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. ½ ¼ ⅛ 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 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

(i r2, 0(r1) # read RXFIFO
(i r2, (AAA) # 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 0x1000
b PITcode

.org 0x1010
b FITCode

.org 0x3000
lis r1, 0x8448
li r0, 4(r1)   # N/A perimetr.
stw r0
li r2, 0x05C0
mttcr r2       # Prep FIT Bits
li r3, 0x0C00
mttsr r3       # Set FIT Bits
loop b loop    # * Some PIT setting may also be applied but FIT will work if PIT is negated. (PIT is never asserted - mtpit not issued)

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 necessa.
# because mtpit was not
# issued.

.org 0x4000
b fitcode

fitcode: li 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
       li r1, 0x4000
       li r2, 0x5000
       stmw r24 0(r1)
       bl SUB
here   b here
```

What making this?

```
.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
```

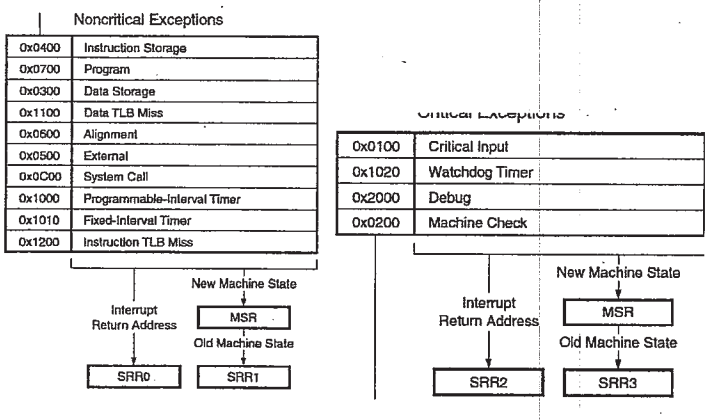


*Table 4:* **XPS INTC Registers and Base Address Offsets**

| 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.

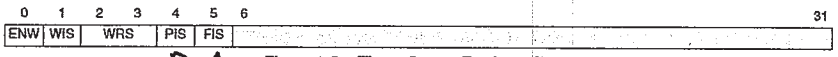


*Figure 8-5:* **Timer-Status Register (TSR)**

0400!