EECS 373 Exam 1

Fall 2002, Prof. Mark Brehob

Name: _____Key_____ uname: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

Page	Points
2	/20
3	/13
4	/19
5	/13
6	/18
7	/17
Total	**/100**

NOTES:

- Closed notes. White book and the ABI only!
- ***Don't spend too much time on any one problem.***
- You have 80 minutes for the exam.
- Multiple choice questions and certain other questions will be graded as right or wrong with no partial credit given.

_____

Section I – short answer – 65 points

1) Which of the following is true? **[3]**
   a) <u>The PCI bus has a turn-around cycle between address and data on a read but not on a write.</u>
   b) The PCI bus has a turn-around cycle between address and data on a write but not on a read.
   c) The PCI bus has a turn-around cycle between address and data on a read and on a write of greater than 32 bytes.
   d) The PCI bus has a turn-around cycle between address and data only on reads of greater than 32 bytes.
   e) None of the above are true

2) Explain why your answer to the above question (#1) is correct. (That is, why is a turn-around cycle required in that case but not others?) **[5]**

The PCI bus has a shared ADS/Data bus. On a read the ADS is sent by the initiator, and the data is sent by the target. As such, a turn-around cycle is needed to insure that the two devices are not driving the bus at the same time. For a write the initoator drives both ADS and data.

3) What is a split-transaction bus? Which of the three buses we studied is/are a split-transaction bus(es)? Explain your answer. **[4]**

It is a pipelined bus. That is, different stages of different transactions are being carried bout at the same time. The P6 bus is a split-transaction bus.

4) Say that you are required to expand the memory design in Lab 4 to 1K or 1024 bytes while maintaining the 32-bit data port. For this design you are to use 64 X 8bit ram chip instead of the 32 X 8bit ram used in lab 4.

   i) How many ram chips are required to implement this new memory? **[4]**
      (a) 1
      (b) 2
      (c) 4
      (d) 8
      (e) <u>none of the above</u>

(64 x 8 bits=64 bytes. 1024 bytes/64 bytes = 16)

   ii) Which <u>*additional*</u> address lines are you required to use in order to address this memory (that is, in addition to the ones needed in the lab 4 assignment)? **[4]**

1024 bytes need to be addressed. In lab we addressed 256 bytes. So we had 8 bits ($log_2 256$) now we need 10 bits. We need the lowest order bits, so ADS 22 and 23 need to be added.

_____

5) In the EABI discussed in class some of the registers are labeled as "volatile". Using the terms "caller save" and/or "callee save" define what it means for a register to be volatile. **[4]**

Volatile implies that the called function will not save the register. As such, it is up to the caller. So Volatile == <u>Caller save</u>

6) In lab 1 you used a macro (D2_4E) for the 2 to 4 decoder. Build a simple 2 to 4 decoder using only AND and NOT gates (you may use bubbles for NOT gates.) Label the inputs as **I[0:1]** and the outputs as **O[0:3]**. **[5]**

No one got this wrong, so I'll not bother with an answer ☺

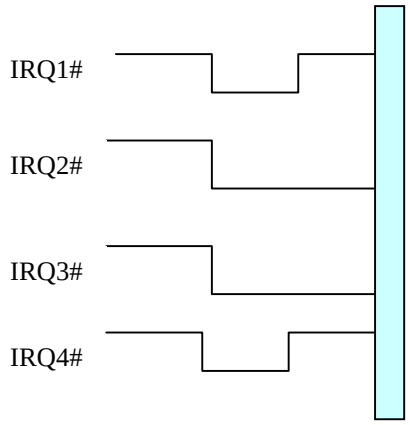
7) Which of the following code snippets will always result in the value 0xDEAD0555 being loaded into r4? **[4]**

```
I.   addis r4, r4, 0xDEAD0555@h
     ori r4, r4, 0xDEAD0555@l

II.  lis r4, 0xDEAD0555@h
     ori r4, r4, 0xDEAD0555@l

III. lis r4, 0xDEADBEEF@h
     addi r4, r4, 0xDEADBEEF@l
```

a. I only
b. II only
c. I and II only
d. III only
e. all of I, II, and III

_____

8) Say that SIEL has a value of 0x60800000 and SIMASK has a value of 0xDF3F0000. Say that the following IRQ's behaved as shown:



Assume that all the IRQs and LVL lines have been high since reset unless shown otherwise. Further, assume SIPEND has not been written to since reset.

What is the value, written as a 32-bit hex number is found in SIPEND? **[5]**
0x0A00000

What value, written as a decimal number, is found in SIVEC? **[5]**
16 if you read SIVEC as a byte. 0x1000 or 0x10000000 as word or ½ word (in hex)

9) Consider the PowerPC instruction addi. What is the smallest immediate value that can be used with that instruction (use a 2^X notation if possible)? **[4]**

-2^{15} ---- we also took "0" because smallest *could* refer to magnitude.

10) What is the primary advantage of using DMA rather than programmed I/O? **[5]**

That the CPU can do other things while the I/O device does DMA. With programmed I/O the processor has to be constantly involved.

11) Using PowerPC assembly write a program that does the same as the following C code-segment.

```
int a,b;
…… /* Other code happens here */
if(a>b)
        a++;
else
        b--;
…… /* Code continues */
```

Assume that a and b are 4-byte memory locations identified by labels of the same name. You may use any intermediate locations (registers or memory locations) you need. **[8]**

```
        lis r3, a@h
        ori r3, r3@l
        lis r4, b@h
        ori r4, b@l
        lwz r5, 0(r3)
        lwz r6, 0(r4)
        cmpd r6, r5
        bgt else
        addi r5, r5, 1
        stw r5, 0(r3)
        b done
else:   addi r6, r6, -1
        stw r6, 0(r4)
done:
```

12) When writing an ISR (interrupt service routine) which of the following is true? **[5]**

i) Before returning from the ISR, volatile registers must be restored.
ii) Before returning from the ISR, non-volatile registers must be restored.
iii) If you want to allow the ISR to itself be interrupted you need to change a bit in the MSR.
iv) You may not use the stack.

a) i and iii only
b) i, ii and iii only
c) ii and iii only
d) i, ii, and iv only
e) all of i, ii, iii, and iv

Section 2 – 35 points

1) Consider a 32-bit wide memory module connected to the MPC823 data bus starting at 0x0000 and ending at 0x0FFFF. The module has been interfaced with a control module using the MPC823 control and address lines to perform byte, half word and word accesses according to the rules set down in tables 13-2 and 13-3 of the White Book. This module will perform identically to the one you built in lab except the memory range is different.

   For the following instructions, determine the number of accesses, TSIZE, address bus contents and data bus contents. Space is provided for as many as three accesses whether you need them or not. **[18]**

   Assume the following
   Assume the following values are in memory and the registers at the *start of each* assembly comman

   Memory location 0x0200 = 01234567 and memory location 0x0204 = 0x89ABCDEF
   r3 = 0x0100 r4 = 0x01234567 r5 = 0xXXXXXXXX r6 = 0x0200
   X = unknown * = any value

Instruction	Access	TSIZE [0:1]	Address Bus Contents	Data Bus Contents
lbz r5, 1(r6)	1	01	0x0201	0x01234567 0x**23****
stb r4, 1(r3)	1	01	0x0101	0x6767XXXX 0x6767****
sth r4, 3(r3)	1	01	0x0103	0x4545XX45 0x4545**45
	2	01	0x0104	0x67XXXXXX 0x67******
sth r4, 0(r3)	1	10	0x0100	0x4567XXXX 0x4567****
lw r5, 1(r6)	1	01	0x0201	0x01234567 0x**23****
	2	10	0x0202	0x01234567 0x****4567
	3	01	0x0204	0x89ABCDEF 0x89******
stw r4, 3(r3)	1	01	0x0103	0x0101XX01 0x0101**01
	2	10	0x0104	0x2345XXXX 0x2345****
	3	01	0x0106	0x67XX67XX 0x67**67**

_____

2) The following program pushes an array of characters onto a stack and then pop's them off into a buffer. The stack grows from low memory to high. Fill in the missing code. **[13]**

```
        .data
        .align 0
        .equ   null,  0

array:  .byte null, 'h', 'e', 'l', 'l', 'o', null

        .align 2
buffer:  .skip 100

stack:  .skip 100
        .equ length,  6
        .global _start

        .text
_start:
        addi    r4,     r0,     0
        addi    r5,     r0,     0
        addi    r6,     r0,     0
        lis     r1,     array@h         # load r1 with address of array
        ori     r1,     r1,     array@l
        lis     r2,     stack@h         # load r2 with address of stack
        ori     r2,     r2,     stack@l
        addi    r1,     r1,     -1
        addi    r2,     r2,     -1

loop1:  lbzu    r3,     __1__(r1)       # load char from array into r3
        _stbu_  r3,     __1__(r2)       # push the first byte onto the stack
                                        #    AND update stack pointer
        addi    r5,     r5,     1       # r5++
        _cmpd_  r5,     length
        bne     loop1

        lis     r1,     buffer@h        # load r1 with address of buffer
        ori     r1,     r1,     buffer@l

loop2:  lbzux   r3,     r2,     r6      # pop byte off stack
        stbux   r3,     r1,     r4      # store byte into buffer
        addi    r5,     r5,     -1      # decrement counter
        addi    r6,     r0,     -1      # r6 = -1
        addi    r4,     r0,     1       # r4 = 1
        cmp     r5,     r0
        bne loop2

done:   b done
```

3) What are the contents of buffer after the above program reaches done? **[4]**

o,l,l,e,h,null

_____