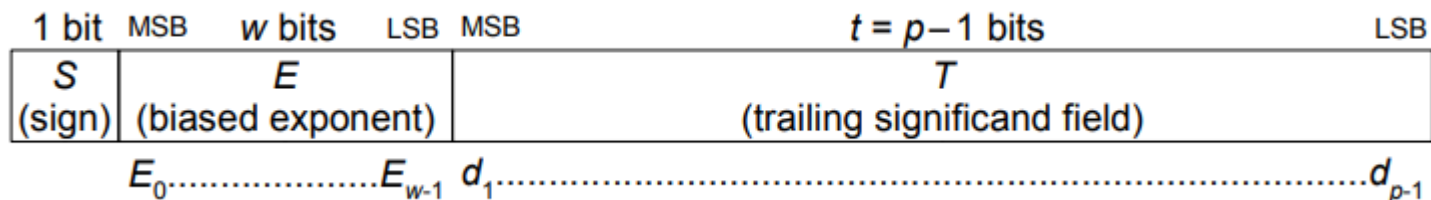


Lecture 6. Floating point arithmetic and data conversion

Dmitry Zaitsev

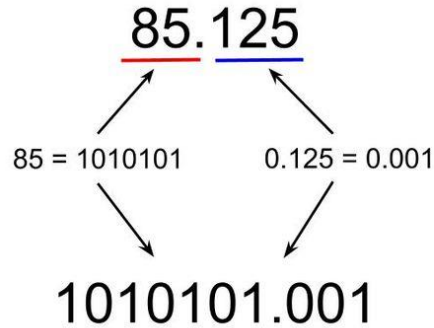
<http://daze.ho.ua>

Standard for Floating-Point Arithmetic (IEEE 754)




Parameter	binary16	binary32	binary64	binary128	binary{k} ($k \geq 128$)
k , storage width in bits	16	32	64	128	multiple of 32
p , precision in bits	11	24	53	113	$k - \text{round}(4 \times \log_2(k)) + 13$
$emax$, maximum exponent e	15	127	1023	16383	$2^{(k-p-1)} - 1$
<i>Encoding parameters</i>					
$bias$, $E - e$	15	127	1023	16383	$emax$
sign bit	1	1	1	1	1
w , exponent field width in bits	5	8	11	15	$\text{round}(4 \times \log_2(k)) - 13$
t , trailing significand field width in bits	10	23	52	112	$k - w - 1$
k , storage width in bits	16	32	64	128	$1 + w + t$

Example: get binary code




Whole Number Division	Result	Remainder
85/2	42	1
42/2	21	0
21/2	10	1
10/2	5	0
5/2	2	1
2/2	1	0
1/2	0	1



85 = 1010101

Decimal Number Multiplication	Result	Number in front of decimal
0.125 x 2	0.25	0
0.25 x 2	0.5	0
0.5 x 2	1.0	1
0.0 x 2	0.0	0



0.125 = 001

Example: Fitting format

1010101.001

Move decimal 6 places

1.010101001 x 2⁶

Whole Number Division	Result	Remainder
133/2	66	1
66/2	33	0
33/2	16	1
16/2	8	0
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1

133 = 10000101

Exponent

Sign

+ → 0

- → 1

1.010101001 x 2⁶

127 + 6 = 133

1.010101001 x 2⁶

Mantissa

001000110010101001

010000101

Get hex code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char * argv[])
{
    if(argc<2)
    {
        printf("USAGE: numr type value\n");
    }
    else if(strcmp(argv[1],"int")==0)
    {
        int i = atoi(argv[2]);
        unsigned int * pi = (unsigned int *)&i;
        printf("int %d internal 0x%08x\n",i,*pi);
    }
}
```

```
else if(strcmp(argv[1],"char")==0)
{
    char c = argv[2][0];
    printf("char %c internal 0x%02x\n",c,(int)c);
}
else if(strcmp(argv[1],"float")==0)
{
    float f = (float)atof(argv[2]);
    unsigned int * pf = (unsigned int *)&f;
    printf("float %g internal 0x%08x\n",f,*pf);
}
else if(strcmp(argv[1],"double")==0)
{
    double d = atof(argv[2]);
    unsigned long int * pd = (unsigned long int *)&d;
    printf("float %lg internal 0x%016lx\n",d,*pd);
}
else printf("TYPES: int, char, float, double\n");
}
```

SSE versus x87 FPU

- X87 Floating-Point Unit (FPU) – stack of registers, 1981
- Streaming SIMD Extensions (SSE), 70 instruction, 8 registers (%xmm0-%xmm7)
- Single Instruction, Multiple Data (SIMD) instruction set extension to the x86 architecture, 1999
- Advanced Vector Extensions (AVX), 2011

Floating point unit overview

- Check presence of FPU

mov eax, 1 ; *argument request feature report*
cpuid

xor rax, rax ; *wipe clean accumulator register*

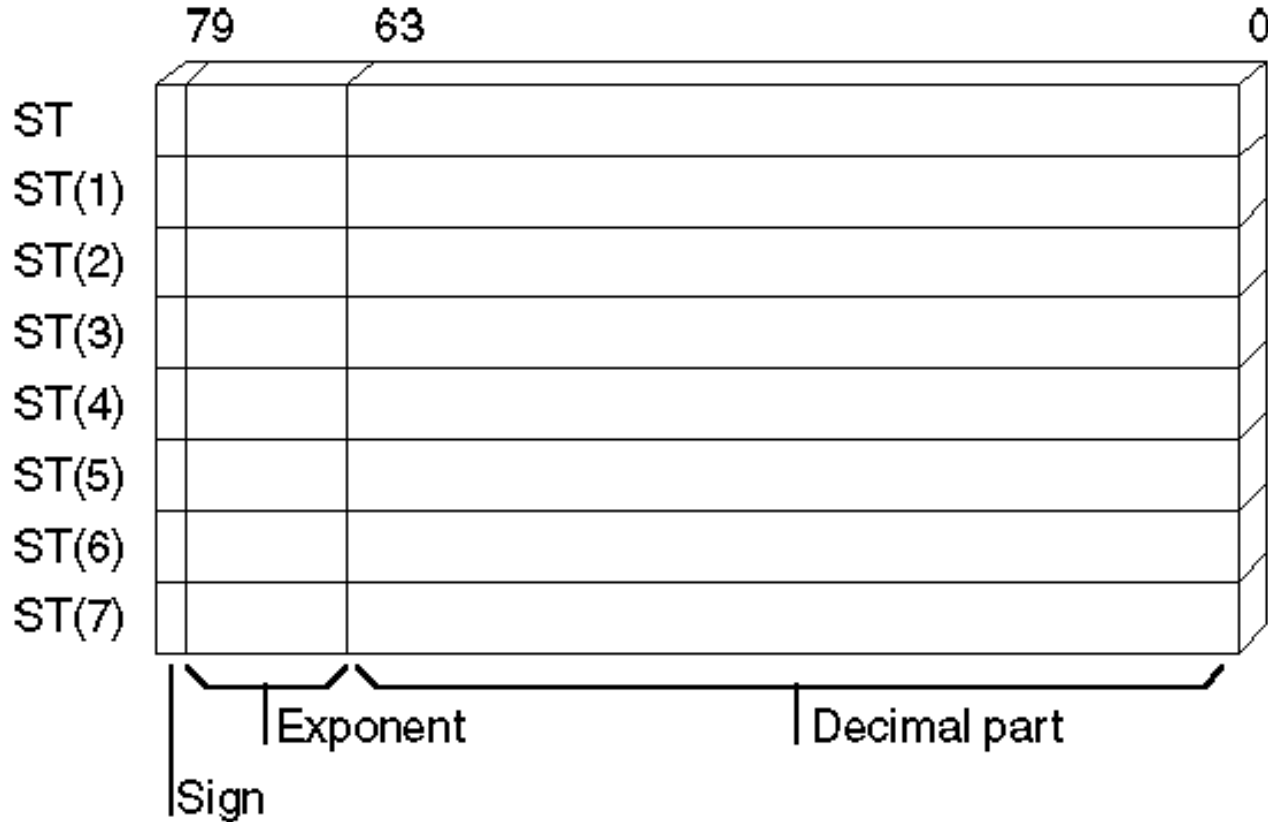
bt edx, rax ; *CF := edx[rax] retrieve bit 0*

setc al ; *al := CF*

Floating point stack

- Floating point registers: $st(0) - st(7)$
- Register width is 80 bits
- Organized as a stack
- $st(0)$ – top
- $st(7)$ – bottom

Floating-point stack scheme



Special data transfer instructions

- Load floating point value
fld source
- Store floating point value
fst destination
- Store floating point value and pop
fstp destination

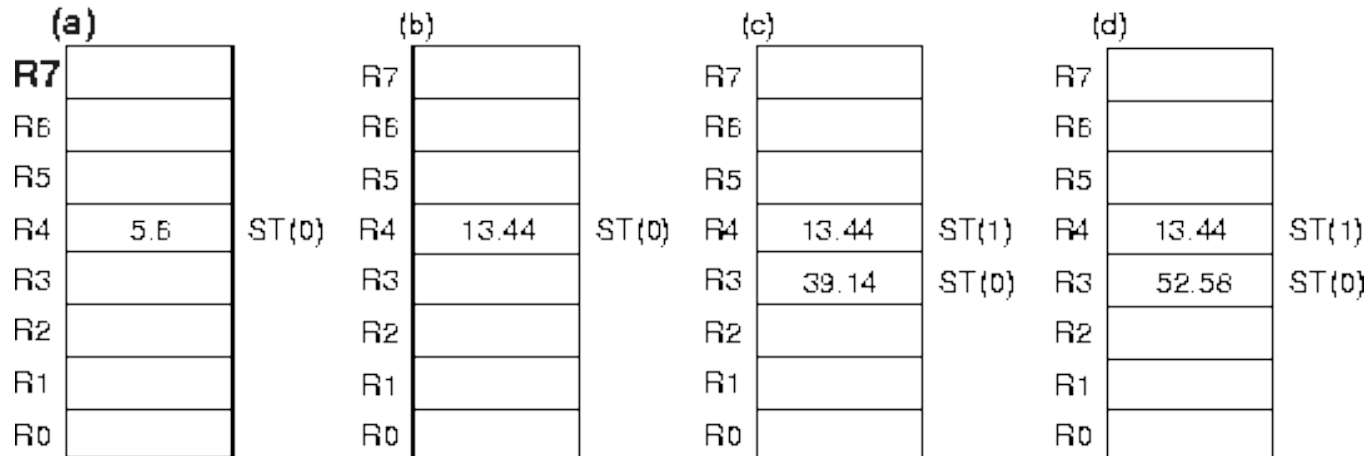
An example of computations

Computation

Dot Product = $(5.6 \times 2.4) + (3.8 \times 10.3)$

Code:

```
FLD  value1 ; (a) value1=5.6  
FMUL value2 ; (b) value2=2.4  
FLD  value3 ; value3=3.8  
FMUL value4 ; (c) value4=10.3  
FADD ST(1)  ; (d)
```



Basic Arithmetic Instructions

Mnemonic	Description
fabs	absolute value
fadd	add
fchs	change sign
fdiv	divide
fmul	multiply
frndint	round to ineger
fsqrt	square root
fsub	subtract

Mnemonic	Description
fcom	compare
fcomi	compare and set %eflags
fexam	examine
fcos	sine
fsin	cosine
fptan	partial tangent
f2xm1	computes $2^x - 1$
fyl2x	computes $y * \log_2 x$

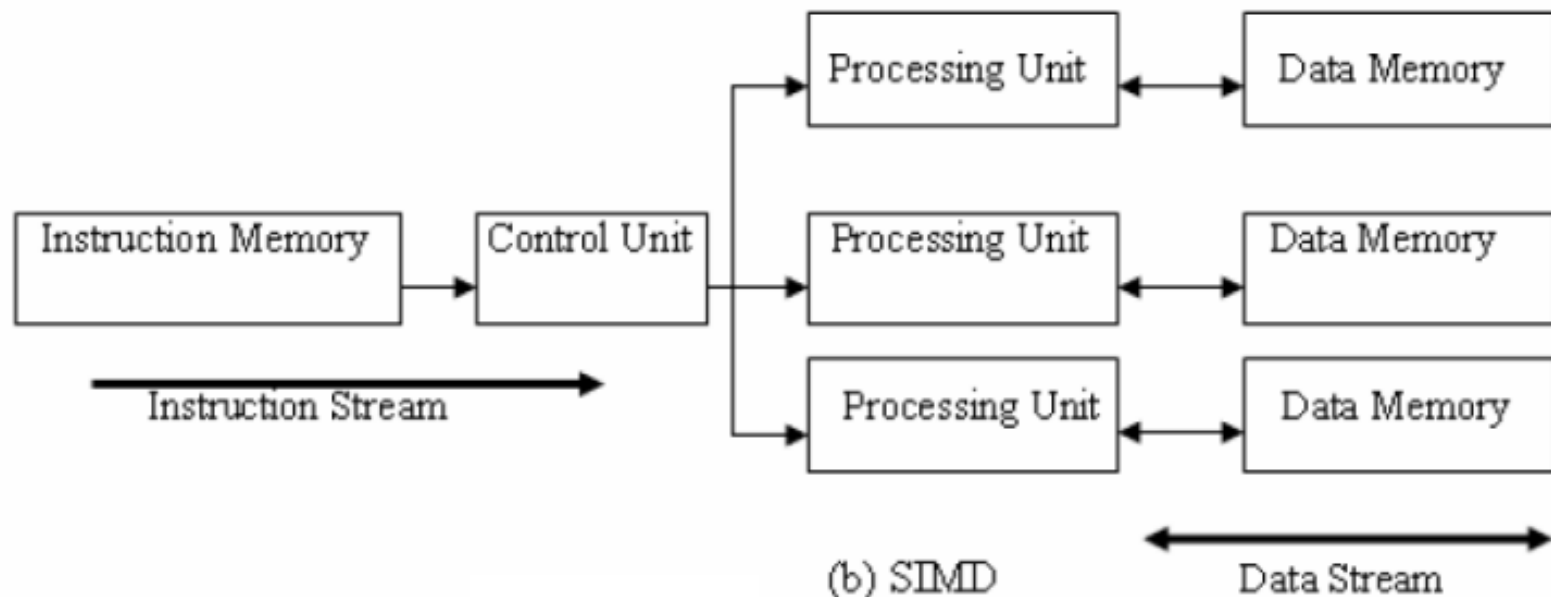
suffix “p” – “and pop”

Other Instructions

Mnemonic	Description
fld1	load +1.0
fldl2e	load $\log_2 e$
fldl2t	load $\log_2 10$
fldln2	load $\log_e 2$
fldpi	load π
fldz	load +0.0
fldlg2	load $\log_{10} 2$

Mnemonic	Description
fwait	wait for floating-point unit
fdecstp	decrement floating-point register stack pointer
ffree	free floating-point register
fincstp	increment floating-point register stack pointer
fldcw	load floating-point unit control word
fldenv	load floating-point unit environment

SIMD –data parallel architecture



SSE floating-point arithmetic

- Registers (%xmm0-%xmm7)
- Dedicated moving data:
 - `movapd %xmm7, %xmm2`
 - `movsd e(%rip), %xmm8`
- Arithmetic instructions:
 - `addsd %xmm7, %xmm6`
 - `mulsd %xmm1, %xmm4`

SSE instructions

- Data movement instructions
- Arithmetic instructions
- Logical instructions
- Comparison instructions
- Shuffle instructions
- Miscellaneous

Data conversion, Cache control, State management

Example - e^x

- Power series representation – as basics for computer implementation of math functions

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1} + \frac{x^2}{1 \cdot 2} + \frac{x^3}{1 \cdot 2 \cdot 3} + \dots$$

- for $|x| < 1$
- Computing hint: a member is greater than a “tail”

C language prototype

```
#define EPS 0.000001
double myexp(double x)
{
    double s=1.0,si=1.0,xi=1.0,fi=1.0,i=0.0;
    while( fabs( si ) >= EPS )
    {
        i=i+1.0;
        xi=xi*x;
        fi=fi*i;
        si=xi/fi;
        s=s+si;
    }
    return s;
}
```

- efficient computing:

$$x^i = x \cdot x^{n-1}$$

$$x! = x \cdot (x - 1)!$$

- notation of variables:
 - ✓ s – partial sum
 - ✓ si – current member
 - ✓ xi – numerator
 - ✓ fi – denominator
 - ✓ EPS – precision

Function myexp

(for $x > 0$)

.global myexp

myexp:

pushq %rbp

movq %rsp, %rbp

movsd %xmm0, %xmm1 # x

movsd e(%rip), %xmm8 # e

pxor %xmm6, %xmm6 # i

movsd c1(%rip), %xmm7 # 1.0

movapd %xmm7, %xmm2 # s

movapd %xmm7, %xmm3 # si

movapd %xmm7, %xmm4 # xi

movapd %xmm7, %xmm5 # fi

loop: comisd %xmm3, %xmm8

jnb quit

addsd %xmm7, %xmm6 # $i = i + 1.0$

mulsd %xmm1, %xmm4 # $xi = xi * x$

mulsd %xmm6, %xmm5 # $fi = fi * i$

movsd %xmm4, %xmm0 # %xmm0 = xi

divsd %xmm5, %xmm0 # %xmm0 = xi / fi

movsd %xmm0, %xmm3 # si = xi / fi

addsd %xmm3, %xmm2 # $s = s + si$

jmp loop

quit: movsd %xmm2, %xmm0

popq %rbp

ret

.section .rodata

e: .double 0.0000001

c1: .double 1.0

Main program in C

```
#include <stdio.h>
#include <math.h>
extern double myexp(double x);
int main()
{
    double x;
    scanf("%lf",&x);
    printf("%lf %lf\n",myexp(x),exp(x));
}
```

- compile and link

gcc -o ex ex.c myexp.s -lm

- run

./ex

0.1

1.10517 1.10517

Task

- Write assembler programs for computing basic math functions: sin, cos, tg, ...
- Compare result and performance with the corresponding libc function
- For benchmarks use big array of source data
- For measuring execution time use function `magma_wtime` from file `Adriana.c` at <https://github.com/dazeorgacm/ParAd>