Exam 1 - 1 EECE 344 Fall 2010

Name: [redacted]

Problem	Possible	Score
1	20	20
2	20	20
3	20	20
4	15	15
5	15	13
6	15	15
Total	5	103

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

Please write legibly.

1. General information question:

   a) What is the basic tenet of all stored program computers?

   fetch, decode, execute

   b) Identify the four different types of instructions and give an example of each from the PowerPC instruction set.

work	movement	program control	system control
add	lwz	b	mtspr
(technically nop counts on PPC)			

   c) When a non-critical interrupt occurs, where is the current value of the Machine State Register stored?

   SRR1

   d) We have used a mnemonic instruction `lis` to load the upper 16 bits of a register. What is the instruction that is actually invoked in order to do this work?

   lis rx, n

   addis rx, 0, n

   e) Assume that the 4 LEDs at the edge of the trainer board have been associated with the base address of 0x81420000. What value at what address is used to make sure the pins associated with the 4 LEDs are established as outputs?

   DDR = base + 4 = 0x 8142 0004

   write 0 → establish output

   f) Assume that register 7 contains 0x00008080. What address is accessed by the instruction '`ldw r6, 0x1140(r7)`'?

```
  0x1140
+ 0x8080
--------
  0x91C0
```

2. Subroutine question: A programmer wrote a small subroutine to wait for the RxFIFOValidData flag of the Uartlite system to be set; then to clear the bit and return. This subroutine was used in a system that needed a UART, but without using the interrupt system. The programmer called this routine from a larger system handling routine. This code fragment is as follows:

```
Addr   Bits                Instr
4400 60000000            nop
4404 4800003D            bl getc
4408 60000000            nop

4440 3D008400 getc:      lis r8,0x8400          # UART at addr 0x84000000
4444 81280008 again:     lwz r9,0x8(r8)         # Stat reg offset by 8 bytes
4448 712A0001            andi. r10,r9,0x0001
444c 4182FFF8            bt 2,again   beq again # bit 2 is equal
4450 81680000            lwz r11,0(r8)          # Receive FIFO offset by 0 bytes
4454 2C8B0033            cmpwi 1,r11,0x0033     # 0x33 is ascii '3'  cmpwi cr1,r11,'3' also works in gas
4458 4086FFEC            bf 6,again   bne cr1,again # bit 6 is also equal
445c 4E800020            blr
```

This question deals with the registers used in the routine. Below is a before and after representation for 16 of the registers. The before values are given (values of registers before executing the instruction at 0x4404); fill in the after values (values of registers after returning from the subroutine, what system is like when PC points to 4408). The UART system is enabled and configured to the right baud rate, etc, but to cause no interrupts. Only mark in the *After* area those registers that have been changed by the above code fragment, and in those boxes place the correct value for the register.

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 =	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 =	r3 =
r4 = 0x44444444	r5 = 0x55555555	r4 =	r5 =
r6 = 0x66666666	r7 = 0x77777777	r6 =	r7 =
r8 = 0x88888888	r9 = 0x99999999	r8 = 0x 8400 0000 ✓	r9 = 1 ✓
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 = 1 ✓	r11 = 0x 33 ✓
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 =
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 =	r15 =
CR = 0x00000000	LR = 0x00000000	CR 0x 4200 0000	LR 0x 4408

assuming blr doesn't change LR, only reads it

we only return if '3' is typed
↳ ergo if we have a valid char

check cr0: = 1 → >0 → 0100 = 0x4
check cr1: = 0x33 → 0010 = 0x2

3. Data movement question: In the first laboratory you explored moving information to and from memory with the 'ld' and 'st' instructions. Below is a small code fragment, followed by another memory and register contents description. The code fragment is set up to be somewhat tricky, and not particularly straightforward, but implement the work of each instruction and you should be okay. Identify the locations in memory and the registers that are changed by the code fragment, and give the updated values.

```
Addr   Bits            Insruction

1100 3D400001          lis r10,0x0001
1104 614A3000          ori r10,r10,0x3000  ←   r10 ← 0x13000
1108 816A0018          lwz r11,0x18(r10)  ←    r11 ← (0x13018) w
110c A1CA0012          lhz r14,0x12(r10)  ←    r14 ← (0x13012) h
1110 89EA000F          lbz r15,0x0F(r10)  ←    r15 ← (0x1300F) b
1114 992A0037          stb r9,0x37(r10)
1118 B0EA002A          sth r7,0x2A(r10)
111c 90AA0030          stw r5,0x30(r10)
```

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 =	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 =	r3 =
r4 = 0x44444444	r5 = 0x55555555	r4 =	r5 =
r6 = 0x66666666	r7 = 0x77777777	r6 =	r7 =
r8 = 0x88888888	r9 = 0x99999999	r8 =	r9 =
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 = 0x13000 ✓	r11 = 0x12131415 ✓
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 =
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 = 0xAABB ✓	r15 = 0x77 ✓

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00013000	01	23	45	56	89	AB	CD	EF	00	11	22	33	44	55	66	77
00013010	88	99	AA	BB	CC	DD	EE	FF	12	13	14	15	16	17	18	19
00013020											77	77 ✓				
00013030	55	55	55	55 ✓				99 ✓								

4. Instruction question: Assume that there is a data structure, an array of words, consisting of 1000 values, that starts at the address 0x00050000. These words are stored as 2's complement numbers. Write code in the space below that will look through the 1000 values and find out how many occurrences of either +110 or –110 are found in the array. Leave the count of number of values in register 4.

decimal

```
count vals:
            # load ctr
            li r31, 1000
            mtctr r31

            # base addr
            lis r31, 5
            # init count
            li r4, 0
loop:       lwz r30, 0(r31)
            cmpwi r30, 110
            bne nextchk # match if eq 110
match:      addi r4, r4, 1    # if match, increment count
            b endloop
nextchk:    cmpwi r30, -110  # match if eq -110.
            beq match
endloop:    addi r31, r31, 4  # point to next element in array
            bdnz loop

            nop  # ...
```

Good job

What bit sets UART ints in IER? 3?

5. Interrupt question: This question has two parts. The first is setup/initialization, the second is steady state. Assume that a Uartlite interface module has been configured at 19,200 baud, no parity, 8 data bits, just like we have used in class and in the laboratory. The address associated with the Uartlite is 0x84000000. The address associated with the Interrupt Controller is 0x81800000. In the space provided below, give instructions that will set up Uartlite and the PowerPC to allow Uartlite interrupts to occur. (No other interrupts are to be enabled.) Configure the base register to specify the second legal value (we did first legal value in class, at location zero).

```
~~.set IER, 8~~
.set IAR, 12
.set MER, 0x1C

.set UART, 31     # use these
.set INTC, 30     # registers
.set Rx, 0        #indices for
.set Tx, 4        # mem-mapped
.set STAT, 8
.set CTRL, 12     # I/O

R31? R30?
lis UART, 0x8400
lis INTC, 0x8180

li r29, 0b10000
stw r29, CTRL (UART)

li r29, 0b1000          same bit
stw r29, ~~IER (INTC)~~
MER

# "base register" is evpr?
lis r29, 1
mtevpr r29
# I thought anything divisible by 65536 was legal for evpr

li r29, 0b11
stw r29, MER (INTC)

wrteei 1
```

For the second part of this question, give code for an interrupt service routine that will echo the character received.

```
.org 0x10500 →
               # assume the .sets above
               lis UART, 0x8400
               lis INTC, 0x8180
               lwz r29, STAT(UART)   # not sure we have to do this
               andi. r29, r29, 1     # (the fact that there's an interrupt may imply
               ~~beq endint~~            # that Rx has valid data), but let's be safe
               lwz r28, Rx (UART)
check tx empty: lwz r29, STAT (UART)
               andi. r29, r29, 0b100  # is there room in Tx?   don't need
               bne checktxempty
               stw r28, Tx (UART)

endint:
               li r29, 0b1000
               stw r29, IAR (INTC)
               rfi
```

good job

6. Information representation question:

Fill in the table below. The number of bits in each case is 16. Also note that there is a column for *p*, where p represents the location of the radix point. *p* = 0 is for whole numbers; *p* > 0 allows the system to represent fractional values. *p* specifies the number of bit positions to the left to move the radix point.

Representation Method	*p*	Value	Bit pattern
Unsigned Binary	0	384	0000 0001 1000 0000
Two's Complement	0	-5000	1110 1100 0111 1000 ✓
Unsigned Binary	8	80.125	0101 0000 0010 0000 ✓
Two's Complement	8	-80.125	1010 1111 1110 0000 ✓
Unsigned Binary	15	1.8125 ✓	1110 1000 0000 0000
Two's Complement	15	-0.0625	1111 ~~1000~~ 0000 0000 ✓

$$\begin{array}{r}128\\+256\\\hline 384\end{array}$$

$1+0.5+0.25+0.0625$ =

$80 = 64 + 16$
$= 2^6 + 2^4$

$.125 = \frac{1}{8}$

$0.0625 = \frac{1}{16}$

0.000 1000 0000 0000

1111 0111 1111 1111

1111 1000 0000 0000

$5000 = 4096 + 512 + 256 + 128 + 8$ 0001 0011 1000 1000
$\quad 2^{12} \quad 2^9 \quad 2^8 \quad 2^7 \; 2^3$

$$\begin{array}{r}5000\\-4096\\\hline 904\\512\\\hline 392\\256\\\hline 136\\128\\\hline 8\end{array}$$

→ 1110 1100 0111 0111
$\qquad\qquad\qquad\quad + 1$
1110 1100 0111 1000

check: most negative 1000 .0000 0000 0000 = -128

1010 1111 1110 0000 $= -128 + 1+2+4+8+32 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$

$= -128 + 47 + \frac{7}{8}$

$= -81 + \frac{7}{8} = -80 - \frac{1}{8}$

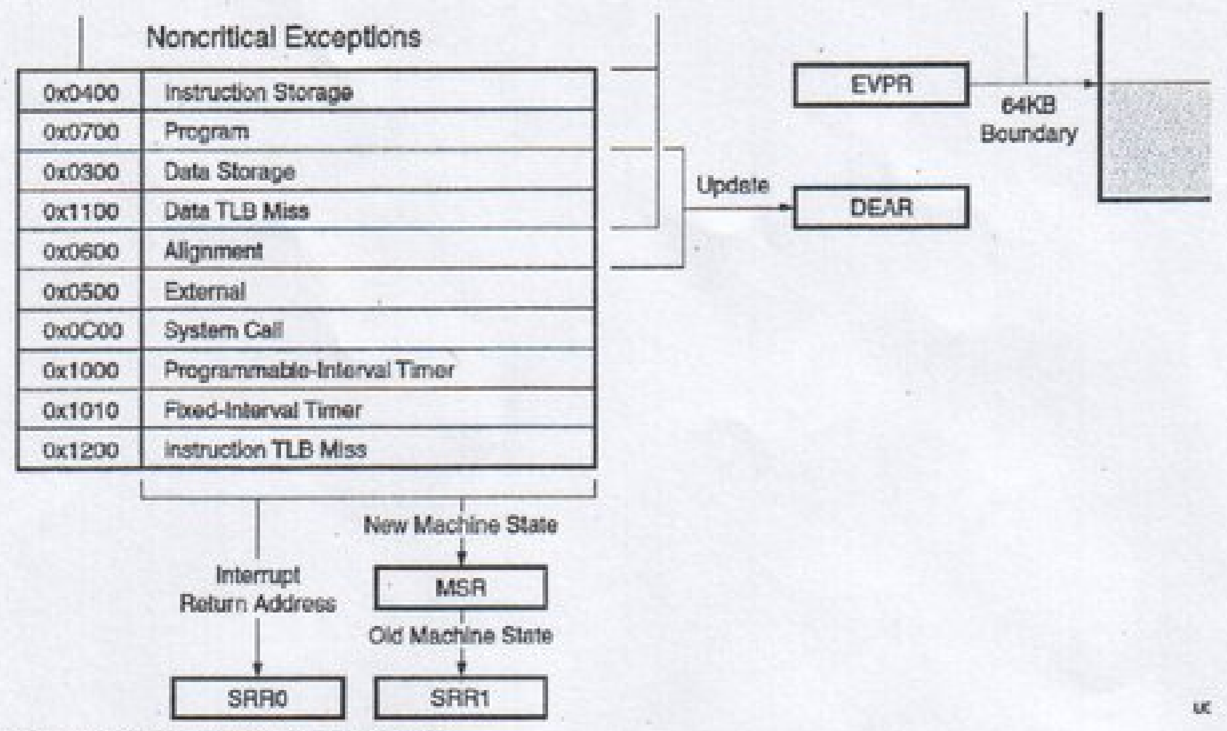

Table 4: **XPS UART Lite Registers**

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x0	Rx FIFO	Read	0x0	Receive Data FIFO
C_BASEADDR + 0x4	Tx FIFO	Write	0x0	Transmit Data FIFO
C_BASEADDR + 0x8	STAT_REG	Read	0x4	UART Lite Status Register
C_BASEADDR + 0xC	CTRL_REG	Write	0x0	UART Lite Control Register

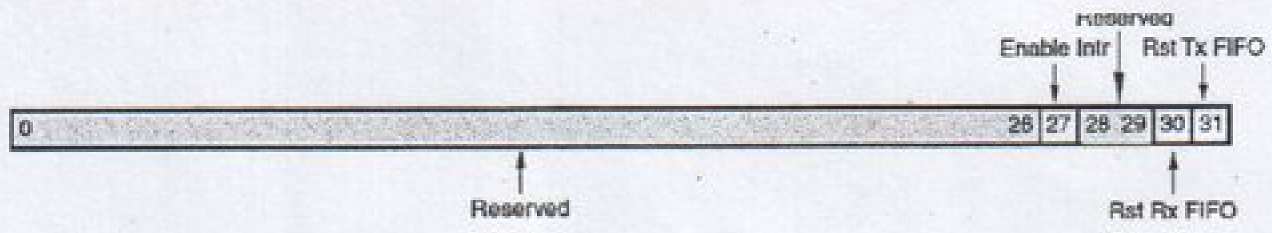

Figure 4: **UART Lite Control Register**

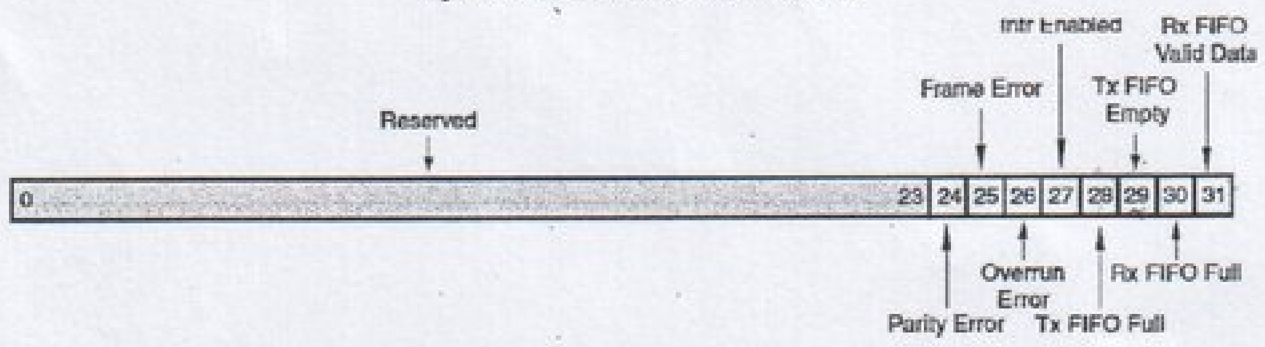

Figure 5: **UART Lite Status Register**

Table 4: **XPS INTC Registers and Base Address Offsets**

Register Name	Base Address + Offset (Hex)	Access Type	Abbreviation	Reset Value
Interrupt Status Register	C_BASEADDR + 0x0	Read / Write	ISR	All Zeros
Interrupt Pending Register	C_BASEADDR + 0x4	Read only	IPR	All Zeros
Interrupt Enable Register	C_BASEADDR + 0x8	Read / Write	IER	All Zeros
Interrupt Acknowledge Register	C_BASEADDR + 0xC	Write only	IAR	All Zeros
Set Interrupt Enable Bits	C_BASEADDR + 0x10	Write only	SIE	All Zeros
Clear Interrupt Enable Bits	C_BASEADDR + 0x14	Write only	CIE	All Zeros
Interrupt Vector Register	C_BASEADDR + 0x18	Read only	IVR	All Ones
Master Enable Register	C_BASEADDR + 0x1C	Read / Write	MER	All Zeros