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	stw	bl	mtctr

   c) The instruction `rfi` returns from normal interrupt (as opposed to critical interrupt). In terms of registers, what transfers occur? That is, what two registers transfers are carried out when the `rfi` instruction executes?

When the intrupt occured the address where the ISR was called was plased into the return register. When the ISR calls the rfi command that address is used to know where in the main program to return to. how? -1

SRR1 gets placed in MSR
SRR0 goes to program counter

   d) Assume that the GPIO interface of one of our 32 bit LED modules has been configured to respond to the address 0x83330000. Also assume that R10 contains the value 0x83330000 and that R11 contains zero. Give a three instruction sequence that will turn on all the LEDs of the module.

lis r1, 0x1111 ✗
ori, r1, r1, 0x1111 ✗
stw r1, 0(r10) -1

```
stw r11, 4(r10)   Setup DDR2
nor r11, r11, r11 invert R10
stw r11, 0(R10)   Store
```

   e) How does the system switch from privileged mode to user mode?

the MSR is updated to switch from priv to user mode. When PPC is powered on it is in priv mode allowing the operator to make the switch -1 How?

PR BIT IN MSR set PR BIT TO 1 to user mode

   f) Assume that register 12 contains 0x00011080. What address is accessed by the instruction '`lwz r13,0x400(r12)`'?

0x00011480 ✓

2. Subroutine question: A programmer wrote a small subroutine to wait for two different buttons to be pressed; the two buttons in question are those associated with the 0th bit and the 3rd bit (bit positions 0 and 3). The programmer tested for the least significant bit, then checked for the other bit. If both switches were set, then the programmer obtained the time from TBU,TBL (hence the mftbu and mftbl instructions). The fragment is shown here:

r4=0x84440000
r5=

```
Addr     Bits          Instr
2000   60000000        nop
2004   4800021D        bl button
2008   60000000        nop


2220   3C808444  button: lis r4,0x8444     # buttons at 0x84440000
2224   80A40000  chk1:   lwz r5,0(r4)   <- getting 1ST Value at this memory address
2228   70A60001          andi. r6,r5,1   <-
222c   4182FFF8          bt 2,chk1
2230   2C850009          cmpwi 1,r5,0x09
2234   4086FFF0          bf 6,chk1
2238   7D4D42E6          mftbu r10
223c   7D6C42E6          mftbl r11
2240   4E800020          blr
```

This question deals with the registers used in the routine. Below is a before and after representation for some of the registers. The before values are given (values of registers before executing the instruction at 0x2004); fill in the after values (values of registers after executing the instruction at 0x2004 and returning from the subroutine)–like what you expect to happen if you put a breakpoint at location 0x2008. The button GPIO module has been initialized elsewhere, so don't worry about that aspect. Only write 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. Assume that the successful read happened at time 0x0000123456781111.

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 = 0x00000001	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 =	r3 = 0x00000009
r4 = 0x44444444	r5 = 0x55555555	r4 = 0x84440000	r5 = 0x44444444 (-2)
r6 = 0x66666666	r7 = 0x77777777	r6 = 0x84440001 (-2)	r7 =
r8 = 0x88888888	r9 = 0x99999999	r8 =	r9 =
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 = 0x00001234	r11 = 0x56781111 (-4)
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 = VAL TBL	r13 =
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 =	r15 =
lr = 0xABCDEF00	ctr = 0x00000000	lr = 0x2008	ctr =
cr = 0xFFFFFFFF	tsr = 0x90000000	cr = 0x42000000	tsr =

0100 0010

3. Data movement question: In the first laboratory you explored moving information to and from memory with various load and store 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.

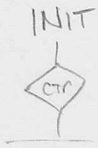
```
Addr      Bits       Insruction

10280   38603000  strt:   li r3,0x3000
10284   39C00004          li r14,4
10288   7C647030          slw r4,r3,r14   # shift left word
1028c   7C841A14          add r4,r4,r3
10290   38840040          addi r4,r4,0x40
10294   80C40014          lwz r6,20(r4)
10298   A0E40012          lhz r7,18(r4)
1029c   89040007          lbz r8,7(r4)
102a0   99240029          stb r9,0x29(r4)
102a4   B1440032          sth r10,0x32(r4)
102a8   9164003C          stw r11,0x3C(r4)
```

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 =	r11 =
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 =
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 =	r15 =

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00033040	01	23	45	56	89	AB	CD	EF	00	11	22	33	44	55	66	77
00033050	88	99	AA	BB	CC	DD	EE	FF	12	13	14	15	16	17	18	19
00033060																
00033070																

4. Instruction/data structure question: In the space provided below, write a code fragment that will create a loop (use the counter register to implement the loop) that will increment the individual elements of an array of words. The array starts at address 0x00024800 and continues for 0x18000 locations (that is 0x18000 words slots). Each element of the array is to be incremented by the value of 0x1234.




```
li r1, 0x24800      too big
li r2, 0x18000      seeks initialize counter
mtctr r2            # move to counter

Loop:
    addi r1,r1,0x1234

    bdnz loop
```

Doesn't say to store/load/print array elements.

yes, but how else will you end up with an array 0x1234 bigger than before.

```
    .org 0x1000
    lis  r16, 0x0002
    ori  r16,r16,0x4800
    lis  r17, 0x0001
    ori  r17,r17,0x8000
    mtctr r17
loop: lwz  r18,0(r16)
    addi r18,r18,0x1234
    stw  r18, 0(r16)
    addi r16,r16,4
    bdnz loop
    Nop
```

1) Set up ADDR
2) Send 0x10 to UART
3) Send 8 to INTC/IER
4) Send 3 to INTC/MER
5) Set up EVPR
6)
7) wrteei 1

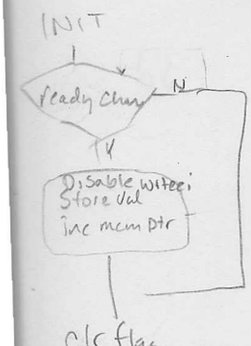
5. Interrupt question: This question has two parts. The first is setup/initialization, the second is steady state. Much of the configuration of the UARTLite from Xilinx is done in the initialization of the board, as opposed to the initialization of the processor. Nevertheless, we need to initialize the system for interrupt driven activity. In the space provided below, give instructions that will set up the system (UART, Processor, etc.) for interrupt driven applications for receiving characters. Assume that the system under consideration is configured as we have done it in the laboratory (UARTLite at 0x84000000; interrupt controller at 0x81800000; UARTLite interrupt bit in the interrupt controller is the '8' bit (IBM numbering this would be 28, normal numbering would put it at position/weighting 3)).

```
.set UARTLITE, 0x84000000
.set RxFIFO, 0x0
.set TxFIFO, 0x4
.set STATREG, 0x8
.set INTC, 0x81800000
```

```
lwz r1, RxFIFO
```

```
.org 0x2000
lis r8, 0x8400      Point to UART
lis r9, 0x8180      Point to INTC
                    Set interrupt bit
li r10, 0x10
stw r10, 0x0C(r8)   UART Control reg
li r11, 8           Set UART bit in
stw r11, 0x08(r9)   INTC
li r12, 3           Set two bits in
stw r12, 0x1C(r9)   MER reg in INTC
li r13, 0           Setup EVPR
```

For the second part of this question, give code for an interrupt service routine that will remove characters from the receive FIFO (on receive interrupt) and place the characters in an array of characters in memory. The array starts at address 0x00032100 and grows to higher addresses. Don't worry about an upper limit for the array. Make sure you check to be sure a character is available.




```
wrteei 0
li r7, 0x32100
lwz r8, r
andi. r9, r8, 1
bf 3, out
stw r1, 0(r7)
addi r7, r7, 4
```

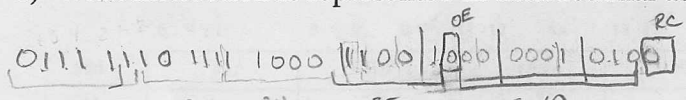
out:
Clear interrupt flag
wrteei 1
rfi

```
li r14, 0x0003      Setup r14 as pointer
ori r14, r14, 0x2100  with 0x32100
mtevpr r13          Set EVPR
wrteei 1            Allow interrupts
```

```
.org 0x500
lis r18, 0x8400     Point UART
lis r19, 0x8180     Point INTC
lwz r20, 8(r18)     Get Status Reg
andi. r21, r20, 1   look @ LSB
bt 2, go out        if not 1 go out
lwz r22, 0(r18)     data reg UART
stb r22, 0(r14)     Send to array
addi r14, r14, 1    bump pointer
go out: li r15, 8   set bit and
stw r15, 0x0C(r19)  write to IAR reg
rfi                 return from interrupt
```

0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7
1000 8
1001 9
1010 10 A
1011 11 B
1100 12 C
1101 13 D
1110 14 E
1111 15 F

6. Instruction coding question:
   a) What instruction is represented in hexadecimal as: `0x7EF8C814`?



   b) Give the coding for the branch-and-link instruction located at 0x1000 where the target of the instruction is the address 0x2220. Do not use absolute addressing mode, but rather the relative addressing mode.

0x3220

2220
-1000
1220
branch = 18 (I is displacement)
18
0100 1000 0000 0000 0001 0010 0010 0001
4 8 0 0 1 2 2 1

   c) How many different addresses can be targets of a conditional branch instruction?

8x4 = 32
24 different
bc
2^{14} bits to address

Table B-1 lists the PPC405 instruction set in alphabetical order by mnemonic.

Table B-1: **Instructions Sorted by Mnemonic**

	0 1 2 3 4 5	6 7 8 9	10	11 12 14	16 17 20	21	22 26	30	31
add	31	rD		rA	rB	OE	266		Rc
addc	31	rD		rA	rB	OE	10		Rc
adde	31	rD		rA	rB	OE	138		Rc
addi	14	rD		rA	SIMM				
addic	12	rD		rA	SIMM				
addic.	13	rD		rA	SIMM				
addis	15	rD		rA	SIMM				
addme	31	rD		rA	00000	OE	234		Rc
addze	31	rD		rA	00000	OE	202		Rc
and	31	rS		rA	rB		28		Rc
andc	31	rS		rA	rB		60		Rc
andi.	28	rS		rA	UIMM				
andis.	29	rS		rA	UIMM				
b	18	LI						AA	LK
bc	16	BO		BI	BD			AA	LK
bcctr	19	BO		BI	00000		528		LK
bclr	19	BO		BI	00000		16		LK
cmp	31	crfD	00	rA	rB		0		0
cmpi	11	crfD	00	rA	SIMM				
cmpl	31	crfD	00	rA	rB		32		0

Table 3-32: **Sign-Extension Instructions**

Mnemonic	Name	Operation	Operand Syntax
Extend-Sign Byte Instructions		rA[24:31] is loaded with (rS[24:31]). The remaining bits rA[0:23] are each loaded with a copy of (rS[24]).	
extsb	Extend Sign Byte	CR0 is *not* updated.	rA,rS
extsb.	Extend Sign Byte and Record	CR0 is updated to reflect the result.	
Extend-Sign Halfword Instructions		rA[16:31] is loaded with (rS[16:31]). The remaining bits rA[0:15] are each loaded with a copy of (rS[16]).	
extsh	Extend Sign Halfword	CR0 is *not* updated.	rA,rS
extsh.	Extend Sign Halfword and Record	CR0 is updated to reflect the result.	

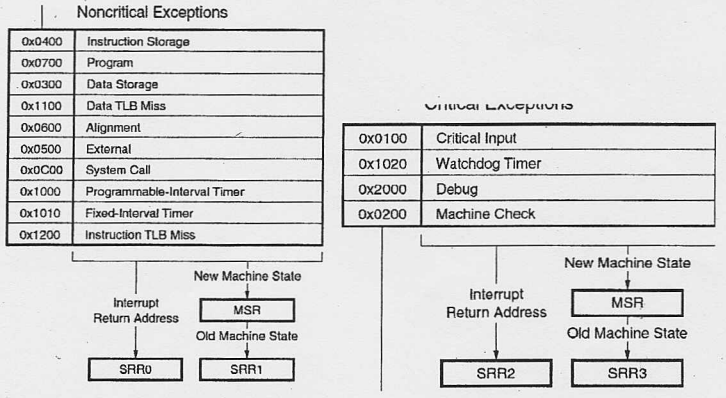


Table 4 shows all the XPS UART Lite registers and their addresses.

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

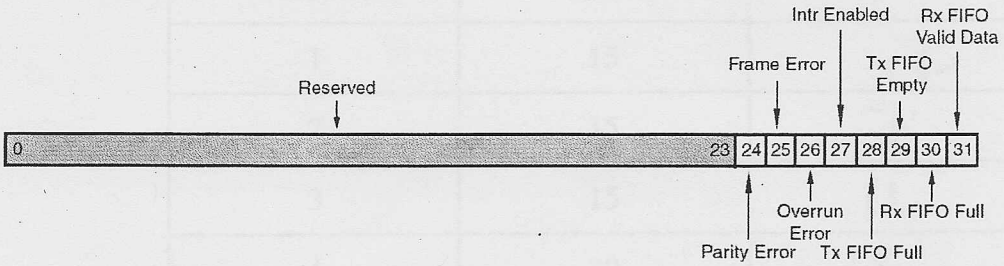


Figure 5: **UART Lite Status Register**

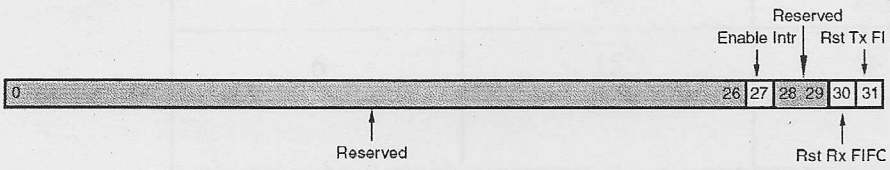


Figure 4: **UART Lite Control Register**

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