7/26 Midterm

makefile
Makefile
s.makefile
s.Makefile

Make -f compArch.mak

<op>
+
-
*

ex. makefile

```
EX01:    EX01.c
   $(compile) -c EX01 EX01.c
Error.o:   Error.c Error.h
   $(COMPILE) -c Error.c
```

all: bunch of files

Tar files: // do this on iLab machine

create

tar cvf filename.tar | List of directories or filenames

or

tf just to see files

see contents

tar tvf filename.tar

Extraction

Tar xvf filename.tar

① makefile on Ilab machine
② make clean all .o files (only .c, .h, .pdf files etc)
③ tar xvf to extract

* can use macros to put different files in different subdirectories

macros →
Machine = Intel
OS = Linux
UI = cmd line

Disassem.o: $(Machine)/Disassem.c \ ← continue on next line
$(Machine)/Disassem.h

transferring file to remote pc

SCP CS211.TAR morbius@base.cs.rutgers.edu:~Home

filename login remote pc name dir

★ 1 in IEEE is never there b/c it's always there.

Decimal Scientific Notation:

$1 = 1 \times 10^0$	1×10^0
$3000 = 3000 \times 10^0$	3×10^3
$412 = 412 \cdot 10^0$	$4.12 \cdot 10^2$
$1/3 = 0.\overline{3} \cdot 10^0$	$3.33\overline{3} \cdot 10^{-1}$
unnormalized 0.2	Normalized $2 \cdot 10^{-1}$

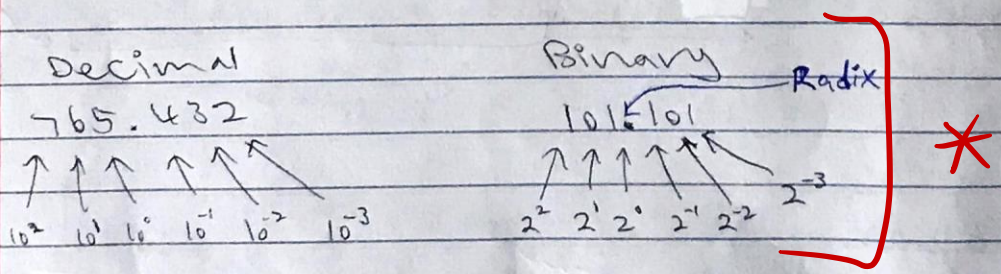
- Normalized
- Mantissa (or magnitude)
- Between 0 and (not quite) 10 (ex) 0 to 9.999)

★ 1. magnitude
2. Exponent
3. Sign for magnitude

Binary Floating Point:

A binary (base 2) magnitude to some positive or negative power of 2.

$1 = 1 \times 2^0$		$1 = 1 \times 2^0$
$2 = 1 \times 2^1$	Normalized $\Rightarrow$	$2 = 1 \times 2^1$
$4 = 1 \times 2^2$		$4 = 1 \times 2^2$
$3 = 11 \times 2^0$		$3 = 1.1 \times 2^1$
$5 = 101 \times 2^0$		$5 = 1.01 \times 2^2$



★

$1/2 = 1 \times 2^{-1}$

$1/4 = 1 \times 2^{-2}$

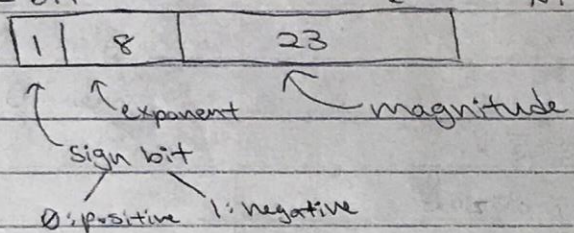
$1/3 = 1.0101010\overline{1} \times 2^{-2}$

$1/5 = 1.10011001100\overline{1100} \times 2^{-3}$

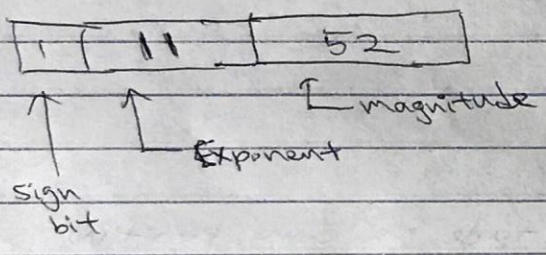
- magnitude
- Sign: 0 or 1
- Exponent: power of 2

IEEE 754:

- 32 bit

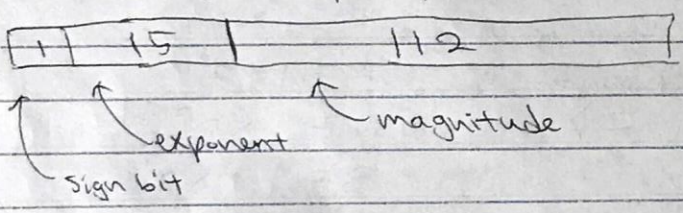


- 64 bit



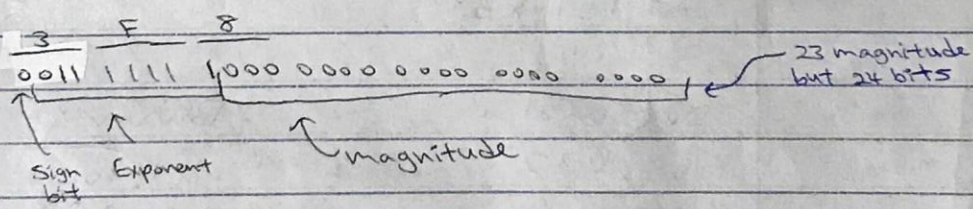
~15 decimal digits precision

- 128 bit quadruple precision



~34 decimal digits precision

ex) HEX 3F800000



Exponent bias:

- range from ⊕ to ⊖ value by adding the bias value to the exponent.
- for K bit exponent, bias value is $2^{k-1} - 1$.

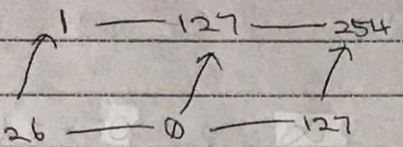
This means that exponent value N is stored as IEEE exponent value N + Bias.

- 32 bit bias is 127 8-bit exponent
- 64 bit bias is 1023 11-bit exponent
- 128 bit bias is 16383 15-bit exponent

Float

32-bit IEEE754

Exponent + Bias



Exponent

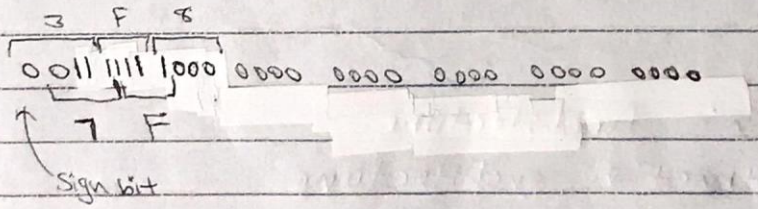
Exponent	Exponent + Bias		Representing
127	254	0XFE	2^{127}
⋮	⋮		⋮
2	129	0x81	2^2
1	128	0x80	2^1
0	127	0X7F	$2^0 = 1$
-1	126	0X7E	$2^{-1} = \frac{1}{2}$
-2	125	0x7D	$2^{-2} = \frac{1}{4}$
⋮	⋮		⋮
-126	1	0X01	2^{-126}

*Advantage: can compare floating numbers just like integers

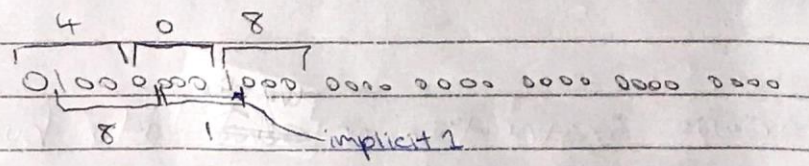
What about 0x00 exponent? ..... means 0

Exponent 0xFF? ..... ∞ (+inf.)

ex) 1 = 3F800000



ex) 4 = 40800000



HW

NaN = exponent value is 0XFF and magnitude is non-zero

ex) 7F800000 = ∞ , 7F810100 = +NaN

(0XFF) (0XFF)

FF800000 = -∞ FF8xxxxx = -NaN

- Right shift of unsigned short, int, or long is logical.
- Right shift of short, int, or long is arithmetic.

know byte size

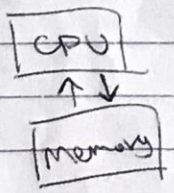
```
Union FI{
    int intPart;
    float floatPart;
} x;
x.intPart = 0;
```

actually same as binary shifting

```
char* ptr = argv[1];
while (*ptr != '\0'){
    number.i = number.i << 1;   //left shift
    number.i += *ptr - '0';
    ptr++;
}
```

07/10 Von Neumann: Architecture

- Stored instruction architecture



It can retrieve instruction or transfer data but cannot do them simultaneously.