$X_{\wedge}$ XXX XXXX XXXX

1. Information representation. We gotta have some question about number representation. Consider number system(s) that contain 12 bits, with the radix point just to the right of the MSB. For that arrangement of bits, fill in the missing elements of the following table. (Remember that maximum is right-most on the number line; minimum is left-most on the number line.)

Value	Unsigned binary pattern	Twos-complement pattern
Maximum	$1_{\wedge}$ 111 1111 1111 ✓	$0_{\wedge}$ 111 1111 1111 ✓
Minimum	$0_{\wedge}$ 000 0000 0000 ✓	$1_{\wedge}$ 000 0000 0000 ✓
$^{37}/_{64}$	$0_{\wedge}$ 100 1010 0000 ✓	$0_{\wedge}$ 100 1010 0000 ✓
$^{-23}/_{32}$	N/A	$1_{\wedge}$ 010 0100 0000 ✓
$1^{11}/_{16}$	$1_{\wedge}$ 101 1000 0000 ✓	N/A
	N/A	100101000000

-6 -4 -1 $1/2$ $1/64$ $1/16$ $\frac{32}{64} + \frac{1}{64} + \frac{4}{64}$ $\frac{1}{32} + \frac{16}{32} + \frac{4}{32}$ $\frac{1}{32} + \frac{1}{2} + \frac{1}{8} + \frac{1}{16}$ $0_{\wedge}$101 1100 0000

copy

$11/16 = \frac{8}{16} + \frac{1}{16} + \frac{2}{16} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16}$

2. General ~~information~~ question:

a) Where is the return address stored on a go-to-subroutine instruction?

LINK REGISTER (LR) ✓

b) Give a sequence of instructions (only 2 needed) that will set up the system to expect the interrupt table to be found at the third legal location for the table. That is, what is the third legal location for the interrupt table, and how do you set it up? 3RD LEGAL VALUE = 0X00020000

```
lis  r31, 0x0002 ✓
mtevpr r31
```

c) We have programmed the system recognizing that the Interrupt Controller directs the hardware to utilize the instructions found at 0x0500. If we had enough interrupts to utilize the full capabilities of the module, how many different interrupt sources could be handled by the Interrupt Controller? (Hint: it is more than the four modules we have dealt with this semester.)

I BELIEVE WE COULD STORE A FULL WORD TO THE IER

SO 32 ✓

d) Assume R14 contains the value 0x00100000. Consider the instruction `stw r13,<offset>(r14)`. What is the address of the word which is located as far away as possible – to higher addresses – that can be accessed by this instruction?

d is A 16 BIT FIELD SO MAX

0X0010 FFFF ✓ ~~0X0010 7FFC~~

e) How many different periods are available via the Fixed Interval Timer?

4 ✓

3. Interrupt Controller Question: In the table below – which shows registers in the interrupt controller - identify the registers that need to be initialized, and give values for each. In this interrupt system, there are four interrupt sources, starting in the least significant bit position, and all are to be enabled. Also, the software activation of interrupts is not to be utilized. Assume that you want to assert the appropriate bits to reset any flags that may have remained from an earlier program.

Addr Offset	Register	Bit Pattern
0x00	ISR	
0x04	IPR	
0x08	IER	0X 000 F
0x0C	IAR	
0x1C	MER	0X 0003

0x0F

Now, in the space provided below, give instructions that will establish the bit patterns given above as well as to a)set the EVPR to zero and b)set up any enabling activity needed to allow interrupts to occur. Assume that the interrupt controller has been located at address **0x82340000**.

```
.org 0x3000
li   r31, 0        # LOAD 0
mtevpr r31         # STORE TO EVPR
li r31, 0x000F     # LOAD 0b1111
lis r30, 0x8234    # LOAD INTC
stw r31, 0x08(r30) # ENABLE 4 EXT INTERRUPTS
li r31, 0x0003     # BIT PATTERN FOR MER
stw r31, 0x1C(r30) # STORE TO MER
wrteei 1
:
:
```

INIT

STEADY STATE

4. ISR question: For system of problem three (Interrupt Controller, 4 bits …) create an Interrupt Service Routine for the following situation. Timer Module is hooked to the most significant bit of the four identified in the question. When the timer service is requested, reset the appropriate flags, in the appropriate order, increment the value in R23 and send to the LEDs. The Interrupt controller address is identified in Problem 3. The Timer Module is located at **0x82440000**, and the LED interface GPIO is located at **0x82560000**. Do not worry about register volatility.

1000
↑
MSB

```
.org 0x500
lis r19, 0x8234      # POINTER TO INTC
lis r20, 0x8244      # POINTER TO TIMER
lis r21, 0x8256      # POINTER TO LEDS
lwz r18, 0(r19)      # DOWNLOAD STATUS REG
andi. r18, r18, 8    # CHECK FOR timer INT   use IPR
bt 2, AROUND         # BRANCH IF NOT TIMER
li r18, 0b1000       # LOAD TO CLEAR IAR
stw r18, 0x0C(r19)   # CLEAR IAR
addi r23, r23, 1     # ADD1 TO LEDs
stw r23, 0(r21)      # STORE TO LEDs
```

clear flag in timer

```
AROUND :
   :
   :
rfi
```

SEND 0x102 TO TIMER TO CLEAR FLAG

look again

I ASSUMED THAT THE TIMER WAS SET UP USING BIT PATTERNS 0x2F6 followed by 0x206. THIS WILL ENABLE INTERUPTS AND AUTO RELOAD so NO WORK NEEDED IN ISR but need to clear flag.

Sent to TCSR

5. Data structure question. The recursive definition of the Fibonacci sequence is: F(n) = F(n-1) + F(n-2), with F(0) = 0 and F(1) = 1. Give code for a subroutine that implements this sequence. Use a stack mechanism to implement the recursion. CALLED FIB

```
# PARAMETERS PASSED IN
# r21 - VALUE OF N
# r22 - LOCATION OF STACK

FIB: li r31, 0x100      # LOCATION for link reg
     mflr r30           # GET LINK reg VALUE
     stw r30, 0(r31)    # Store link reg
     li r1, 0           # NUM ELEMENTS IN STACK
     li r2, 0           # STACK OFFSET
SEQ: cmpwi 0, r1, r21   # COMPARE # to N ?
     bt 2, done         # Branch if done
     cmpwi 0, r1, 0     # LOOK FOR 0
     bf 2, CHK1         # branch AROUND ≠0
     li r3, 0           # LOAD 0
     stwx r3, r22, r2   # STORE to STACK
     addi r2, r2, 4     # INC OFFSET
     addi r1, r1, 1     # INC NUM
     b SEQ
CHK1: cmpwi 0, r1, 1    # CHECK 1
     bf 2, NUM          # BRANCH IF ≠1
     li r3, 1           # LOAD 1
     stwx r3, r22, r2   # STORE
     addi r2, r2, 4     # INC OFFSET
     addi r1, r1, 1     # INC #
     b SEQ
```

I cannot see how this solution is recursive

→

```
NUM: li r5, 4
     subf r6, r5, r2    # GET OFFSET n-1
     lwzx r7, r22, r6   # LOAD n-1
     subf r6, r5, r6    # GET OFFSET n-2
     lwzx r8, r22, r6   # LOAD n-2
     add r7, r7, r8     # f(n) = f(n-1) + F(n-2)
     stwx r7, r22, r2   # STORE VALUE
     addi r1, r1, 1     # INC # in STACK
     addi r2, r2, 4     # INC OFFSET
     b SEQ

done: li r30, 0(r31)    # LOAD LINK reg
      mtlr r30          # MOVE TO LINK reg
      blr               # b LINK reg
```

2
4
8
16 17
34
68 69
138

6. Instruction coding question:

a) What instruction is represented in hexadecimal as: **0x7FD9E114**?

0111 1111 1101 1001 1110 0001 0001 0100

31 30 25 28 0 138 0

NO record

addc r30, r25, r28 ✓

b) Give the coding for the instruction '**andis. r2,r3,0x78AB**'. Remember that on logical instructions, register order is reversed in bit pattern.

29 RS RA VIMM

011101 r3 r2 0x78AB

00011 00010

7 8 6 2 ✓ ✓

0x78627 8AB

c) Wat instruction, located at address 0x4444, is represented by **0x4811EDCC**?

0100 1000 0001 0001 1110 1101 1100 1100

18 0 4 7 B 7 3

b

0x4444

0x047B73

+

0x 4CFB7

THIS IS A BRANCH INSTRUCTION THAT WHEN ACCESSED AT 0X4444 WILL BRANCH TO 0X4CFB7

0x0011EDCC

+ 0x00604444

0x00123210