Introduction to 8086 Assembly

Lecture 20

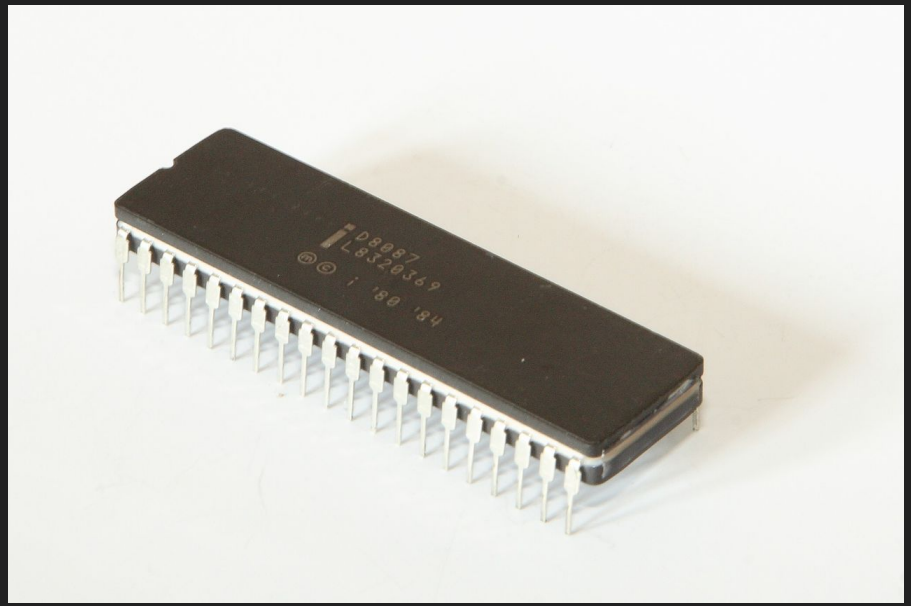
x86 floating point

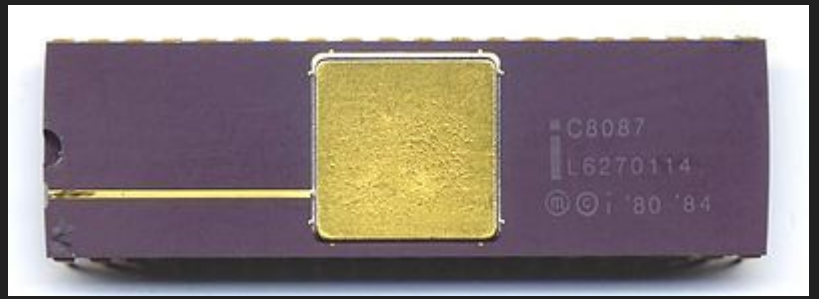
Floating point operations

- Floating point libraries
- Math coprocessor (floating point coprocessor, floating point unit FPU)
  - add-on
  - Integrated

Intel 8087 coprocessor

K. N. Toosi
University of Technology

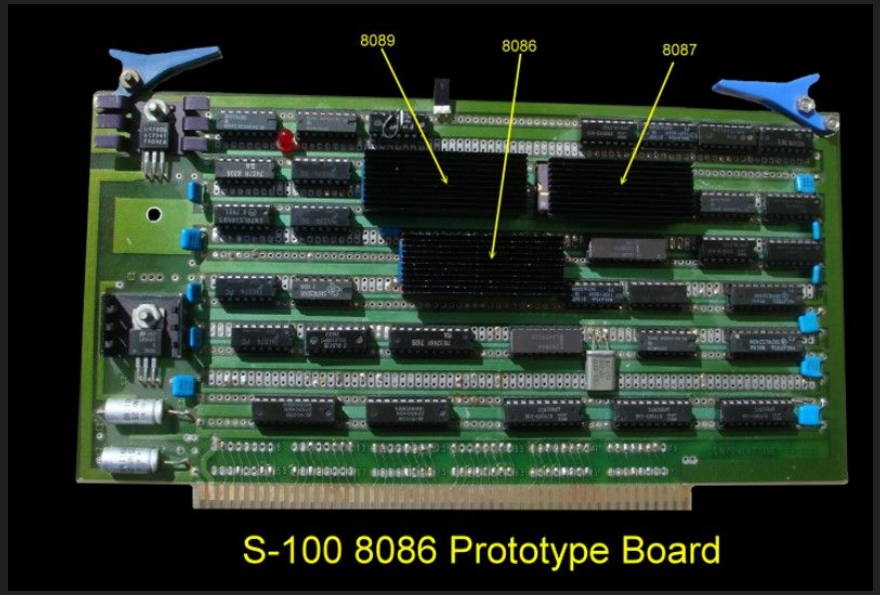




https://en.wikipedia.org/wiki/Intel_8087

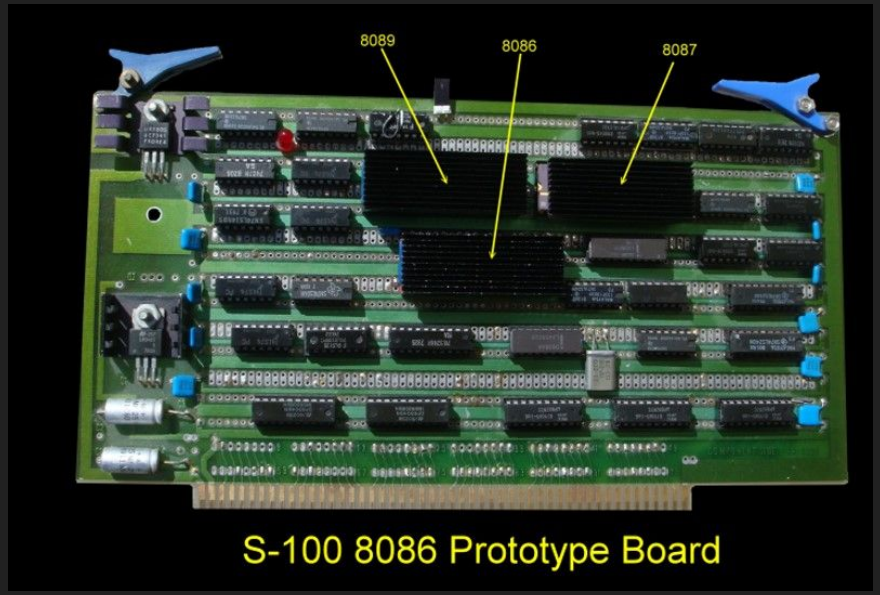
Intel 8087 coprocessor

K. N. Toosi
University of Technology



http://www.s100computers.com/My%20System%20Pages/8086%20Board/8086%20CPU%20Board.htm

Intel 8087 coprocessor



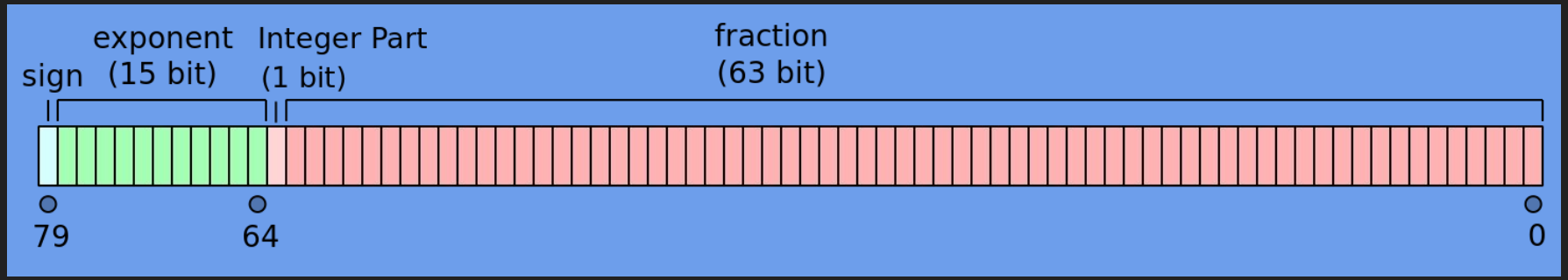
http://www.s100computers.com/My%20System%20Pages/8086%20Board/8086%20CPU%20Board.htm

http://7review.com/remembering-maths-coprocessor/

K. N. Toosi
University of Technology

Intel x87 extended precision

K. N. Toosi
University of Technology

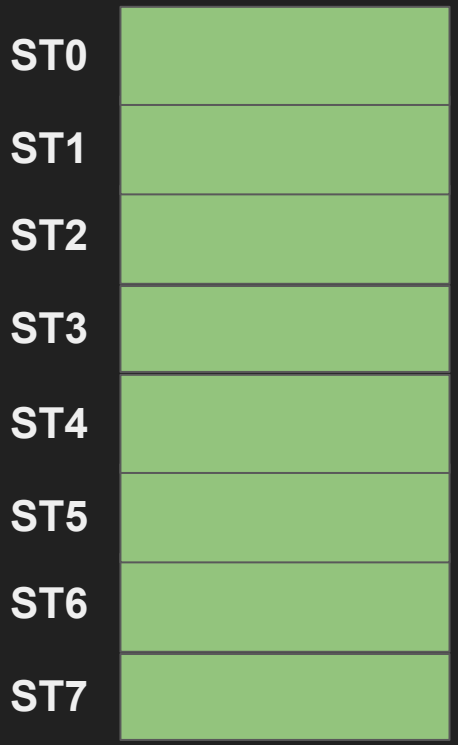


https://en.wikipedia.org/wiki/Extended_precision v

8087 register stack

- ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7

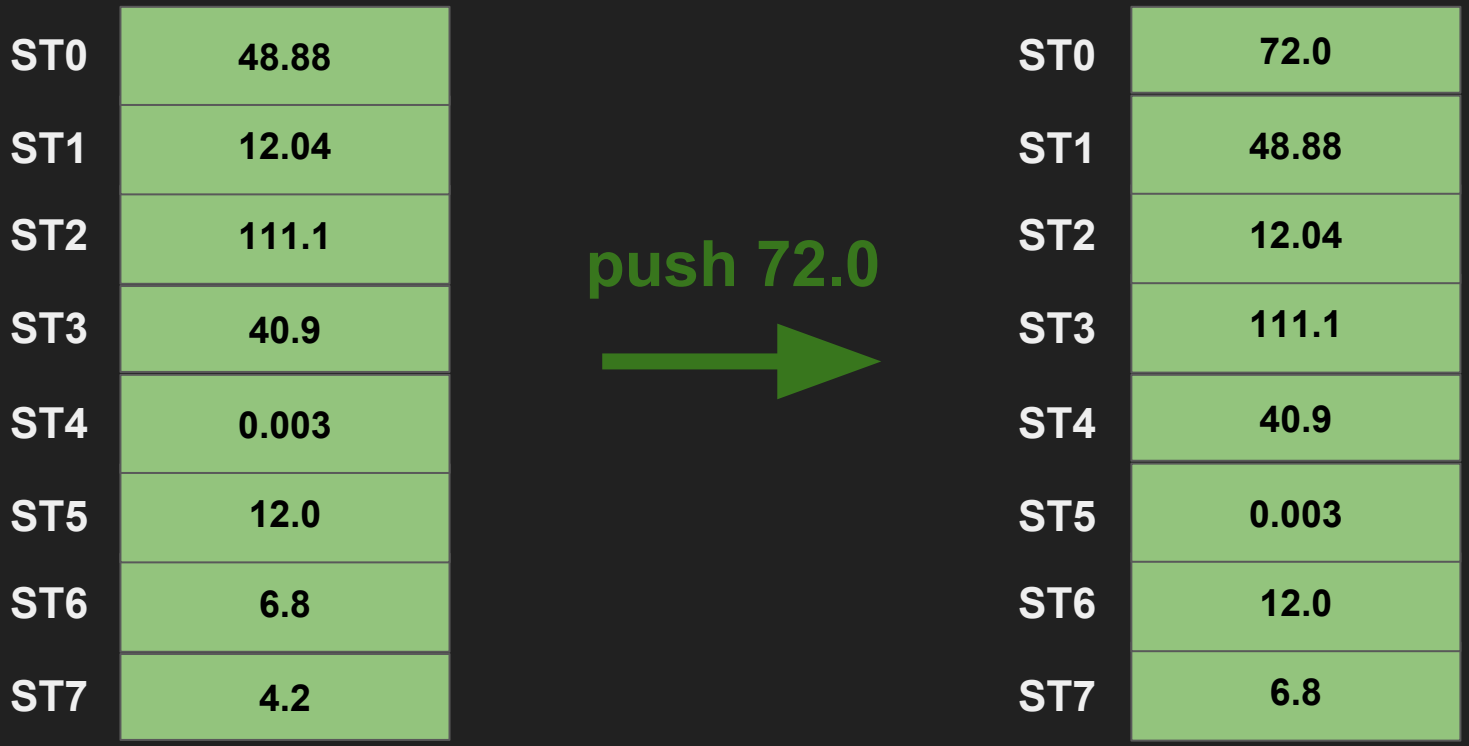
8087 register stack



8087 register stack

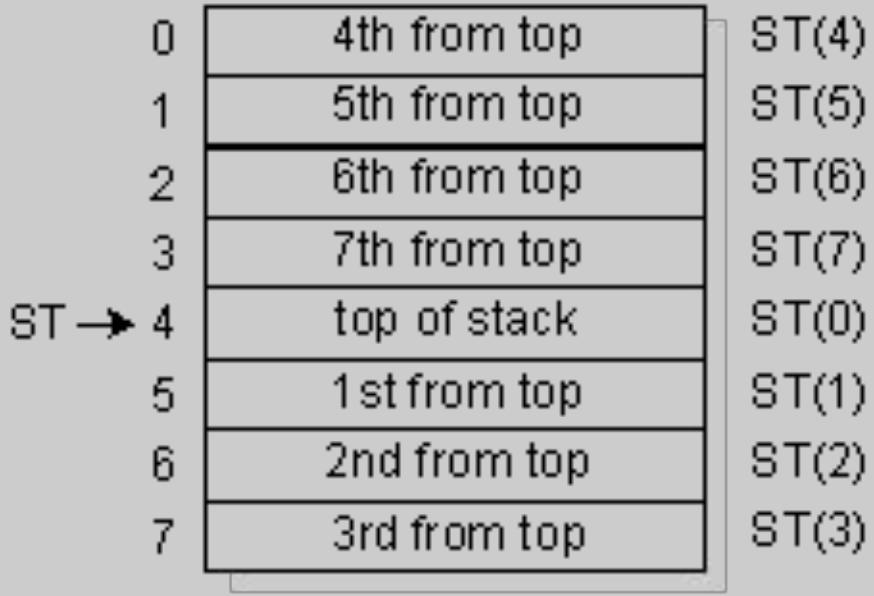
ST0	48.88
ST1	12.04
ST2	111.1
ST3	40.9
ST4	0.003
ST5	12.0
ST6	6.8
ST7	4.2

8087 register stack



K. N. Toosi
University of Technology

8087 register stack



https://users.pja.edu.pl/~jms/qnx/help/watcom/compiler16/pragma16.html

Floating points in memory

K. N. Toosi
University of Technology

single precision

```
segment .data
l1:     dd  1.0, 18.9, 0.00001, 1e-14, 12.0
l2:     dq  1.0, 18.9, 0.00001, 1e-14, 12.0
```

double precision

Floating points in memory

K. N. Toosi
University of Technology

32 bit integer

```
l1:     dd  1.0, 18.9, 0.00001, 1e-14, 12
l2:     dq  1.0, 18.9, 0.00001, 1e-14, 12.0
```

double precision

Example


```
segment .data

l1:     dd  1.0, 18.9, 0.00001, 1e-14, 12.0

l2:     dq  1.0, 18.9, 0.00001, 1e-14, 12.0
```

```
  fld dword [l1]          ; ST0 = [l1]
  fld dword [l1+4]        ; ST0 = [l1+4]

  fadd ST1                ; ST0 += ST1

  fist dword [l1]         ; [l1] = (int) ST0

  mov eax, [l1]
  call print_int
  call print_nl
```

Example


```
segment .data

l1:     dd  1.0, 18.9, 0.00001, 1e-14, 12.0

l2:     dq  1.0, 18.9, 0.00001, 1e-14, 12.0
```

```
  fld dword [l1]          ; ST0 = [l1]
  fld dword [l1+4]        ; ST0 = [l1+4]

  fadd ST1                ; ST0 += ST1

  fist dword [l1]         ; [l1] = (int) ST0

  mov eax, [l1]
  call print_int
  call print_nl
```

```
b.nasihatkon@kntu:lecture20$ ./run.sh float1
20
```

Loading (pushing)

`FLD mem32 (/mem64/mem80)`	`push ST0 <- mem32 (so on)`
`FLD STi`	`push ST0 <- STi` `STi: ST0, ST1, … or ST7`
`FILD mem16 (/mem32/mem64)`	`push ST0 <- int2float(mem32)`

Example:

```
FLD dword [l1]
FLD qword [l2]
```

Loading (pushing) constants

FLD1	push ST0 <- 1.0
FLDZ	push ST0 <- +0.0
FLDPI	push ST0 <- π (the pi number)
FLDL2T/FLDL2E FLDLG2/FLDLN2	

Storing

`FST mem32/mem64`	`mem32 <- ST0`
`FST STi`	`STi <- ST0  (i=0,1,...,7)`
`FIST mem16/mem32/mem64`	`mem32 <- float2int(ST0)`
`FSTP dest` `FISTP dest`	similar to FSTP and FISTP but also pops top of stack

Example:

```
FST dword [l1]
FST qword [l2]
```

Exchange

`FXCH  STi`	`ST0 <-> STi  (i=0,1,...,7)`

Math operations

`FADD src`	`ST0 += src`
`FSUB src`	`ST0 -= src`
`FMUL src`	`ST0 *= src`
`FDIV src`	`ST0 /= src`
	`src: STi/mem32/mem64`

Math operations

`FADD src`	`ST0 += src`
`FSUB src`	`ST0 -= src`
`FMUL src`	`ST0 *= src`
`FDIV src`	`ST0 /= src`
`FSUBR src`	`ST0 = src - ST0`
`FDIVR src`	`ST0 = src / ST0`
	`src: STi/mem32/mem64`

Math operations

FADDP STi	STi += ST0
FSUBP STi	STi -= ST0
FMULP STi	STi *= ST0
FDIVP STi	STi /= ST0
FSUBRP STi	STi = ST0 - STi
FDIVRP STi	STi = ST0 / STi
	And pop top of stack STi: ST0, ST1, … or ST7

Math operations

`FADDP STi` `FADDP STi, ST0`	`STi += ST0`
`FSUBP STi` `FSUBP STi, ST0`	`STi -= ST0`
`FMULP STi` `FMULP STi, ST0`	`STi *= ST0`
`FDIVP STi` `FDIVP STi, ST0`	`STi /= ST0`
`FSUBRP STi` `FSUBRP STi, ST0`	`STi = ST0 - STi`
`FDIVRP STi` `FDIVRP STi, ST0`	`STi = ST0 / STi`
	And pop top of stack

Math operations

`FADD STi, ST0`	`STi += ST0`
`FSUB STi, ST0`	`STi -= ST0`
`FMUL STi, ST0`	`STi *= ST0`
`FDIV STi, ST0`	`STi /= ST0`
`FSUBR STi, ST0`	`STi = ST0 - STi`
`FDIVR STi, ST0`	`STi = ST0 / STi`
	`STi: ST0, ST1, … or ST7`

Math operations

FIADD src	ST0 += int2float(src)
FISUB src	ST0 -= int2float(src)
FIMUL src	ST0 *= int2float(src)
FIDIV src	ST0 /= int2float(src)
FISUBR src	ST0 = int2float(src) - ST0
FIDIVR src	ST0 = int2float(src) / ST0
	src: mem32/mem64

Example


```
segment .data

l1:     dd  1.0, 18.9, 0.00001, 1e-14, 12.0

l2:     dq  1.0, 18.9, 0.00001, 1e-14, 12.0
```

```
  fld dword [l1]          ; ST0 = [l1]
  fld dword [l1+4]        ; ST0 = [l1+4]

  fadd ST1                ; ST0 += ST1

  fist dword [l1]         ; [l1] = (int) ST0

  mov eax, [l1]
  call print_int
  call print_nl
```

Math operations

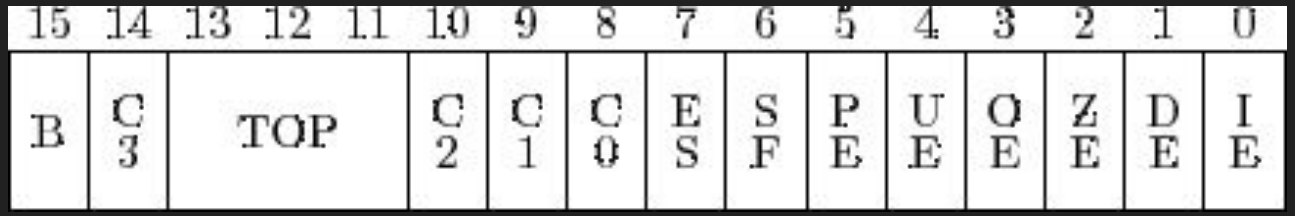
FCHS	ST0 = -ST0
FABS	ST0 = |ST0|
FSQRT	ST0 = sqrt(ST0)
FSCALE	ST0 *= 2^floor(ST1)
FRNDINT	ST0 = round(ST0) still floating point

Math operations

`FSIN`	`ST0 = sin(ST0)`
`FCOS`	`ST0 = cos(ST0)`
`FSINCOS`	`ST0 = cos(ST0)` **then** `ST0 <- push sin(ST0)`

x87 status register

K. N. Toosi
University of Technology



15	14	13 12 11	10	9	8	7	6	5	4	3	2	1	0
B	C3	TOP	C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE

http://www.russinoff.com/libman/text/node48.html

Making comparisons

FCOM src	compare ST0, src src: mem32/mem64/STi
FCOMP src	like FCOM src AND pop top of stack
FCOMPP	Compare ST0, ST1 AND pop twice
FICOM mem16/mem32	compare ST0, int2float(src) src: mem32/mem64/STi
FTST	compare ST0, 0

Making comparisons

`FSTSW mem16`	
`FSTSW AX`	
`SASF`	`FLAGS <- AX` `(not all bits)`
`LASF`	`AX <- FLAGS` `(not all bits)`

Making comparisons

FSTSW mem16	
FSTSW AX	
SASF	FLAGS <- AX (not all bits)
LASF	AX <- FLAGS (not all bits)

```
FSTSW AX
SASF
follows every comparison
```

Directly setting EFLAGS

`FCOMI STi`	`compare ST0, STi` `sets EFLAGS`
`FCOMIP STi`	`FCOMI STi` `pops top of stack`

Practice: roots of a quadric

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4: dq     -4.0
temp:   dq      0.0
format: db   "%f", 10, 0
no_roots_msg: db "No real roots",10, 0

segment .text
        :
```

Practice: roots of a quadric

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4:  dq     -4.0
temp:    dq     0.0
format:  db   "%f", 10, 0
no_roots_msg: db "No real roots",10, 0

segment .text
        :
       fld  qword [b]
       fld  qword [b]
       fmulp ST1
```

Practice: roots of a quadric

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4:  dq     -4.0
temp:    dq     0.0
format:  db   "%f", 10, 0
no_roots_msg: db "No real roots",10, 0

segment .text
        :
        fld  qword [b]
        fld  qword [b]
        fmulp ST1

        fld qword [a]
        fld qword [c]
        fld qword [minus4]
        fmulp ST1
        fmulp ST1
```

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4:  dq     -4.0
temp:    dq     0.0
format:  db   "%f", 10, 0
no_roots_msg: db "No real roots",10, 0

segment .text
        :
        fld  qword [b]
        fld  qword [b]
        fmulp ST1

        fld qword [a]
        fld qword [c]
        fld qword [minus4]
        fmulp ST1
        fmulp ST1
        faddp ST1
        fldz
        fcomip ST1
        ja    no_roots
```

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4:  dq     -4.0
temp:    dq     0.0
format:  db   "%f", 10, 0
no_roots_msg: db "No real roots",10, 0
segment .text
        :
      fld  qword [b]
      fld  qword [b]
      fmulp ST1

      fld qword [a]
      fld qword [c]
      fld qword [minus4]
      fmulp ST1
      fmulp ST1
      faddp ST1

      fldz
      fcomip ST1
      ja    no_roots

      fsqrt

      fld st0
      fld qword [b]
      faddp st1
      fchs

      fld qword [a]
      fld1
      fld1
      faddp
      fmulp
      fdivp ST1             ; ST1 /= ST0

      ; print st0
      fst  qword [temp]
      push dword [temp+4]
      push dword [temp]
      push format
      call printf
      add  esp, 12

      fcomp

      fld qword [b]
      fchs
      faddp

      fld qword [a]
      fld1
      fld1
      faddp
      fmulp

      fdivp ST1             ; ST1 /= ST0

      ; print st0
      fst  qword [temp]
      push dword [temp+4]
      push dword [temp]
      push format
      call printf
      add  esp, 12

      jmp endl

no_roots:
      mov eax, no_roots_msg
      call print_string
endl:
```

quadroots.asm

```
extern printf
segment .data
a:      dq      1.0
b:      dq     -3.0
c:      dq      2.0
minus4:  dq     -4.0
temp:    dq     0.0
format:  db   "%f", 10, 0
no_roots_msg: db "No real
segment .text
        :
        fld  qword [b]
        fld  qword [b]
        fmulp ST1

        fld qword [a]
        fld qword [c]
        fld qword [minus4]
        fmulp ST1
        fmulp ST1
        faddp ST1

        fldz
        fcomip ST1
        ja     no_roots
```

```
        fsqrt

        fld st0
        fld qword [b]
        faddp st1
        fchs

        fld qword [a]
                                  [temp]
                                  [temp+4]
                                  [temp]
                                  t
        push dword [temp]
        push format
        call printf
        add  esp, 12

        fcomp
```

```
        fld qword [b]
        fchs
        faddp

        fld qword [a]
        fld1
        fld1
        faddp
        fmulp
                                  ; ST1 /= ST0

        add  esp, 12

        jmp endl

no_roots:
        mov eax, no_roots_msg
        call print_string
endl:
```

This code is inefficient!
Go through the code and make it more efficient.