     disc                ; stack: disc, b, 1/(2*a)
        fsubrp  st1                 ; stack: disc - b, 1/(2*a)
        fmulp   st1                 ; stack: (-b + disc)/(2*a)
        mov     ebx, root1
        fstp    qword [ebx]         ; store in *root1
        fld     b                   ; stack: b
        fld     disc                ; stack: disc, b
        fchs                        ; stack: -disc, b
        fsubrp  st1                 ; stack: -disc - b
        fmul    one_over_2a         ; stack: (-b - disc)/(2*a)
        mov     ebx, root2
        fstp    qword [ebx]         ; store in *root2
        mov     eax, 1              ; return value is 1
        jmp     short quit
no_real_solutions:
        ffree   st0                 ; dump disc off stack
        mov     eax, 0              ; return value is 0
quit:
        pop     ebx
        mov     esp, ebp
        pop     ebp
        ret
```