Name: ANTHONY MANCUSO

Problem	Possible	Score
1	15	12
2	15	12
3	20	13
4	15	15
5	20	18
6	15	10
Total	100	80

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. 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.

   Work (Add) Movement (lwz) Program control (bl) System control (wrteei)

   c) When a non-critical interrupt occurs, where is the current value of the machine state register stored?

   M.SR

   DUH

   d) True or false: 0x01234000 is a valid content for the EVPR.

   (multiple of 64k)

   e) When the record bit (Rc) is set in a work instruction (such as add.) where can you find an indication that the result is equal to zero?

   Condition register bit 2

   appropriate lsb of instruction

   f) Assume that register 27 contains 0x00021080. What address is accessed by the instruction 'lhz r26,r27(0x0022)'?

   0x 00021 0.2 00

   DUH

   10A2 not 1102

   DUH

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 of 8 bytes
4448 712A0001          andi. r10,r9,0x0001
444c 4182FFF8          bt 2,again
4450 81680000          lwz r11,0(r8)        # Receive FIFO offset of 0 bytes
4454 2C8B0033          cmpwi 1,r11,0x0033   # 0x33 is ascii '3'
4458 4086FFEC          bf 6,again
445c 4E800020          blr
```

r8 = 0x84000000
r9 = 0x00000001
r10 = 0x00000001
r11 > 0x11
check GT

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 = 0x84000000	r9 = 0x00000001
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 = 0x00000001	r11 = >0x11 0x55
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 = -2
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 =	r15 =
CR = 0x00000000	LR = 0x00000000	CR 0x40000000	LR 0x4408

42 (-1)

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; num bits in r14
1028c   7C841A14         add r4,r4,r3
10290   38840040         addi r4,r4,0x40
10294   80C40014         lwz r6,24(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)
```

r3 = 0x3000
r14 = 0x4
r4 = 0x30000000
r4 = 0x30003000
r4 = 0x30003040
r6 = 0x00000000
r7 = 0x00001213
r8 = 0x000000EF

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 =	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 =	r3 = 0x00003000
r4 = 0x44444444	r5 = 0x55555555	r4 = 0x30003040	r5 =
r6 = 0x66666666	r7 = 0x77777777	r6 = 0x00000038	r7 = 0x00001213 (-2)
r8 = 0x88888888	r9 = 0x99999999	r8 = 0x000000EF	r9 =
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 =	r11 =
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 =
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 = 4 (-2)	r15 =

Address																
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					Goes to r6					99						
00033070			AA	AA									BB	BBBB	BB	

(-1)

Corrections

r3: 0x00003000
r4: 0x30003040
r6: EF121314 15
r7: AABB
r8: EF

4. Coding 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 start at address 0x00030400 and fill each word location with its addresses. Do this for 10000 locations.

```
.set DATA 0x00030400
.org 0x3000
lis r1, Data@h
ori r1, r1, Data@l
li r2, 10000
mtctr r2
loop: stw r1 0(r1)
addi r1, r1, 4
bdnz loop
b .
```

5. Interrupt question: This question has two parts. The first is setup/initialization, the second is steady state. In the space provided below, give instructions that will set up the interrupt system to allow the Programmable Interval Timer system to cause an interrupt every 5 microseconds. (Internal system clock is 200 MHz.) In the initialization code set up the required registers appropriately; the interrupt table should be set up at its lowest legal value. After providing initialization code, provide also the Interrupt Service Routine needed for continued operation. Work of the Interrupt Service Routine is to put a non-zero value in the mailbox at 0x7000.

- set COUNT 1000
- set TCR 0b0001
- set MER 0x1C
- set FER 0x08
- set MBOX 0x7000

$\frac{5\,\mu s}{5\,ns} = 1000$ cycles

PIE
000.1 000
1 000/0000

- org 0x1000
  b Pitcode
  b .
- org 0x3000

li r1 COUNT
li r30 MBOX
li r2 0b0001
li r3 0b00010 ——— try diff pattern
mttsr r3
mttcr r2
mtpit r1
mtevpr 0 (zero)
wrteei 1

Pitcode:
mttsr r3
li r5, 0x0B
stw r5, 0(r30)
rfi

6. Instruction coding question:

a) What is the instruction that is represented by the bit pattern 0x7C044000.

0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

31 0 0 4 8

cmp 0 r4, r8

b) Give the coding for the instruction 'addeo. r9, r10, r11'.

0 1 1 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 1 1 0 1 0 1

adde r9 r11 r10 0E 138 RC

7 D 2 B 5 5 1 5

A D

-2

c) Assume that a branch-conditional instruction (bc) is located at 0x13238. What is the highest address that can be the target of that branch?

add 0x7FFFC

0x13238

too many F's

-3

0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0

0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0

9 3 2

b) 0 1 1 1 1 1 0 1 0 0 1

13238

7FFC

0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

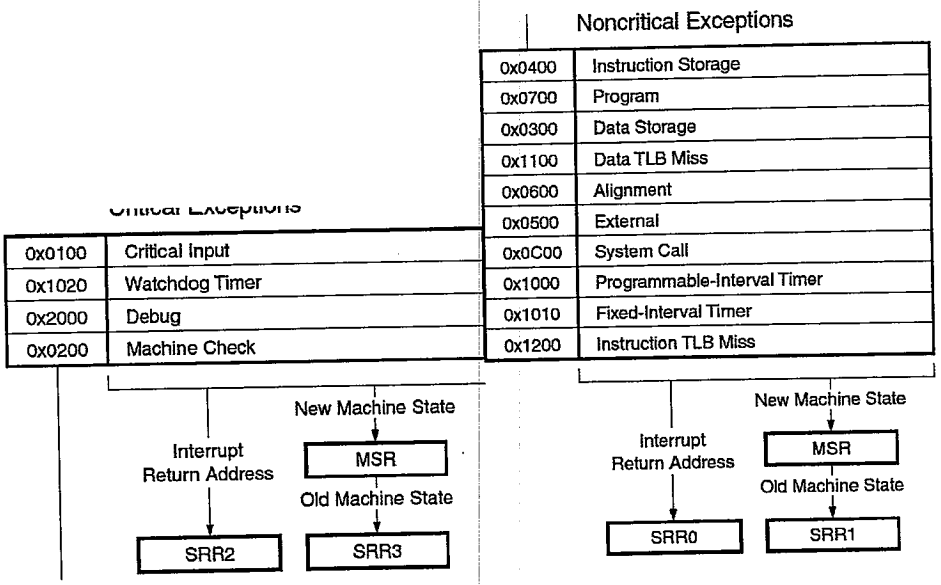
1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0

1 B 2 3 4

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

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

	0	6	9	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



Timer-Control Register

The timer-control register (TCR) is a 32-bit register used to control the PPC405 timer events. Figure 8-4 shows the format of the TCR. The fields in TCR are defined as shown in Table 8-3.

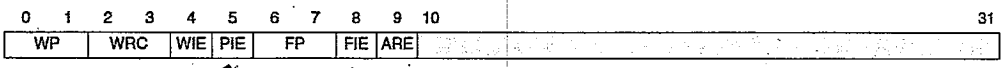


Figure 8-4: **Timer-Control Register (TCR)**

I need PIE!!

Timer-Status Register

The timer-status register (TSR) is a 32-bit register used to report status for the PPC405 timer events. Figure 8-5 shows the format of the TSR. The fields in TSR are defined as shown in Table 8-4.

Figure 8-5: **Timer-Status Register (TSR)**

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

Name: ANTHONY MANCUSO

Problem	Possible	Score
1	15	15
2	15	9
3	25	25
4	20	15
5	20	15
6	15	9
Total	110	88

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 *very* legibly!

1. Information representation. We gotta have some question about number representation. Consider number system(s) that contain 8 bits, with the radix point right in the middle. That is, there are four bits, then a radix point, then 4 more bits (xxxx.xxxx). 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.) Oh, and for the last line, provide the Value for the given bit pattern. Don't worry about turning the fraction version into a decimal version – just leave it as an integer-part, fraction-part answer.

1/2 1/4 1/8 1/16

Value	Unsigned binary pattern	Twos-complement pattern
Maximum	1111.1111	0111.1111
Minimum	0000.0000	1000.0000
$3\ ^{5}/_{16}$	0011.0101	0011.0101
$-5\ ^{9}/_{16}$	N/A	1010.0111
$6\ ^{3}/_{8}$	0110.0110	0110.0110
$-5\ ^{11}/_{16}$	N/A	10100101

0101.1001
−1
0101.1000
1010.0111

1010.0101
0101.1010
+1
0101.1011

2. General information question:
a) True or false, with a single instruction, R15 can be forced to the value 0xFFFFFFF0.

addi
Ri r2, -16

b) What is the highest address that is a legal location for the interrupt table?

0x F0000

0x FFFF0000

c) What is the order of bits involved in the Condition Register? That is, the Condition Register consists of eight groups of four bits. What is the order of the four bits that make up a group?

LT, GT, EQ, SO

d) Assume a conditional branch is located at address 0x00100000. 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?

0x 00100000
+ 0x 00007ffc
0x 00100ffc

e) An interrupt enabled UARTlite module will cause an interrupt when a character shows up in the input FIFO. What does a user do to remove this interrupt request?

Interrupt is removed by sending the UART interrupt bit to the IAR. However, the input FIFO must also be read to clear the UART assertion of the interrupt output

① Read RXFIFO
② Stroke IAR UART Interrupt bit

lw rx, 0(ry) # read RXFIFO
li r2, (AAAA) # Load IAR bits

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 your system is the first program to execute after reset, so there is no need to worry about any flags from an earlier program execution.

Addr Offset	Register	Bit Pattern
0x00	ISR	N/A
0x04	IPR	Read-only
0x08	IER	0x F
0x0C	IAR	N/A
0x1C	MER	0x 3

← Can use if MER has not been activated yet.

Now, in the space provided below, give instructions that will establish the bit patterns given above as well as to (a)set up the EVPR register (to **0x000F0000**) and (b) set up any enabling activity needed to allow interrupts in general. Assume that the interrupt controller has been located at address **0x80440000.**

```
.set  VECTOR, 0x000F0000
.org  0x3000
lis r1, VECTOR@h
~~ori r1,r1, VECTOR@l~~
lis r2, 0x8044   # INTC
mtevpr r1
li r3, 0xF
stw r3, 8(r2)
li r3, 0x3
stw r3, 0x1C(r2)
wrteei 1
```

4. ISR question: For system that utilizes the FIT to create a periodic time function, give code for an Interrupt Service Routine for the following situation. When the FIT module causes an interrupt, perform the 'normal' FIT activity, increment the value in R27 and send the value to the LEDs. The LED interface GPIO is located at **0x84480000**. Do not worry about register volatility, nor about LED GPIO initialization (i.e., this is a steady state question).

```
.org 0x100 (or 1010)
b FITCode

.org 0x3000
lis r1, 0x8448
(stw r0, 4(r1)) # N/A per instr.
li r2, 0x0500
mttcr r2 # Prep FIT bits
li r3, 0x8000
mttsr r3 (not both bits) # * Some PIT setting may also be applied but FIT will work if PIT is negated. (PIT is never asserted - mtpit not issued)
loop b loop

FITCode:
('normal' FIT activity...)
addi r27, r27, 1
stw r27, 0(r1)
mttcr r2
mttsr r3
rfi

PITCode: # To remove effect of other bits
rfi # Did not remember FIT bits...
# But probably not necessary
# because mtpit was not
# issued.
```

normal → mttcr r2 / mttsr r3

```
.org 0x1000
b PITCode

.org 0x4000
b fitcode

fitcode: lis r8, 0x0400
mttsr r8
addi R27, r27, 1
lis r28, 0x8448
stw r27, 0(r28)
rfi
```

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 **0x00FF0500** (and progresses to higher addresses). Create code that will calculate the sum of the first 800 values of this array. Place the sum value in the word location 0x00060500.

```
.set ARRAY, 0x00FF0550
.set RESULT, 0x00060500
.org 0x3000
    lis r1, ARRAY@h
    ori r1, r1, ARRAY@l
    lis r2, RESULT@h
    ori r2, r2, RESULT@l
    li r3, 800
    li r4, 0
    mtctr r3
loop:
    lwz r5, 0(r1)
    add r4, r4, r5
    addi r1, r1, 4
    stw r4, 0(r2)
    bdnz loop
end b end
```

6. Recursive subroutine call. Occasionally it is necessary to provide a programming solution that allows for what is called re-entrant code, or code that can call itself. This was demonstrated in class with recursive Fibonacci number generation and in the lab with recursive factorial generation. In the space provided below give a sequence of instructions that implement a recursive subroutine call. Be sure to include what is needed to correctly handle the return address appropriately.

```
.org   0x3000  0x4000
       li r1, 0x5000
       li r2, 0x5000
What   stmw r24 0(r1)
making bl SUB
this?
here   b here
```

```
.org  0x3500
SUB:  mflr r3
      stmw r24 0(r1)
      stmw r24 0(r2)
      addi r2, r2, -32
      bl SUB
      addi r2, r2, +32
      mtlr r3
      blr
```

(Scratchwork)

Recursion: into a routine that calls itself?

```
li r1, 0x5000
nop
nop
bl SUB
```

```
.org 0x3500
mflr r2
```

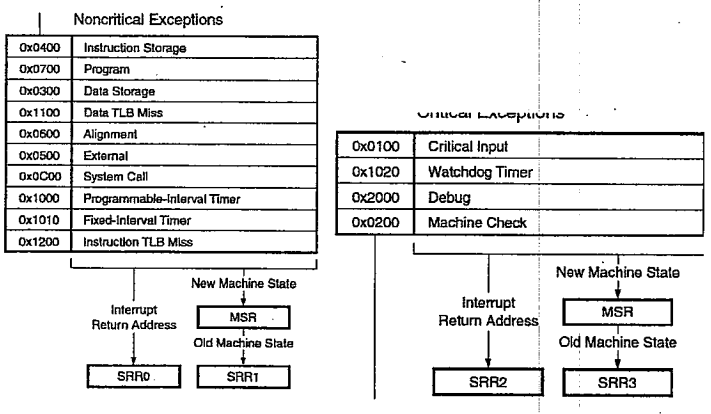


Table 4: **XPS INTC Registers and Base Address Offsets**

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

Timer-Control Register

The timer-control register (TCR) is a 32-bit register used to control the PPC405 timer events. Figure 8-4 shows the format of the TCR. The fields in TCR are defined as shown in Table 8-3.

Figure 8-4: **Timer-Control Register (TCR)**

Timer-Status Register

The timer-status register (TSR) is a 32-bit register used to report status for the PPC405 timer events. Figure 8-5 shows the format of the TSR. The fields in TSR are defined as shown in Table 8-4.

Figure 8-5: **Timer-Status Register (TSR)**

0400!