

IA32 Floating Point

History

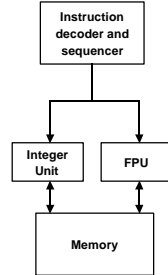
- 8086: first computer to implement IEEE FP
 - separate 8087 FPU (floating point unit)
- 486: merged FPU and Integer Unit onto one chip

Summary

- Hardware to add, multiply, and divide
- Floating point data registers
- Various control & status registers

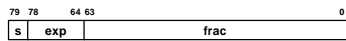
Floating Point Formats

- single precision (C float): 32 bits
- double precision (C double): 64 bits
- extended precision (C long double): 80 bits



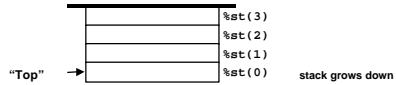
FPU Data Register Stack

FPU register format (extended precision)



FPU registers

- 8 registers
- Logically forms shallow stack
- Top called %st(0)
- When push too many, bottom values disappear



FPU instructions

Large number of floating point instructions and formats

- ~50 basic instruction types
- load, store, add, multiply
- sin, cos, tan, arctan, and log!

Sample instructions:

Instruction	Effect	Description
<code>fldz</code>	push 0.0	Load zero
<code>flds Addr</code>	push M[Addr]	Load single precision real
<code>fmulb Addr</code>	$\%st(0) \leftarrow \%st(0) * M[Addr]$	Multiply
<code>faddp</code>	$\%st(1) \leftarrow \%st(0) + \%st(1)$; pop	Add and pop

Sample Code

```
float compute(float x, float y) {
    return(x+y);
}
```

```
compute:
    pushl   %ebp
    movl   %esp, %ebp
    flds   8(%ebp)
    fadds  12(%ebp)
    popl   %ebp
    ret
```

CompOrg Fall 2002 - IA32 Floating Point

4

Better Sample Code

```
float compute2(float x, float y) {
    return(x*2+y*3.1415);
}
```

```
.LC1: .long 0xc083126f,0x400921ca
compute2:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $4, %esp
    flds   8(%ebp)          # put x on fpstack
    fadd  %st(0), %st      # add to itself
    flds   12(%ebp)        # put y on fpstack
    fldl   .LC1            # put 3.1415 fpstack
    fmulp  %st, %st(1)     # multiply/pop
    faddp  %st, %st(1)     # add/pop
    fstps  -4(%ebp)        # cvt to float
    flds  -4(%ebp)         # put float on stack
    leave
    ret
```

CompOrg Fall 2002 - IA32 Floating Point

5

Floating Point Code Example

Compute Inner Product of Two Vectors

- Single precision arithmetic
- Scientific computing and signal processing workhorse

```
float ipF(float x[],
         float y[],
         int n)
{
    int i;
    float result = 0.0;
    for (i = 0; i < n; i++) {
        result += x[i] * y[i];
    }
    return result;
}
```

```
    pushl %ebp          # setup
    movl %esp,%ebp
    pushl %ebx

    movl 8(%ebp),%ebx   # %ebx=x
    movl 12(%ebp),%ecx  # %ecx=y
    movl 16(%ebp),%edx  # %edx=n
    flds  # push +0.0
    xorl %eax,%eax     # i=0
    cmpl %edx,%eax     # if i>n done
    jge .L3

.L5:
    flds (%ebx,%eax,4) # push x[i]
    fmuls (%ecx,%eax,4) # st(0)=x[i]*y[i]
    faddp # st(1)=st(0); pop
    incl %eax          # i++
    cmpl %edx,%eax     # if i<n repeat
    jl .L5

.L3:
    movl -4(%ebp),%ebx # finish
    movl %ebp, %esp
    popl %ebp
    ret                # st(0) = result
```

CompOrg Fall 2002 - IA32 Floating Point

6

Inner Product Stack Trace

Initialization

1. `fldz`

0.0	%st(0)
-----	--------

Iteration 0

2. `flds (%ebx,%eax,4)`

0.0	%st(1)
x[0]	%st(0)

3. `fmuls (%ecx,%eax,4)`

0.0	%st(1)
x[0]*y[0]	%st(0)

4. `faddp`

0.0+x[0]*y[0]	%st(0)
---------------	--------

Iteration 1

5. `flds (%ebx,%eax,4)`

x[0]*y[0]	%st(1)
x[1]	%st(0)

6. `fmuls (%ecx,%eax,4)`

x[0]*y[0]	%st(1)
x[1]*y[1]	%st(0)

7. `faddp`

x[0]*y[0]+x[1]*y[1]	%st(0)
---------------------	--------

CompOrg Fall 2002 - IA32 Floating Point 7
