

Convert the **base 10 real number** **107.8125** into

A. Base **2** _____

B. Base **8** _____

C. Base **16** _____

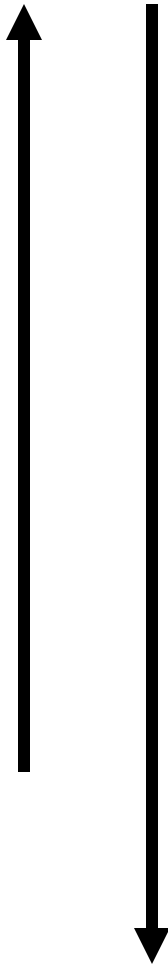
107

8125

15
POINTS
1.

2	107	
2	53	1
2	26	1
2	13	0
2	6	1
2	3	0
2	1	1
2	0	1

	.8125
	x2
1	.6250
	x2
1	.250
	x2
0	.500
	x2
1	.0

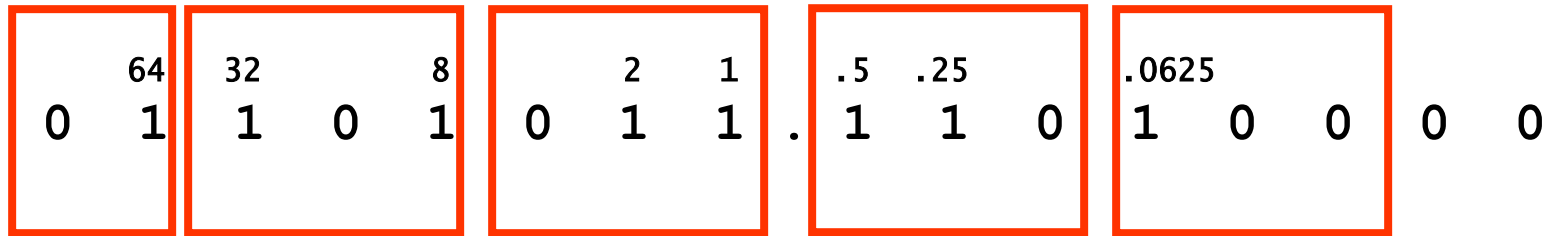


0 1 1 0 1 0 1 1 . 1 1 0 1

15 POINTS

1. Convert the **base 10 real number** 107.8125 into

A. Base 2



1 5 3 . 6 4

B. Base 8

C. Base 16

15 POINTS

1. Convert the **base 10 real number** 107.8125 into

	64	32		8		2	1		.5	.25		.0625				
0	1	1	0	1	0	1	1	.	1	1	0	1	0	0	0	0

A. Base 2 _____

B. Base 8 _____

6 B . D 0

C. Base 16 _____

As you can see below, the following code beginning at the label **main:** pushes two arguments in the form of simple 2s complement integers on the stack and then calls a label named **max:**. **You must write the code** at the label named **max:** as a function, using our conventions of expecting arguments on the stack and returning a result in the **AC**. Of course the **max:** function you must write **must return the larger of the two arguments** passed to it on the stack, or the common value if the arguments happen to be the same value. You can see that the code at **main:** sets up the stack for the call to **max:** , makes the call, and then stores the value that is in the AC after the call into the memory location labeled **opres:**

```

op1:  <any 16 bit 2s complement value>
op2:  <any 16 bit 2s complement value>
opres: 0
      .LOC 50
main: lodd op1:
      push
      lodd op2:
      push
      call max:
      insp 2
      stod opres:
      halt

```

```

MAX:   LODL 1 ;OP2
      SUBL 2 ;OP2-OP1
      JNEG OP1BIG:
      LODL 1
      RETN   ;OP2
OP1BIG: LODL 2
      RETN   ;OP1

```



← write the max function

For the following 16 bit sequence:

1 111 111 110 010 101

- A.** What is the **base 10** value if the sequence is a **signed 2's complement integer** ??

$$2 + 8 + 32 + 64 + 1 \rightarrow -107$$

- B.** **Add** the following 2's complement 16 bit sequence to the sequence shown in part **A.** above, and express the answer as a **base 10 signed value**:

0 000 000 001 001 101

$$\begin{array}{r} 1 + 4 + 8 + 64 \rightarrow +77 \\ -107 + 77 \rightarrow -30 \\ 1\ 111\ 111\ 110\ 010\ 101 \\ 0\ 000\ 000\ 001\ 001\ 101 \\ \hline 1\ 111\ 111\ 111\ 100\ 010 \\ 1 + 4 + 8 + 16 + 1 \rightarrow -30 \end{array}$$

Given the following 32 bit sequence:

0 1 0 0 0 0 1 1 0 1 0 1 1 1 0
 sign exponent and mantissa components

- A. If the sequence represents a signed magnitude floating point value using the **IBM format** discussed in class, what is the **base 10 floating point value** of the sequence ??

$$\begin{aligned}
 &0 \text{ 1000011 0101110} \text{---}0 \\
 &+ 67: +3 \quad (2^{-2}+2^{-4}+2^{-5}+2^{-6}) \\
 &\quad 16^{+3} \\
 &\quad 2^{+12} \rightarrow 2^{10}+2^8+2^7+2^6 \rightarrow 1472
 \end{aligned}$$

- B. If the sequence represents a signed magnitude floating point value using the **IEEE 754 single precision** format discussed in class, what is the **base 10 floating point value** of the sequence ??

$$\begin{aligned}
 &0 \text{ 10000110 101110} \text{---}0 \\
 &+ 134: +7 \text{ so } 2^7 * (1+2^{-1}+2^{-3}+2^{-4}+2^{-5}) \\
 &\quad \rightarrow 2^7+2^6+2^4+2^3+2^2 \rightarrow 220
 \end{aligned}$$

The following bit string represents an **IEEE 754 floating point value** called Float 1. You must add to Float 1 the base 10 number shown as Float 2 (you'll have to convert it to a bit pattern first):

Float 1: **0 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

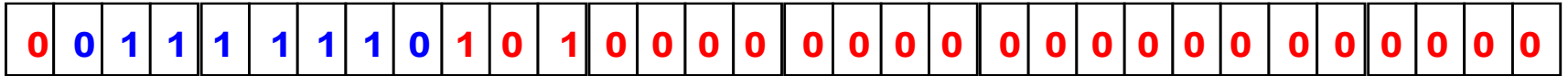
Float 2: **.8125₁₀**

A. Show the **IEEE 754 floating point bit representation** of the sum of these two numbers

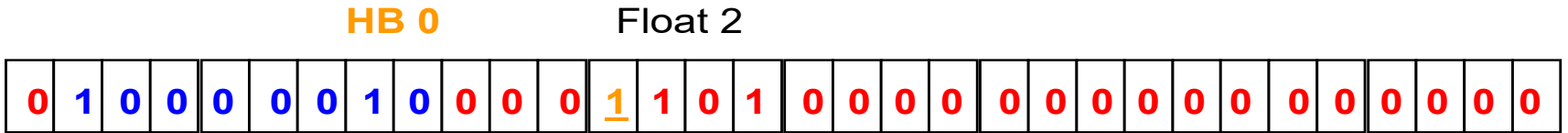
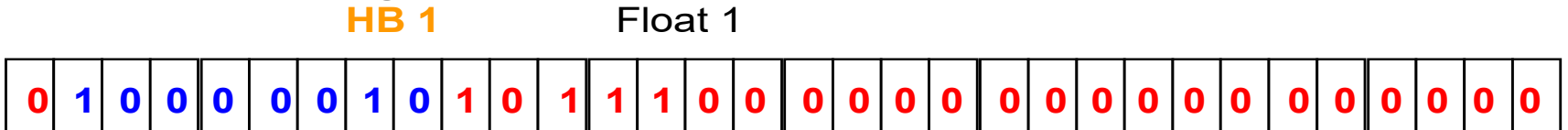
Float 1: 0 1 0 0 0 0 1 0 1 0 1 1 1 0 → 13.75
 Float 2: .8125₁₀

.8125 → . 1 . 1 0 1 go right 1 place so 2⁻¹

A. Show the IEEE 754 floating point bit representation of Float 2 before shifting:

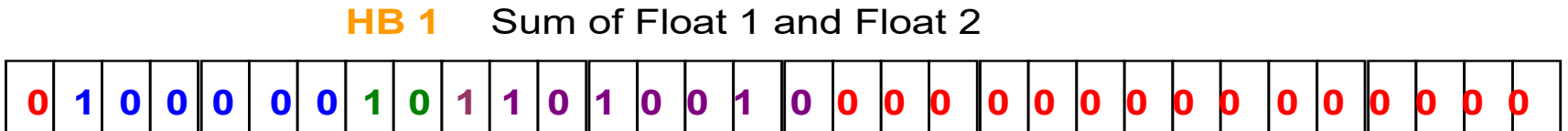


B. Show the IEEE 754 floating point bit representation of Float 1 and Float 2 after shifting:



HB 1 1 1 0 1 0 0 1

C. Show the IEEE 754 floating point bit representation of the final normalized sum of Float 1 and Float 2



D. What is the **base 10 value of the sum** of these two numbers ?

$$\begin{aligned}
 &13.75 + .8125 = 14.5625 \\
 &2^3 * (2^0 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-7}) \\
 &= 8 + 4 + 2 + .5 + .0625
 \end{aligned}$$

List the values in r0:, r1:, r2:, r3:, and r4: after the following program executes from main: to the halt instruction:

```
0 c1: 1
1 c3: 3 → 6, 12, 24, 48, 96
2 index: 5
3 r0: 0 → 6
4 r1: 0 → 12
5 r2: 0 → 24
6 r3: 0 → 48
7 r4: 0 → 96
8 main: lodd index:
9       jzer done:
10      subd c1:
        stod index:
        lodd c3:
        addd c3:
        stod c3:
smc1:  stod r0: store at: 3, 4, 5, 6, 7
        lodd smc1:
        addd c1:
        stod smc1:
        jump main:
done:  halt
```

Self modifying code

Write in the final values of r0:, r1:, r2:, r3:, and r4: below:

Write a routine named **MySub**: which will **subtract two positive only 16 bit 2s complement numbers passed as value arguments**, and will place the **result** back into memory at the location **pointed to by a third argument (which is an address, not a value)**. The routine itself should return a **value of 0 if the result is positive, and -1 if the result is negative**. **Just show the subroutine, not the calling code**. Assume that when the call to your routine is made the arguments are passed on the stack such that:

- SP** points to the location that holds the return PC
- SP+1** points to the location that holds the result address
- SP+2** points to the location that holds the minuend (top number)
- SP+3** points to the location that holds the subtrahend (bottom number)

MySub:

← write
required
code

Write a routine named **MySub**: which will **subtract two positive only 16 bit 2s complement numbers passed as value arguments**, and will place the **result** back into memory at the location **pointed to by a third argument (which is an address, not a value)**. The routine itself should return a **value of 0 if the result is positive, and -1 if the result is negative**. **Just show the subroutine, not the calling code**. Assume that when the call to your routine is made the arguments are passed on the stack such that:

SP points to the location that holds the return PC
SP+1 points to the location that holds the result address
SP+2 points to the location that holds the minuend (top number)
SP+3 points to the location that holds the subtrahend (bottom number)

```
MySub:
    lodl    2
    subl   3
    push
    jneg   neg:
    lodl   2
    popi
    loco   0
    retn
neg:
    lodl   2
    popi
    lodd   cn1:
    retn
cn1:
    -1
```

The following bit string represents an **IEEE 754 single precision floating point number**:

FLOAT: 1 **0 1 1 1 1 1 1 1** 0 1 0

A. Show the **bit string** after the number it represents has been **divided by the base 10 number 128**

1 **0 1 1 1 1 0 0 0** 0 1 0
Original = $-2^0 * 1.25$ New = $-2^{-7} * 1.25$

B. The floating point number shown **above** can be written in hex as:
0x BFA00000

If this value was **stored in a computer system's memory byte by byte** as shown below beginning at memory address 300, explain what type of **endian storage** this system has.

BF	Address 300
A0	Address 301
00	Address 302
00	Etc.

Big Endian, most significant byte first

The MIC-1 bit format is shown below. You should be familiar with all the fields and how they are used. Also below are 5 MAL instructions. Indicate if a given MAL is valid or invalid for MIC-1, and, if valid, fill in the **DECIMAL** (i.e. bits **1101** are filled as **13**) values for each field **in the space provided**.

Register designations are as follows:

pc=0 (prog counter) ac=1 (accumulator) sp=2 (stack ptr) ir=3 (instr reg)
 tir=4 (tmp inst reg) zr=5 (fixed zero) po=6 (plus 1) no=7 (minus 1)
 amask=8 (addr msk) smask=9 (stack msk) a=10(a scratch) b=11(b scratch)
 c=12(c scratch) d=13(d scratch) e=14(e scratch) f=15(f scratch)

- A. pc := pc + 255;mar := pc;rd;
- B. ac := inv(mbr);mar := inv(mbr);wr;
- C. tir := lshift(band(tir,mbr));if n then goto 150;
- D. mbr := lshift(band(ir,amask));ir := lshift(band(ir,amask));goto 0;wr;
- E. b := ir - ac;mar := ac;if z then goto 158;

VALID ?	A	C	M	O	A	M	M	E							
	U	N	L	S	B	A	R	W	N	C	C	B	A	ADDR	

MUX	COND	ALU	SH	MBR, MAR, RD, WR, ENC
A latch	0 = no jmp	0 = A + B	0 = no shift	0 = no
MBR	1 = jmp if n=1	1 = A and B	1 = shift rt	1 = yes
	2 = jmp if z=1	2 = A	2 = shift lt	
	3 = always jmp	3 = not A		

Register designations are as follows:

pc=0 (prog counter) ac=1 (accumulator) sp=2 (stack ptr) ir=3 (instr reg)
 tir=4 (tmp inst reg) zr=5 (fixed zero) po=6 (plus 1) no=7 (minus 1)
 amask=8 (addr msk) smask=9 (stack msk) a=10(a scratch) b=11(b scratch)
 c=12(c scratch) d=13(d scratch) e=14(e scratch) f=15(f scratch)

- A. pc := pc + 255;mar := pc;rd;
- B. ac := inv(mbr);mar := inv(mbr);wr;
- C. tir := lshift(band(tir,mbr));if n then goto 150;
- D. mbr := lshift(band(ir,amask));ir := lshift(band(ir,amask));goto 0;wr;
- E. b := ir - ac;mar := ac;if z then goto 158;

MAL	VALID ?	A M U X	C O N D	A L U	S H	M B R	M A R	R D	W R	E N C	C	B	A	ADDR
A	yes	0	0	0	0	0	1	1	0	1	0	0	9	
B	no													
C	yes	1	1	1	2	0	0	0	0	1	4	4		150
D	yes	0	3	1	2	1	0	0	1	1	3	8	3	0
E	no													

AMUX
 0 = A latch
 1 = MBR

COND
 0 = no jmp
 1 = jmp if n=1
 2 = jmp if z=1
 3 = always jmp

ALU
 0 = A + B
 1 = A and B
 2 = A
 3 = not A

SH
 0 = no shift
 1 = shift rt
 2 = shift lt

MBR, MAR, RD, WR, ENC
 0 = no
 1 = yes

For the base 10 real number 101.21875

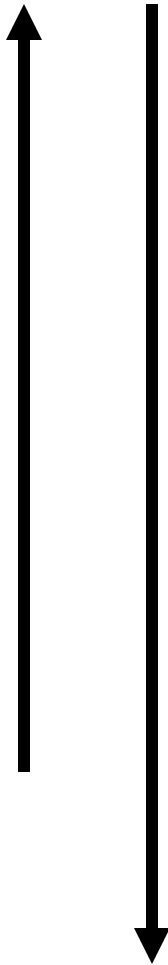
101

.

21875

2		<u>101</u>	
2		<u>50</u>	1
2		<u>25</u>	0
2		<u>12</u>	1
2		<u>6</u>	0
2		<u>3</u>	0
2		<u>1</u>	1
2		<u>0</u>	1

		.21875
		x2
0		.4375
		x2
0		.875
		x2
1		.75
		x2
1		.5
		x2
1		.0



0 1 1 0 0 1 0 1 . 0 0 1 1 1

Suppose a new MACRO instruction was added to the set in the book. This new instruction is called **ANDD (and direct)** and will use a 4 bit op code and a 12 bit address to specify a location in memory that is to be **bitwise and'd** with the accumulator, **leaving the result in the accumulator**.

Assume that microcode line 100 is the starting point for you code

```
. . .  
. . . previous microcode  
. . .
```

```
100:mar := ir;rd;      {IR has operand adr}  
101:rd;               {get operand into MBR}  
102:ac := band(ac,mbr); {and AC with MBR}  
103:goto 0;          {go get next instr}
```