CSCI -112
Introduction to computer Systems

Instructor: Santanu Banerjee

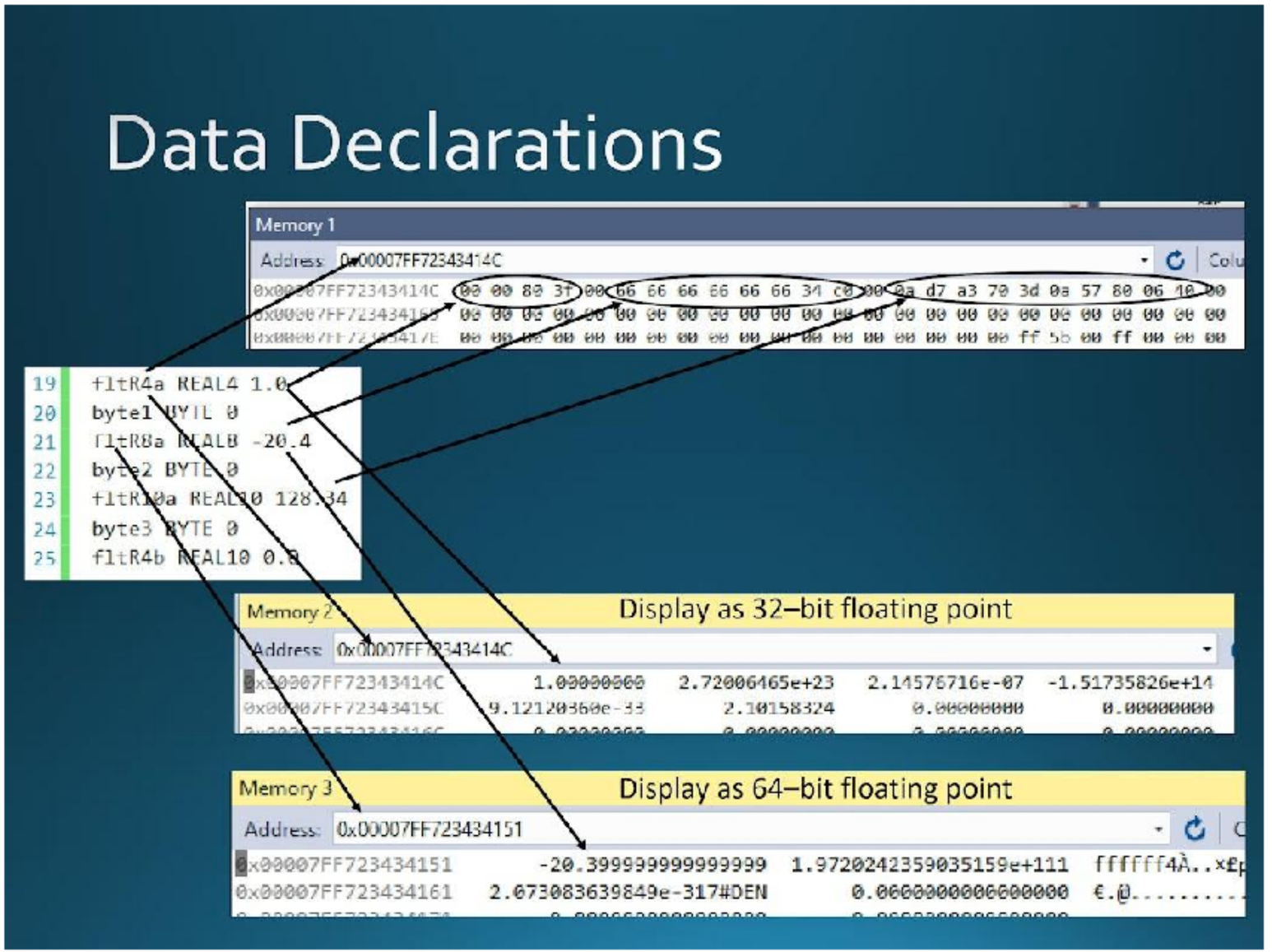
Topics

- 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

Floating Point Formats

Data Declarations



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

fild wnum
fild dwnum
fild qwnum

fld ST
fld ST(1)
fld ST(5)
```

```
ST0 = +0.0000000000000000e+0000   ST1 = +0.0000000000000000e+0000
  ST2 = +0.0000000000000000e+0000   ST3 = +0.0000000000000000e+0000
  ST4 = +0.0000000000000000e+0000   ST5 = +0.0000000000000000e+0000
  ST6 = +0.0000000000000000e+0000   ST7 = +0.0000000000000000e+0000
  CTRL = 027F STAT = 0000 TAGS = 0000 EIP = 00000000 EDO = 00000000
```

```
ST0 = +1.2834000000000000e+0002   ST1 = -2.0399999999999998e+0001
  ST2 = +1.0000000000000000e+0000   ST3 = +0.0000000000000000e+0000
  ST4 = +0.0000000000000000e+0000   ST5 = +0.0000000000000000e+0000
  ST6 = +0.0000000000000000e+0000   ST7 = +0.0000000000000000e+0000
  CTRL = 027F STAT = 2800 TAGS = 00E0 EIP = 720D127C EDO = 720D415A
```

```
ST0 = +0.0000000000000000e+0000   ST1 = +0.0000000000000000e+0000
  ST2 = +0.0000000000000000e+0000   ST3 = +0.0000000000000000e+0000
  ST4 = +0.0000000000000000e+0000   ST5 = +1.2834000000000000e+0002
  ST6 = -2.0399999999999998e+0001   ST7 = +1.0000000000000000e+0000
  CTRL = 037F STAT = 0000 TAGS = 0000 EIP = 00000000 EDO = 00000000
```

```
ST0 = +1.0300000000000000e+0002   ST1 = +4.4000000000000000e+0001
  ST2 = -1.2000000000000000e+0002   ST3 = +0.0000000000000000e+0000
  ST4 = +0.0000000000000000e+0000   ST5 = +0.0000000000000000e+0000
  ST6 = +0.0000000000000000e+0000   ST7 = +0.0000000000000000e+0000
  CTRL = 037F STAT = 2800 TAGS = 00E0 EIP = 720D1291 EDO = 720D4027
```

```
ST0 = -1.2000000000000000e+0002   ST1 = +1.0300000000000000e+0002
  ST2 = +1.0300000000000000e+0002   ST3 = +1.0300000000000000e+0002
  ST4 = +4.4000000000000000e+0001   ST5 = -1.2000000000000000e+0002
  ST6 = +0.0000000000000000e+0000   ST7 = +0.0000000000000000e+0000
  CTRL = 037F STAT = 1000 TAGS = 00FC EIP = D465129B EDO = D4654027
```

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 = +0.0000000000000000e+0000
  ST2 = +1.0000000000000000e+0000   ST3 = +0.0000000000000000e+0000
  ST4 = +0.0000000000000000e+0000   ST5 = -1.2000000000000000e+0002
  ST6 = +1.0300000000000000e+0002   ST7 = +1.0300000000000000e+0002
  CTRL = 037F STAT = 2800 TAGS = 00E0 EIP = 5AD512A4 EDO = 00000000
```

More Loads – Logarithm Constants

- `fldl2e`
  - Pushes $\log_2(e)$ onto floating point stack
- `fldl2t`
  - Pushes $\log_2(10)$ onto fp stack
- `fldlg2`
  - Pushes $\log_{10}(2)$ onto fp stack
- `fldln2`
  - 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
```

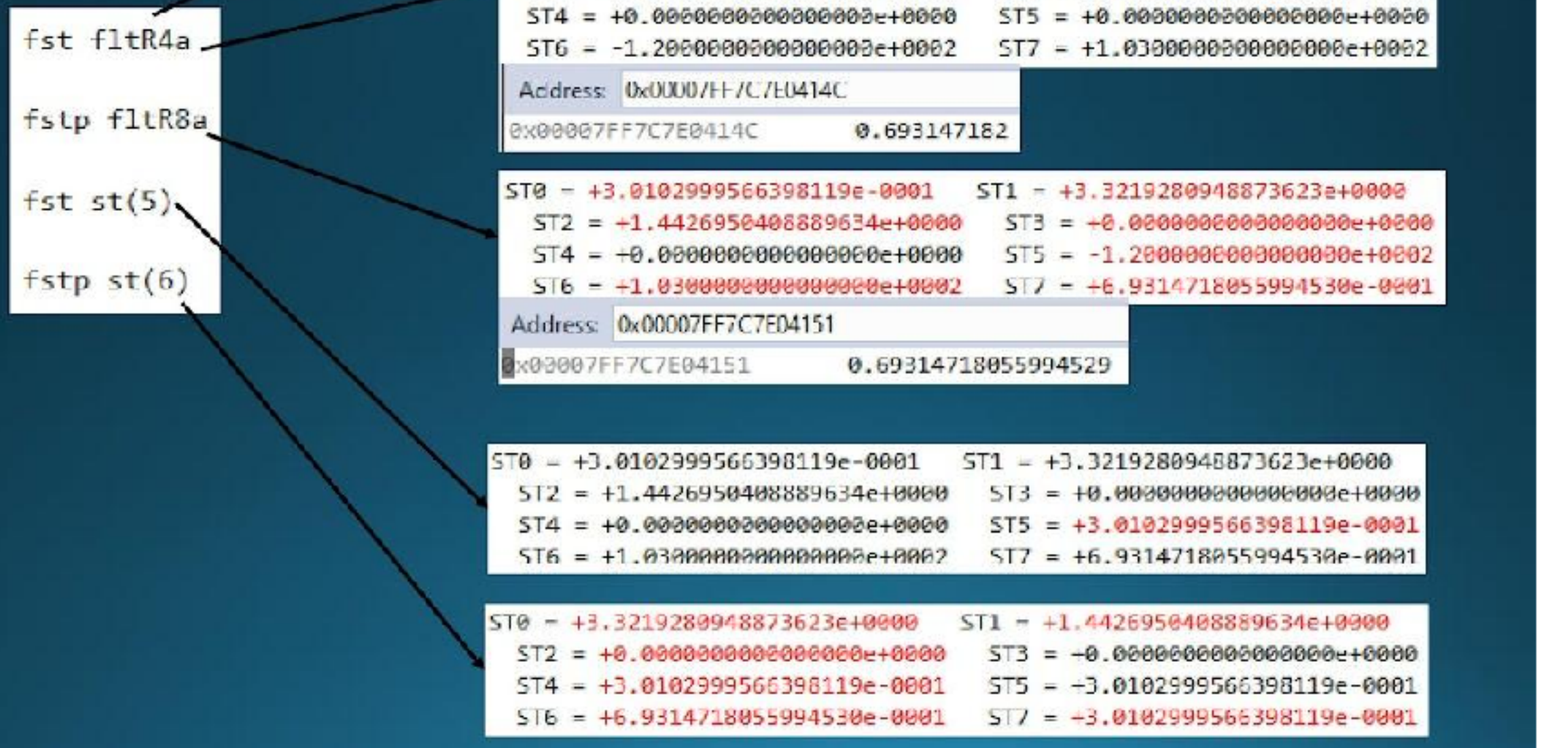
Store Instructions

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

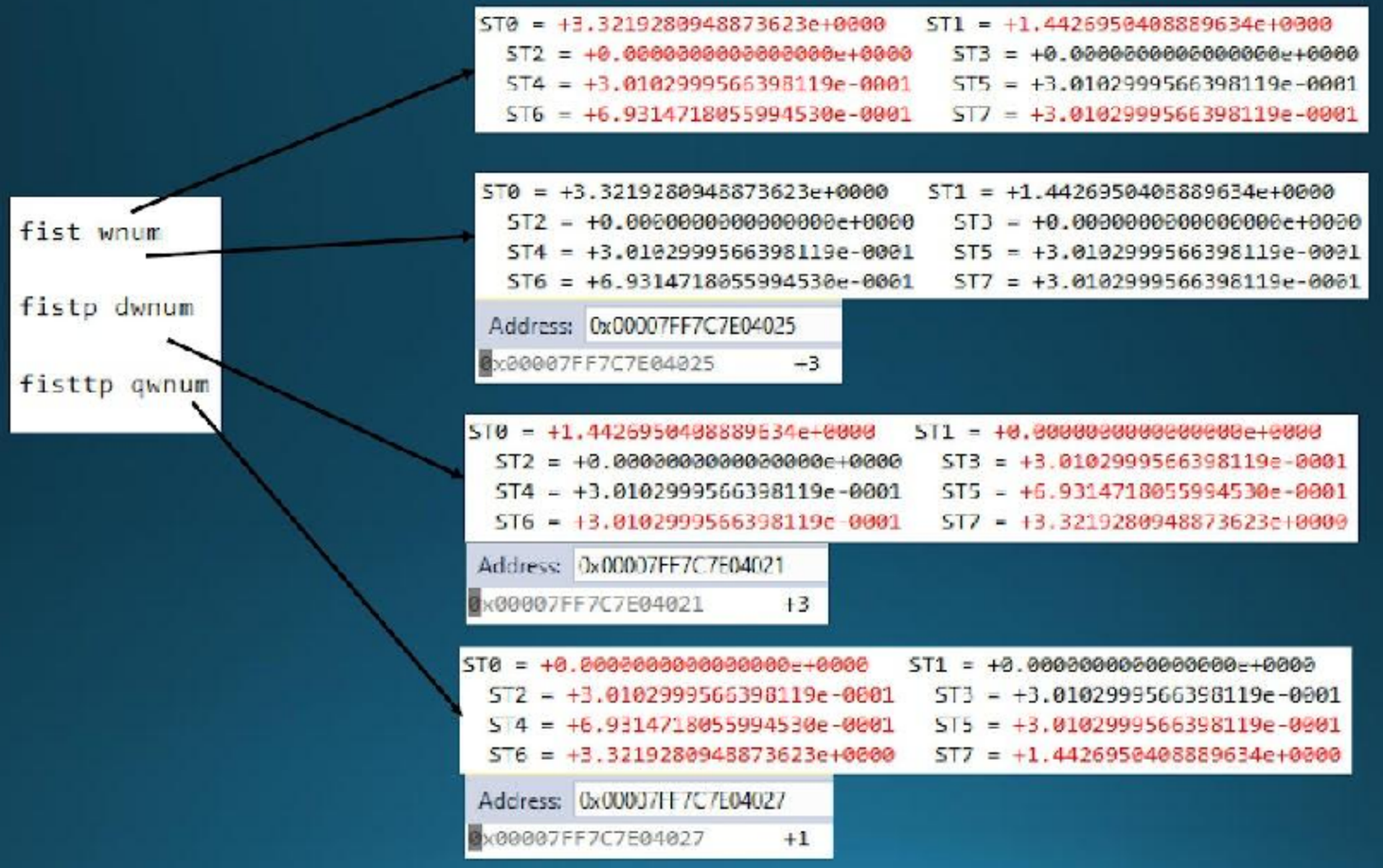
Store Instructions - Example



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



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