Exam 2 - 1 EECE 344 Fall 2008

Name:

Problem	Possible	Score
1	15	12
2	10	8
3	15	13
4	20	13
5	20	14
6	20	14
Total	100	74

General information that may be useful sometime during test:

Table of Powers of Two

N	2^N	N	2^N	N	2^N	N	2^N
0	1	8	256	16	65,536	24	16,777,216
1	2	9	512	17	131,072	25	33,554,432
2	4	10	1,024	18	262,144	26	67,108,864
3	8	11	2,048	19	524,288	27	134,217,728
4	16	12	4,096	20	1,048,576	28	268,435,456
5	32	13	8,192	21	2,097,152	29	536,870,912
6	64	14	16,384	22	4,194,304	30	1,073,741,824
7	128	15	32,768	23	8,388,608	31	2,147,483,648

1. Information representation. We gotta have some question about number representation. Consider number system(s) that contain 10 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	11 1111 1111	01 1111 1111
Minimum	00 0000 0000	11 1111 1111 (-1)
13/32	0.0110100000	0.01101 0000
-3/16	N/A	1.10110 0000 (-1)
1¾	1.1100 0000	N/A
-352	N/A	1010100000

2. General information question:
   a) How does a programmer preserve the values of registers for "normal" processing when an interrupt is encountered? Give a two instruction sequence that will preserve registers R16-R31.

   Copy the registers using (stmw) instruction to known location in memory and reload using lmw

   stmw r16
   lmw r16 (-1)

   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? 0x00020000

   lis r10, 0x0002
   mtspr r10, EVPR

   c) When a branch-to-subroutine is encountered, where does the system store the address to which the subroutine should return?

   link register

   d) Assume a conditional branch is located at address 0x00010000. What is the highest address that can serve as the target of the branch? That is, what is the highest address that can be reached?

   0x00010000
   + 0000FFFF
   0x0001FFFF not quite (-1)

   e) What is the purpose of the watchdog timer interrupt?

   The watchdog timer interrupt is used to prevent infinite loop by causing an exception

3. Interrupt Controller Question: In the table below, 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 (Status)	0x 0000 000F
0x04	IPR (Pending)	read only
0x08	IER (Enable)	0x 000 000F
0x0C	IAR (acknowledge)	0x 0000 000F
0x1C	MER	0x 0000 0003

Now, in the space provided below, give instructions that will establish the bit patterns given above as well as to a)set up the vector register (to **0x000A0000**) and b)set up any enabling activity needed to allow interrupts in general. Assume that the interrupt controller has been located at address **0x84440000.**

```
.set ISR, 0x0
.SET IPR, 0x4
.SET IER, 0x8
.SET IAR, 0xC
.SET MER, 0x1C
.SET EVPR, ...

.org 3000
lis r10, 0x8444          # Pointer to interrupt controller
lis r11, 0x000A          # Pointer to vector table
li r13, 0xF              # Pattern for enabling activity
li r14, 0x3              # Pattern for MER
stw r13, ISR(r10)        # Clear previous interrupts
stw r13, IER(r10)        # Enable interrupts
stw r13, IAR(r10)        # Clear interrupt
stw r14, MER(r10)        # Enable Hardware Interrupt
mtspr r11, EVPR          # set up vector register to 0x000A0000
wrteei 1                 # set EE bit Enable External Interrupt
```

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, increment the value in R11 and send the value to the LEDs. The Interrupt controller address is identified in Problem 3. The Timer Module is located at `0x84460000`, and the LED interface GPIO is located at `0x84480000`. Do not worry about register volatility.

```
.set ISR
.set IRR
.set IER
.set IAR
li r11, 0x0     # r11 initialized to zero outside of routine
.org 0x500      # address for external interrupt service routine
li r20, 0x8     # Pattern to test MSB of the four
lis r21, 0x8446 # Pointer to Timer Module
lis r22, 0x8448 # Pointer to LEDs
li r23, 0x0     # pattern to set GPIO as output
stw r23, 4(r22) # Set GPIO as output.
lis r24, 0x8444 # pointer to interrupt controller
lwz r25, IRR(r24) # check for Interrupt
and r20, r25, r25 # compare
bne end         # if not equal, no interrupt, so leave routine
addi r11, 0x1   # increment value in r11
stw r11, 0(r22) # send out to LEDs
stw r20, IAR(r24) # clear interrupt
end: rfi        # return from interrupt
```

need some more scaffolding

5. Data structure question. Consider the situation where a user has an array of words. The operation that is to be performed on this array of data is to sum all of the elements. The array starts at address `0xFF000100` (and progresses to higher addresses). Create code that will calculate the sum of the first 600 values of this array. Assume that there will be no overflow because of the sum.

```
.org 0x3000
lis r10, 0xFF00
ori r10, r10, 0x0100   # pointer to array
lwz r11, 0(r10)        # load first value
li r12, 600            # num of values
mtctr r12              # load the counter with number of values
loop: addi r10, r10, 0x4   # increment pointer to next word.
lwz r13, 0(r10)        # load next value
add r11, r13, r13      # Running Total is in r11
bdnz loop              # continue until all 600 values summed
```

look again
got oops here

6. Another strange arithmetic/I/O question. Consider a system that has the following characteristics (that are germane to this question): GPIO module located at address `0x86410000` hooked to 32 LEDs (like in lab). GPIO module located at address `0x86420000` hooked to another 32 LEDs (like in lab). A third GPIO module located at `0x86430000` attached to 32 wires from unknown source. Mailbox assigned to address `0x00023200` (one word mailbox). The mailbox provides synchronization only. When the mailbox is non-zero, data is available on the wires of the third GPIO module. So, in the space below provide code to set up and do the following. When data is available, read the value and compare to maximum and minimum. Send maximum values to LEDs at first address. Send minimum value to second address. After 500 values, quit.

```
        li r13,0x0          # Count
        lis r10,0x7FFF      # MAX Positive Value
        ori r10,r10,0xFFFF
        lis r11,0xFFFF      # MIN neg Value
        ori r11,r11,0xFFFF
        lis r1, 0x8641      # Pointer to 1st set of 32 LEDs
        lis r2, 0x8642      # pointer to 2nd set of 32 LEDs
        lis r3, 0x8643      # pointer to 3rd GPIO Module
        lis r4, 0x0002@h    # } Pointer to mailbox
        ori r4,r4 0x3200@l  # }
        li r5, 0x0          # Pattern to set GPIO as Output
        li r6, 0xFFFF       # Pattern to Set GPIO as Input
        ori r6, r6, 0xFFFF
        stw r5, 4(r1)       # 1st GPIO set to Output LED
        stw r5, 4(r2)       # 2nd  "   "   "   "   "
        stw r6, 4(r3)       # 3rd GPIO set as Input
check:  li r7, 0(r4)        # Check the Mail
        cmp 0,0,r7,r5       # if Mail is zero
        bne check           # keep checking
        lt r8, 0(r3)        # if mail box is non zero read 3rd GPIO into r8
        cmp 1,0,r8,r10      # Compare to Max
        bgt 1, Max          #
        cmp 2,0,r8,r11      # Compare to Min
        blt 2, Min          #
Max:    mr r10, r8          # update max value
        stw r10, 0(r1)      # Send out to LEDs 1st set
        addi r13, 0x1       # increment count
        b check1            # check for next input
Min:    mr r11, r8          # update min value
        stw r10, 0(r2)      # Send out to LED 2nd set
        addi r13, 0x1       # increment
Check1: cmpi , r13, 500     # check for 500 values
        bne check           # if not keep checking input
```

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.)

Partial Credit?

twos complement is just inverting all the bits and then adding a 1 to the LSB not so

I thought we would be doing some IEEE floating point

Value	Unsigned binary pattern	Twos-complement pattern
Maximum	1111 1111 1111	0111 1111 1111
Minimum	0000 0000 0000	1000 0000 0000
27/64 = .421875	0.01101100 ✓	0011 0110 0000
-5/16	N/A	1101 1000 0000
1¾ = 1.75	1.1100 ✓	N/A (hidden sign bit)
~~1400~~	N/A	101010000000

2. General information question:
   a) How does a programmer preserve the values of registers for "normal" processing when an interrupt is encountered?

Copy the registers to be changed to a known area, then restore them when leaving the interrupt service routine use: stmw and lmw.

   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 second legal location for the table. That is, what is the second legal location for the interrupt table, and how do you set it up?

(-2) 0xaaa10000 well the second legal loc. is 0xaaa20000 not so

You set the offset of the vector EVPR and it goes to an ISR that will interupt the fetch/decode, execute cycle.

   c) We have programmed the system recognizing that the Interrupt Controller directs the hardware to utilize the instructions found at 0x500. If we had enough interrupts to keep it busy, how many different interrupt sources could be handled by the Interrupt Controller?

well we have 32 bits so it follows

32 is the answer but since we are already at the 16th bit the EE bit, I would say 32-16=16 -1

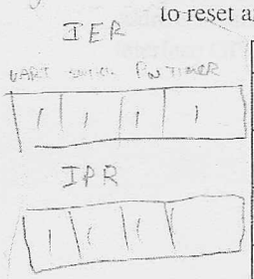
   d) What are the different types of instructions? Identify the types and give an example of each.

- work ✓ add, subtract, multiply, divide, AND, OR, EXOR
- movement ✓ jump to subroutine, conditional jump
- control ✓
- ~~Regs.~~

   e) What is the purpose of the watchdog timer interrupt?

The watchdog timer can have a value set in it so that the timer doesn't get stuck in an infinite loop.

3. Interrupt Controller Question: In the table below, 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	0000
0x04	IPR	1111 ← but, read only!
0x08	IER	1111
0x0C	IAR	0000 need this to be 1111 to clear flags
0x1C	MER	11

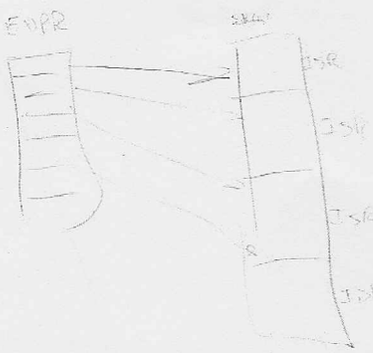
Now, in the space provided below, give instructions that will establish the bit patterns given above as well as to a)set up the vector register and b)set up any enabling activity needed to allow interrupts in general. Assume that the interrupt controller has been located at address 0x82340000.

Plus the offset

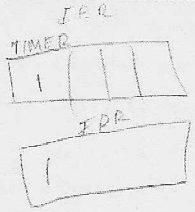
NOTE: EVERY TIME I ATTEMPT TO WRITE CODE ON THE SPOT IT IS WRONG. THIS IS BECAUSE I HAVE LEARNED EVERYTHING I KNOW ABOUT CODING IN THE LAB VIA TRIAL AND ERROR SO I WILL FOCUS ON CONCEPTUAL EXPLANATIONS

The IPR is going to indicate which interrupts are being [illegible] by having a bit pattern where a 1 represents an interrupt condition. The IER will indicate which interrupts I will be paying attention to. For example if the IPR bit pattern is 1011 and the user hits the switch to cause an interrupt, it's not going to happen because I haven't enabled the bits to enable that interrupt.

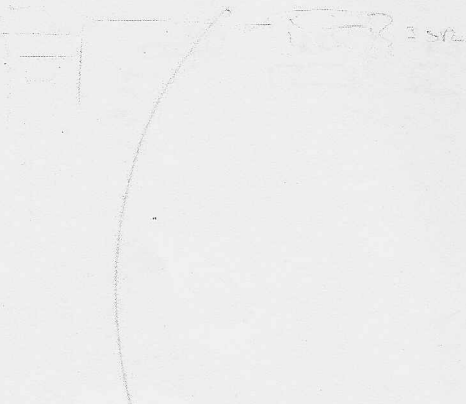
I'M GOING TO SET A REG TO HOLD VALUE FOR THE ISR WHICH IS DETERMINED BY THE BIT PATTERNS ABOVE, FOR EACH CASE.



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.



we will have to be constantly reading the registers after each ISR if we don't want volatility. But we don't have to worry about that as it says here



why?

then begin to use registers for adding ISR value

5. Data structure question. Consider the situation where a user has an array of words. The operation that is to be performed on this array of data is to sum all of the elements. The array starts at address `0xFFFF0100` (and progresses to higher addresses). Create code that will calculate the sum of the first 800 values of this array.

STACK TIME with counter

① set the register to point at 0xFFFF0100 hex

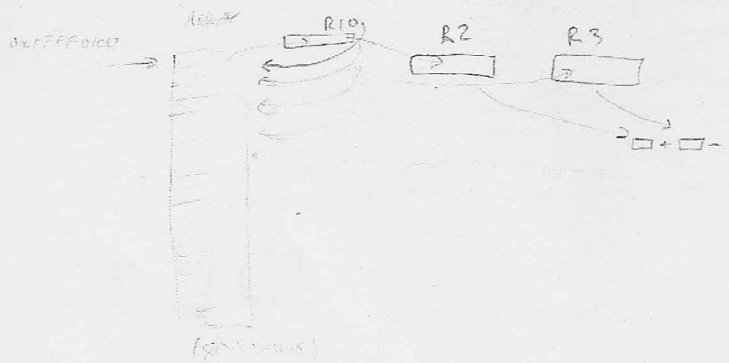
② The pointer will take the value of that location in memory and put it in the register (R2)

③ assign 2 registers to hold values one will hold the value the pointer gives it and the other will hold the result of the sum of the values R2 + R3,

④ the offset of R10 will be incremented to move through the array

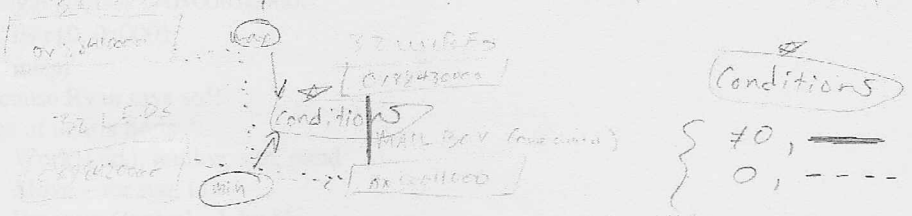
⑤ Add the R2 + R3 and put result into R3

⑥ Repeat 800 times so there will be a counter as well



6. Another strange arithmetic/I/O question. Consider a system that has the following characteristics (that are germane to this question): GPIO module located at address `0x88410000` hooked to 32 LEDs (like in lab). GPIO module located at address `0x88420000` hooked to another 32 LEDs (like in lab). A third GPIO module located at `0x88430000` attached to 32 wires from unknown source. Mailbox assigned to address `0x00011000` (one word mailbox). The mailbox provides synchronization only. When the mailbox is non-zero, data is available on the third GPIO module. So, in the space below provide code to set up and do the following. When data is available, read the value and compare to maximum and minimum. Send maximum values to LEDs at first address. Send minimum value to second address. After 500 values, quit.

Key = Branch Conditional



So the 32 wires send a binary Digital logic signal to mail box
which records it and sends it to either 0x88410000 with the bit pattern
of if it's a min it goes to 0x88420000 with that pattern that is where
the branch conditional and repeat 500 times so there will be a counter

but, how?

I AM PRAYING FOR PARTIAL CREDIT THROUGHOUT THIS EXAM. yes, but essay answers to specific questions are not helping

2nd Exam Hints

1.

Value	Unsigned Binary Pattern	Two's-Compliment Pattern
Maximum	1111 1111 1111	0111 1111 1111
Minimum	0000 0000 0000	1000 0000 0000
27/64	0.011 0011 0000	0011 0110 0000
-5/16	N/A	1.101 1000 0000
1 ¾	1.110 0000 0000	N/A
-1408	N/A	1010 1000 0000

2.

a) Copy the registers to be changed to a known area, and then restore them on leaving the interrupt service routine. Use stmw and lmw

b) Point of computer architecture: PPC requires EVPR be at an address that is a multiple of 646k. So second legal address is 0x00010000.
   - lis r10, 0x0001
   - mtspr

c) 32 – Because Ryan says so!!

d) For types of instructions
   1) Work – add, mulhw, sub, nand
   2) Move – mr stw, ldz
   3) Program Control – b,bc,blr
   4) System Control – mtspr, mfspr, rfi

e) Prevent infinite loop

3.

```
.set IER
.set IAR
.set MER
.set EVPR
.set Address
.org ox3000

lis r10, 0x8234    # r10 is pointer
lis r11, Adress            # r11 address of table
mtspr EVPR, r11

li r12, 0xf                # get r12 ready to set
stw r12, IER (10) # F to IER
stw r12, IAR               # reset bits

li r13, 3          # set LSBs of
stw r13, MER               # MER
wrteii
```

4.
```
        .set IPR, 4
        .set IAR, 0xC
        .org 0x0500         # ISR address

        lis r20, 0x8234     # point at interrupt controller
        lis r24, 0x8244     # point at timer module

        lwz r21, IPR,(r20) # get pending register
        andi r22,r21,0x8   # check on bit

        beq go_back         # not set, get out
        li r24, 0x106       # SEE BELOW JERAMIE ?
        stw r29, 0(r24)     # send to TCR0
        li r25, 0x8         # get bit for flag
        stw r25,IAR(r20) # reset bit in interrupt controller
        lis r26, 0x8256     # now point at LEDs
        addi r23, r23,1     # bump pointer
        stw r22,0(r26)      # send out
go_back: rfi                # return from interrupt
```

BITS IN TIMER (for JERAMIE)

21	22	23	24	25	26	27	28	29	30	31
0	0	1	1	1	0	1	0	1	1	1
		1			D			7		

5.
```
        .org 0x3000
        lis r10,0xffff      # MSB of 0xffff0100
        ori r10,r10,0x0100 # LSB of  0x0100
        li r11,  0          # r11 will be sum
        li r12, 800         # set up for 800 iterations
        mtcr r12            # load counter
again:  lwz r13, 0(r10)     # get value from array
        add r11,r11,r13     # add to running sum
        addi r10,r10,4      # poing to next element
        bdnz again          # loop back if not done
        nop                 # done, answer is in r11
```

6.
```
        .org 0x3000     START
        li r9, 0
        lis r10, 0x8841  LED1
        lis r11, 0x8842  LED2
        lis r12, 0x8843  GPIO
        lis r13, 0x0001@h  > MAILBOX
        ori r13,r13, 0x1000@l
        lis r14, 0x8000@h  > min VAL
        ori r14,r14, 0x0001@l
        lis r15, 0x7fff@h  > MAX VAL
        ori r15,r15, 0xffff@l
        li r6, 0
        stw r16,4(r10)  — Clear LED's
        stw r16,4(r11)  /
        nand r17,r16,r16 — NAND 0's
        stw r17,4(r12) — Store 1's to GPIO to set as outputs
```

```
again:  lwz r17,0(r13)     load mailbox Value
        cmpi 1,0,r17,0     Compare mailbox to Zero
        beq 1, again       if Zero loop again
        stw r16,0(r13)     if not Zero Clear MB
        lwz r18,0(r12)     load GPIO Value
        cmp 1,0,r18,r14    Compare min Value to GPIO Value
        blt 1, chk1        if Greater than branch to chk1
        mr r14,r18         move GPIO Value to R14
        stw r14,0(r10)     Store R14 TO LED1
        b go_on            branch
chk1:   cmp 1,0,r18,r15    if greater than, Compare GPIO Value to MAX
        bgt 1, go_on       if Value in GPIO is greater than max move on
        mr r15,r18         if not move GPIO to R15
        stw r15,0(r11)     Store Value to LED2
go_on:  addi r9,r9,1       increment counter
        cmpi 1,0,r9,500    compare to 500
        bne again          if not 500 loop again
        nop                if 500 end.
```

ECE-344 Final Exam Solutions

1.

Two's Complement Pattern	10011000.10010000	-103.34375
Unsigned Binary Pattern	10011000.10010000	152.5625
Two's Complement Pattern	1011101110000000	24,000
Bit Two's Complement	11111000.10101000	-7.34375
Bit Two's Complement	11010010.01111000	-45 17/32
Unsigned Binary	10101010.10101010	170.664063

2.

a) fundamental principle on which stored program computers are based:

Fetch, decode, execute

b) how is Memory Mapped I/O implemented?

I/O activity is carried out by writing and reading locations in memory.

I/O elements are assigned addresses in the memory space of the processor

c) what instructions move multiple register values to and from memory?

lmw, stmw

d) vector table starts at 0x00100000, ISR is created to handle privilege violations,

where should the ISR be located?

0x00100700

e) give a two instruction sequence to look for the third legal location of interrupt table

lis r3, 0x0002

mtevpr r3

f) why does PowerPC have both a user mode and a privilege mode?

for protection and to manage the resources in the operating system

g) how does the system enter user mode?

Setting PR bit to 1, in the MSR

li r2 0x00010000 mtspr msr r2

h) how far can a branch-to-subroutine go? Assume instruction is at 0x00010000.

0x07fffffc + 0x00010000 = 0x0800fffc

3. registers changed:

r1 = 0x11111118

r2 = 0x00000008

r7 = 0xFFFFFFFF

r9 = 0xBBBBBBBB

r10 = 0x12345678

LR = 0x3008

4. registers changed:

r0 = 0xDDDDDDDD

r3 = 0x54321065

r4 = 0xFFFFFFFF

r13 = 0xFFCCCCCC

r14 = 0x40000400

r15 = 0x0000EFFE

memory changed:

address: 0x4000042c → 99 99 99 99

address: 0x40000432 → 88

5. Interrupt Question #1: Provide code to configure the ML403 as we had in the lab, but fixed so that the only active interrupt is from the UART.

lis r1, 0x83e0

lis r2, 0x80

stw r2, 0x100C(r1) #LCR

li r3, 0x45

```
stw r3, 0x1000(r1) #DLL

li r3, 0x01

stw r3, 0x1004(r1) #DLM

li r3, 0x7

stw r3, 0x100C(r1)

li r3, 0x8

stw r3, 0x1004(r1)

lis r4, 0x8140

li r0, 0

stw r0, 4(r4)

lis r6, 0x8141

stw r0, 4(r6)

lis r7, 0x8142

stw r0, 4(r7)

lis r8, 0x8143

stw r0, 0(r8)

lis r10, 0x8180 #INTC

li r1, 0x3

stw r1, 0xC(r10) #MER

li r1, 0x8

stw r1, 0xC(r10) #IER

li r9, 0xF

stw r9, 0(r10) #ISR

lis r22, 0020

mtevpr r22
```

wrteei 1

b here

6. Interrupt Question #2: deals with steady state activities from Question 5; give code for the ISR for the UART

lis r6, 0x8180 #INTC

lis r5, 0x83e0 #UART

.org 0x500

lwz r1, 0x1000(r5) #load from RBR

lis r2, 0xffff

ori r2, r2, 0x0100

stw r1, 0(r2)

li r3, 0x60

stw r3, 0x1014 #LSR

li r3, 0x8

stw r3, 0xC(r6) #IAR

rfi

7. Provide two sets of instructions, one for the "push" operation and one for the "pop" operation

Push:

.org 0xhe11 (for Mike)

lis r1, 0x2000

stw r3, 0x10(r1)

stw r8, c(r1)

stw r14, 8(r1)

stw r22, 4(r1)

stw r25, 0(r1)

Pop:

.org 0x0200

lwz r25, 0(r1)

addi r1, r1, 4

lwz r22, 4(r1)

addi r1, r1, 4

lwz r14, 8(r1)

addi r1, r1, 4

lwz r8, c(r1)

addi r1, r1, 4

lwz r3, 0x10(r1)

8.

a) What is the instruction represented by the bit pattern 0x38857342?

addi r4, r5, 0x7342

b) what is the instruction represented by 0x48001511 located at 00004000?

bl 0x1510

0x1510 away from 0x00004000

c) give the coding for the instruction xor. r3, r4, r5

0x7C832A79