EECS373: MIDTERM 2 – Time 75 minutes

Rules: Open book. Write and sign the honor code on your blue book.

READ AND FOLLOW THESE INSTRUCTIONS.
Do not begin until you are told to do so.
You have 75 minutes; budget your time. The questions are not of equal weight; do not spend too much time on a question that is not worth many points.
Read through all of the questions before starting to work.
This exam is open book and notes, but you may not share reference materials with other students.
Please print your name and uniqname

Name: _____

Uniqname: _____

Sign the honor code below

I have neither given nor received aid on this exam, nor have I concealed any violation of the Honor Code.

Signature: _____

Answer the questions on the following pages.

Question	Points	Score
I	20	
II	25	
III	15	
IV	40	
Total	100	

I. (20 points) Short questions on RAM. Your answers should be brief. I have a 256M bit memory chip. It is organized as 64M x 4 bits.

1. (5 points) Assume a square memory. Describe the row decoder (i.e. is it a 12 to 1024, or what?)

2. (5 points) Describe the column decoder. How does the decoder account for the x4 bit data path?

3. (5 points) Can I build a 128M byte memory with this?

4. (5 points) How many of these chips do I need to build a 0.5 G byte memory. Sketch how they would be interconnected.

II. (25 points total) PowerPC ABI

(10 points) Rewrite and write the following code using the PowerPC ABI convention we used in the lab. The functions foo and boo are given to you and they must be debugged. This debugging includes anything from adding instructions, removing instructions, or modifying existing instructions. The C code for these functions is as follows:

```
void main(){
     int i;
     i = 2;
     i = foo(i);
}


int foo(int a){
     return(boo(coo(a), a));
}

int boo(int a, int b){
     while(a < b){
          a = a + a;
     }
     return(a);
}

int coo(int a){
     a = a – 1;
     return(a):
}


foo:
     stwu r1, -8(r1)
     stw  r3, 4(r1)
     bl   boo
     lwz  r3, 4(r1)
     bl   coo
     addi r1, r1, 8
     blr

boo:
     stwu r1, -4(r1)
while:
     cmpw r3, r4
     blt  done
     add  r4, r4, r4
     b    loop
```

```
done:
     blr
```

```
foo:
```

```
boo:
```

(10 points) Write the coo function using the PowerPC ABI convention we used in the lab

```
coo:
```

(5 points) What happens if main is executed as is?

III. (15 points total) A/D converters.

(5 points) Consider a four-bit analog-to-digital converter with an input range of 0-10V. After a conversion, the ADC returns a value of 9. Assuming an ideal converter, what is the possible voltage range of the analog signal?

(5 points) If your program assumes that the analog input is in the center of this range, what is the range of possible error in volts?

(5 points) If this is a flash-type ADC, how many voltage comparators does it contain?

IV. (40 points total) ISR Question

This question involves a system built upon the MPC823 you are familiar with from lab. Proceed accordingly. The part of the system you are to design involves the maintenance and monitoring of the water temperature of an aquarium. You will write an ISR to implement your solution. This question relies heavily on your lab section experience.

You must keep the temperature within the range of 60 to 70 degrees Fahrenheit. To facilitate this you are given a Temperature Control System (TCS) involving a heater and cooler separately controlled. The TCS will register three interrupts on IRQ1 of the MPC823. One to indicate passing the lower limit of the valid temperature range, one to indicate passing the upper limit, and one to indicate reaching 65 degrees (the middle of the range). The details of the TCS are hidden from you behind the following function calls:

void **TCS_ctl_cooler**(*bool* **on**)
// Takes a boolean (0 = false, else = true) argument to set state of cooler. If same as current state, no change

v*oid* **TCS_ctl_heater**(*bool* **on**)
// Takes a boolean (0 = false, else = true) argument to set state of heater. If same as current state, no change

void **TCS_clearints**
// Clears ALL interrupts currently asserted by the hardware of the TCS
// Due to semantics of this problem, you should never have more than
// one int from the TCS at a time.

int **tc_gettemp**
// Returns the temperature of the temperature control system

Note: You must use these, and only these, function calls to access the TCS.

Upon receiving a TCS interrupt you must determine the proper action to maintain the temperature range. For example, if you pass the lower limit, turn on the heater, and return from the interrupt. Soon, you will get an interrupt indicating 65 degrees, and you will need to turn the heater off (Hint: You have no way of knowing whether you were heating or cooling, but it should not matter. See function descriptions).

Note: You do not need to worry about fluctuations around the limit temperatures. That is, assume that you will get the interrupt upon passing a border and by time you return from that interrupt the temperature will have already risen or fallen back into the valid range.

Secondly, the RTC will have been initialized to interrupt every second on Level 0. Upon each interrupt, you will need to get the current temperature from the TCS, and display it.

Displaying involves simply writing an integer value to a memory-mapped register residing at *0x03700000***.**

Finally, you will have a manual override switch, which will stop the TCS from interrupting. This switch will assert its interrupt at **IRQ2**. IRQ2 will have been initialized to be an edge-triggered interrupt in the MPC823, so there is no need to clear the interrupt assertion in hardware. There will be no need to re-enable the interrupts.

Part A(20point) Write the ISR and handlers. This ISR must follow all ABI conventions. It is a polling ISR with priority: RTC handler, manual override switch handler, TCS interrupt handler. Do NOT write the handlers within the section titled **ISR:**, you should only (and you must) branch and link to them from here. The manual overide handler has been given to you as an example.

Note: Not commenting will result in a severe loss of points.

ISR: RTCHANDLER:

```
ANUALHANDLER:                                   TCSHANDLER:
mfspr   r3, IMMR
andis   r3, r3, 0xFFFF
lis     r4, 0x0800
stw     r4, SIPEND(r3)      #Clear INT2
lwz     r4, SIMASK(r3)
lis     r5, 0xDFFF
andi    r5, r5, 0xFFFF
and     r4, r4, r5
stw     r4, SIMASK(r3)      #disable IRQ1
blr
```

Part B(10 points) Rewrite the ISR to use auto-vectored interrupts (SIVEC).

Note: Not commenting will result in a severe loss of points.

Part C (10 points) Design the memory-mapped display register. It only needs to handle 32 bit writes. Assume the actual display is a macro with a single 32 bit bus input. It displays constantly the value on this bus (it converts from the binary value to decimal, but you don't need to worry about that). Remember, it resides at **0x037000**

Hint: This is very similar to your 7 segment display register from lab.