

CSCI -112

Introduction to computer Systems

Instructor: Santanu Banerjee

Courtesy: UMBC and JBLearning

Topics

- Floating point formats
- 80x86 floating point architecture
- Floating Point Instructions – Load & Store
- Floating Point Instructions – Arithmetic
- Floating Point Instructions – Other
- Floating point to/from ASCII
- Single instruction, multiple data instruction
- C/C++ with floating point assembly procedure

Floating Point Operations

Courtesy: UMBC and JBLearning

Floating Point Formats

Data Declarations

The screenshot shows a debugger interface with three memory windows and a code editor.

Code Editor:

```
19 fltR4a REAL4 1.0
20 byte1 NYIL 0
21 fltR8a REAL8 -20.4
22 byte2 BYTE 0
23 fltR10a REAL10 128.34
24 byte3 BYTE 0
25 fltR4b REAL10 0.5
```

Memory 1 (Hex View):

Address	0x00007FF72343414C	0x00007FF723434150	0x00007FF723434154	0x00007FF723434158	0x00007FF72343415C	0x00007FF723434160	0x00007FF723434164
0x00007FF72343414C	00 00 80 3F 00 66 66 66 66 66 31 C0 0A D7 A3 79 3D 0A 57 80 06 10 00	00 00 02 00	00 00	00 00	00 00	00 00	00 00

Memory 2 (32-bit Floating Point View):

Address	0x00007FF72343414C	0x00007FF723434150	0x00007FF723434154	0x00007FF723434158
0x00007FF72343414C	1.00000000	2.72006165e+23	2.14576716e-07	-1.51735826e+14
0x00007FF723434150	9.12120360e-33	2.10158324	0.00000000	0.00000000
0x00007FF723434154	0.00000000	0.00000000	0.00000000	0.00000000

Memory 3 (64-bit Floating Point View):

Address	0x00007FF723434151	0x00007FF723434155	0x00007FF723434159	0x00007FF72343415D
0x00007FF723434151	-20.39999999999999	1.9720242359035159e+111	fffff4A...xP	0.0000000000000000
0x00007FF723434155	2.673083639849e-317#DEN	0.0000000000000000	€.0.....	0.0000000000000000
0x00007FF723434159	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000

Load Instructions

Load instructions

- Loads data from memory or stack register to stack top ST
- Mnemonics contain f, l, d and few other letters
- `fld realMemoryOperand`
 - Loads stack top ST with floating point data value
 - Values already on the stack are pushed down
- `fild integerMemoryOperand`
 - Converts integer(16,32,64 bit) value to corresponding fp value that is pushed onto the fp stack (ST get the value)
- `fld st(nbr)`
 - Pushes a copy of `st(nbr)` onto the fp stack (ST get the value)

Stack: clear using finit

- When floating point stack is full, i.e. all 8 registers are populated, any further push will cause fp stack overflow.
 - Attempt will show 1#IND in fp registers(ST(i))
- finit
 - Logically clears the stack
 - Next push will begin from stack top
 - You still may see the old values in the debugger in registers ST(0) ... ST(7), but they are not available.

Load Instructions - Example

```
fltR4a REAL4 1.0
byte1 BYTE 0
fltR8a REAL8 -20.4
byte2 BYTE 0
fltR10a REAL10 128.34
byte3 BYTE 0
fltR4b REAL10 0.0
```

```
dwnum    DWORD 44
wnum     WORD -120
qwnum    QWORD 103
```

Load Instructions - Example

```
fld fltR4a  
fld fltR8a  
fld fltR10a  
  
finit  
  
fld wnum  
fld dnum  
fld qnum  
  
fld ST  
fld ST(1)  
fld ST(5)
```

ST0 = +0.00000000000000e+0000 ST1 = +0.00000000000000e+0000 ST2 = +0.00000000000000e+0000 ST3 = +0.00000000000000e+0000 ST4 = +0.00000000000000e+0000 ST5 = +0.00000000000000e+0000 ST6 = +0.00000000000000e+0000 ST7 = +0.00000000000000e+0000 CTRL = 027F STAT = 0000 TAGS = 0000 EIP = 00000000 EDO = 00000000
ST0 = +1.28340000000000e+0002 ST1 = -2.039999999999998e+0001 ST2 = +1.00000000000000e+0000 ST3 = +0.00000000000000e+0000 ST4 = +0.00000000000000e+0000 ST5 = +0.00000000000000e+0000 ST6 = +0.00000000000000e+0000 ST7 = +0.00000000000000e+0000 CTRL = 027F STAT = 2800 TAGS = 00E0 EIP = 720D127C EDO = 720D415A
ST0 = +0.00000000000000e+0000 ST1 = +0.00000000000000e+0000 ST2 = +0.00000000000000e+0000 ST3 = +0.00000000000000e+0000 ST4 = +0.00000000000000e+0000 ST5 = +1.28340000000000e+0002 ST6 = -2.039999999999998e+0001 ST7 = +1.00000000000000e+0000 CTRL = 037F STAT = 0000 TAGS = 0000 EIP = 00000000 EDO = 00000000
ST0 = +1.03000000000000e+0002 ST1 = +4.40000000000000e+0001 ST2 = -1.20000000000000e+0002 ST3 = +0.00000000000000e+0000 ST4 = +0.00000000000000e+0000 ST5 = +0.00000000000000e+0000 ST6 = +0.00000000000000e+0000 ST7 = +0.00000000000000e+0000 CTRL = 037F STAT = 2800 TAGS = 00E0 EIP = 720D1291 EDO = 720D4027
ST0 = -1.20000000000000e+0002 ST1 = +1.03000000000000e+0002 ST2 = +1.03000000000000e+0002 ST3 = +1.03000000000000e+0002 ST4 = +4.40000000000000e+0001 ST5 = -1.20000000000000e+0002 ST6 = +0.00000000000000e+0000 ST7 = +0.00000000000000e+0000 CTRL = 037F STAT = 1000 TAGS = 00FC EIP = D465129B EDO = D4651027

More Loads - Constants

- `fld1`
 - Pushes 1.0 onto floating point stack
- `fldz`
 - Pushes 0.0 onto fp stack
- `fldpi`
 - Pushes π onto fp stack

Note:
No
operand
is
required.

```
ST0 = +3.1415926535897932e+0000  ST1 = +0.000000000000000e+0000
ST2 = +1.000000000000000e+0000  ST3 = +0.000000000000000e+0000
ST4 = +0.000000000000000e+0000  ST5 = -1.200000000000000e+0002
ST6 = +1.030000000000000e+0002  ST7 = +1.030000000000000e+0002
CTRL = 037F STAT = 2800 TAGS = 00E0 EIP = 5AD512A4 EDO = 00000000
```

More Loads – Logarithm Constants

- `fild2e`
 - Pushes $\log_2(e)$ onto floating point stack
- `fild2t`
 - Pushes $\log_2(10)$ onto fp stack
- `fildlg2`
 - Pushes $\log_{10}(2)$ onto fp stack
- `fildln2`
 - Pushes $\log_e(2)$ onto fp stack

Note:

No
operand
is
required.

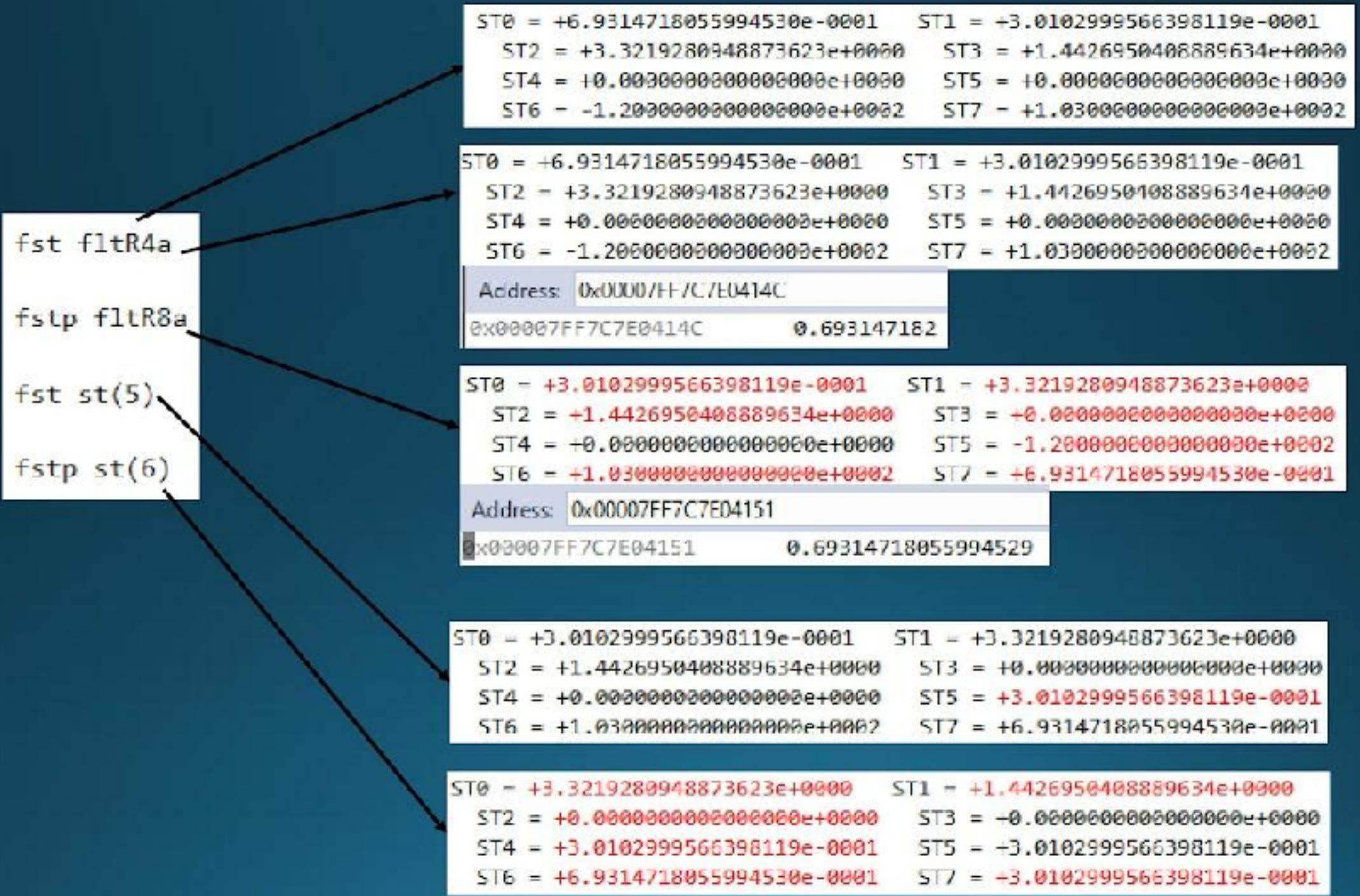
```
ST0 = +6.9314718055994530e-0001    ST1 = +3.0102999566398119e-0001
ST2 = +3.3219280948873623e+0000    ST3 = +1.4426950408889634e+0000
ST4 = +0.0000000000000000e+0000    ST5 = +0.0000000000000000e+0000
ST6 = -1.2000000000000000e+0002    ST7 = +1.0300000000000000e+0002
CTRL = 037F STAT = 2000 TAGS = 00F0 EIP = 5AD512AF EDO = 00000000
```

Store Instructions

Store instructions

- Stores data from stack top ST to memory or stack register
 - Mnemonics contain f, s, t and few other letters
- *fst realMemoryOperand*
 - Copies stack top ST value to memory
- *fstp realMemoryOperand*
 - Copies stack top ST value to memory *and* pops the floating point stack
- *fst st(n)*
 - Copies stack top ST value to register ST(n)
- *fstp st(n)*
 - Copies stack top ST value to register ST(n) *and* pops the floating point stack

Store Instructions - Example



Store Instructions

- `fist integerMemoryOperand`
 - Copies stack top ST value to memory, converting(rounding) to integer
- `fistp integerMemoryOperand`
 - Same as `fist`, but also pops the floating point stack
- `fisttp integerMemoryOperand`
 - Same as `fistp`, but the value is truncated instead of getting rounded.

Store Instructions - Example

The diagram illustrates the effect of different store instructions on memory. A vertical stack of four memory dump windows shows the state of registers ST0-ST7 at three different addresses: 0x00007FF7C7E04025, 0x00007FF7C7E04021, and 0x00007FF7C7E04027. Arrows from the left point to each window, labeling them with their respective instructions: fist wnum, fistp dnum, fisttp qnum, and an unlabeled fourth row.

fist wnum

fistp dnum

fisttp qnum

ST0 = +3.3219280948873623e+0000 ST1 = +1.4426950408889634e+0000
ST2 = +0.0000000000000000e+0000 ST3 = +0.0000000000000000e+0000
ST4 = +3.0102999566398119e-0001 ST5 = +3.0102999566398119e-0001
ST6 = +6.9314718055994530e-0001 ST7 = +3.0102999566398119e-0001

ST0 = +3.3219280945873623e+0000 ST1 = +1.4426950405889634e+0000
ST2 = +0.0000000000000000e+0000 ST3 = +0.0000000000000000e+0000
ST4 = +3.0102999566398119e-0001 ST5 = +3.0102999566398119e-0001
ST6 = +6.9314718055994530e-0001 ST7 = +3.0102999566398119e-0001

Address: 0x00007FF7C7E04025
0x00007FF7C7E04025 -3

ST0 = +1.4426950408889634e+0000 ST1 = +0.0000000000000000e+0000
ST2 = +0.0000000000000000e+0000 ST3 = +3.0102999566398119e-0001
ST4 = +3.0102999566398119e-0001 ST5 = +6.9314718055994530e-0001
ST6 = +3.0102999566398119e-0001 ST7 = +3.3219280948873623e+0000

Address: 0x00007FF7C7E04021
0x00007FF7C7E04021 +3

ST0 = +0.0000000000000000e+0000 ST1 = +0.0000000000000000e+0000
ST2 = +3.0102999566398119e-0001 ST3 = +3.0102999566398119e-0001
ST4 = +6.9314718055994530e-0001 ST5 = +3.0102999566398119e-0001
ST6 = +3.3219280948873623e+0000 ST7 = +1.4426950408889634e+0000

Address: 0x00007FF7C7E04027
0x00007FF7C7E04027 +1

Exchange Instructions

Exchange instructions

- Exchanges data between stack top ST and a stack register
 - Can't be used to exchange values in memory.
- fxch
 - Exchange values in ST and ST(1)
- fxch *st (nbr)*
 - Exchange ST and ST(*nbr*)

Exchange Instructions

The diagram illustrates the effect of three different `fxch` instructions on the floating-point stack (FPU). A central box contains the initial state of the stack:

ST0 = +1.4426950408889634e+0000	ST1 = +0.0000000000000000e+0000
ST2 = +3.1415926535897932e+0000	ST3 = +1.0000000000000000e+0000
ST4 = +0.0000000000000000e+0000	ST5 = +0.0000000000000000e+0000
ST6 = +3.0102999566398119e-0001	ST7 = +3.0102999566398119e-0001

Three arrows point from the text to the boxes:

- An arrow points from the `fxch` label to the top box.
- An arrow points from the `fxch st(3)` label to the middle box.
- An arrow points from the `fxch st(3)` label to the bottom box.

Top Box (fxch result):

ST0 = +0.0000000000000000e+0000	ST1 = +1.4426950408889634e+0000
ST2 = +3.1415926535897932e+0000	ST3 = +1.0000000000000000e+0000
ST4 = +0.0000000000000000e+0000	ST5 = +0.0000000000000000e+0000
ST6 = +3.0102999566398119e-0001	ST7 = +3.0102999566398119e-0001

Middle Box (fxch st(3) result):

ST0 = +1.0000000000000000e+0000	ST1 = +1.4426950408889634e+0000
ST2 = +3.1415926535897932e+0000	ST3 = +0.0000000000000000e+0000
ST4 = +0.0000000000000000e+0000	ST5 = +0.0000000000000000e+0000
ST6 = +3.0102999566398119e-0001	ST7 = +3.0102999566398119e-0001

