UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
MIDTERM EXAMINATION, MARCH 2011

ECE243H1 S – COMPUTER ORGANIZATION
Exam Type: D
Duration: 2 Hours

Prof.s Anderson, Enright Jerger, and Steffan

This is a type D exam. You are allowed to use any book/notes and a non-programmable calculator as allowed by the University regulations.

Last Name (Print): _____

First Name: _____

Student Number: _____

Circle your Prof: Anderson, Enright-Jerger, Steffan

	Marks	Max. Marks
1		11
2		20
3		14
4		10
5		10
6		25
7		20
Total		110

Please:
State your assumptions. Show your work. Comment your code. **Use your time wisely.** The mark value of each question is roughly equivalent to how many minutes it should take to answer. If you think that assumptions must be made to answer a question, state them clearly. If there are multiple possibilities, comment that there are, explain why and then provide at least one possible answer and state the corresponding assumptions.

Part 1. [11] Short Answer

1. A section of memory is shown at right. Circle the most complete, correct answer:

2marks

a) This could be a section of a character string
b) This could be a section of a program
c) This could be a section of a debug monitor program on the NIOSII
d) All of the above
e) All but C

address	data
0x123456	0xef
0x123457	0x43
0x123458	0x35
0x123459	0x53
0x12345a	0x63
0x12345b	0x22
0x12345c	0x52
0x12345d	0x72

2 Briefly, give two differences between the interrupt mechanism used on the MC68000 microprocessors from Motorola, and the interrupt mechanism on the NIOSII.

2marks

vectored interrupt (not ISR at specific address)
prioritized interrupts / can vector based on interrupt priority
interrupts locked out by processor priority
vector addresses can be supplied by interrupting device
more?

3 Show two different ways to multiply by 8 in a NIOS processor.

2marks

with the mul instruction (using movi to move 8 into a register first)
Using a logical shift left instruction, shifting by 3 (log-base-2 of 8) eg., slli r9,r9,3
Using add 3 times (do add r8,r8,r8 three times)

4 Fill in the second column of this table: 1mark each

Value (16 bits)	**Show As:**
0xF844	Binary: 1111 1000 0100 0100
0xF844	Signed decimal: -1980
0xF844	Unsigned decimal: 63556
2's complement of 0xF844	Hexidecimal: 0x7bc
2's complement of 0x6F32	Hexidecimal: 0x90ce

Part 2. [20] Basic Instruction Semantics:

Assume the following initial state for registers, which is the same for all parts (a) to (j):

R8 = 0x33557799
R9 = 0x1234ABCD
R10 = 0xFFFFFFFC
R11 = 0xFFFFFFFA
R12 = 0x00110013

PC = 0x01100024
Value for Branch1 = 0x0104
Value for Branch2 = 0xFFF4

In the following list of instructions, determine what is the outcome of the instruction and write in hexadecimal using all eight digits (that is write 0x00000000 and not 0x0). **In all cases find the outcome of each instruction given that it executes with the initial state given above.** If an instruction is not legal (i.e. the instruction does not exist or the given use of operands is not allowed for the instruction), then write 'illegal'.

2marks each

a. AND R2, R10, R12 — R2 = **0x00110010**

b. ANDI R2, R9, 0xF — R2 = 0x0000000D

c. CMPLTU R2, R10, R11 — R2 = 0x00000000

d. BGT R11, R12, Branch1 — PC = 0x01100028

e. BNE R8, 15, Branch2 — PC = Illegal

f. ADDI R2, R8, 0x1FFF0 — R2 = Illegal

g. SLLI R2, R12, 6 — R2 = 0x044004C0

h. ADDI R2, R0, 0x9876 — R2 = 0xFFFF9876

i. MOVIA R2, 0x4 — R2 = 0x00000004

j. MOVI R2, 0x7000 — R2 = 0x00007000

Part 3. [14] Memory Operations: Assume the following initial state:

```
R8 =  0x11223344
R9 =  0xCCDDEEFF
R10 = 0x55667788
```

What is final value of R2 **in hexadecimal, using all eight digits** (that is write 0x00000000 and not 0x0), after the execution of each sequence of instructions, for both little- and big-endian machines? If any access in the sequence is misaligned, write 'misaligned' instead of an answer.

	Little-Endian	Big-Endian
`STW R9, 0(R8)` `LDW R2, 0(R8)`	0xCCDDEEFF	0xCCDDEEFF
`STH R9,0(R8)` `LDB R2,0(R8)`	0xFFFFFFFF	0xFFFFFFEE
`STW R10,0(R8)` `LDH R2,2(R8)`	0x00005566	0x00007788
`STB R9, 3(R8)` `LDBU R2, 3(R8)`	0x000000FF	0x000000FF
`STW R10,0(R8)` `LDB R2,2(R8)`	0x00000066	0x00000077
`STH R9, 3(R8)` `LDB R2, 3(R8)`	Misaligned	Misaligned
`STH R10,0(R8)` `LDB R2,1(R8)`	0x00000077	0xFFFFFF88

1mark each

Part 4. [10] Data structures:

Given the following:

```
Typedef struct list {
        Short a;
        Short b;
        Short c;
} item_t;

item_t array[15];
```

For the following assembly, which C code does it implement? Assume r9 holds i.

ANSWER: __D___

```
movia r8, array
add r10,r9,r9
add r11,r10,r10
add r11,r11,r10
add r11,r11,r8
ldh r12,-2(r11)
sth r12,2(r11)
```

a)	array[2*i].a = array[2*i].b	0 marks
b)	array[i-1].b = array[i].a	0 marks
c)	array[i+1].b = array[i].c	5 marks
d)	array[i].b = array[i-1].c	10 marks
e)	array[i].b = array[i].c	5 marks
f)	array[i].b = array[i-1].b	5 marks

Part 5. [10] C and Assembly

Which NIOS assembly code matches the following C code**?** **ANSWER: __C___**

```
while (i > 0) {
        count += my_func(i);
        i--;
}
```

a)
```
LOOP:   bge r16, r0, DONE
        mov r4, r16
        call my_func
        add r17, r17, r5
        addi r16, r16, -1
        br LOOP
DONE:  …
```

b)
```
LOOP:   ble r16, r0, DONE
        call my_func
        add r17, r17, r4
        addi r16, r16, 0xffff
        br LOOP
DONE:  …
```

c)
```
LOOP:   ble r16, r0, DONE
        mov r4, r16
        call my_func
        add r17, r17, r2
        addi r16, r16, -1
        br LOOP
DONE:  …
```

d)
```
LOOP:   blt r16, r0, DONE
        mov r4, r16
        call my_func
        subi r16, r16, 1
        br LOOP
DONE:  …
```

Part 6. [25] Interrupts: Assume a harddrive called mydrive, capable of both byte input and output, polling and interrupts; although being a hard-drive it is very slow relative to the CPU. The following spec defines the device:

Device: mydrive
Input/Output: both
Address base: `0XFACE00`

Address map:

Address	R/W	Description
base	R/W	**Data Register** 32-bits wide **bits 7:0 - data value** *one byte, can write-to or read-from*
base+4	R/W	**Control Register** 32-bits wide **bit 5(*) - write interrupt enable** *set to one to have the device request interrupts when it is ready for writing to* **bit 4 - read interrupt enable** *set to one to have the device request interrupts when it is ready for reading from* **bit 3 - write ready bit** *is a one when data register is ready for writing to* **bit 2 - read ready bit** *is a one when data register is ready for reading from* **bit 1 – write interrupt pending** *is a one when the device is requesting an interrupt because it is ready for writing to* **bit 0 – read interrupt pending** *is a one when the device is requesting an interrupt because it is ready for reading from*

(*) Bits are numbered starting at 0, which is the least significant bit

The device is connected to IRQ7. If an interrupt for read or write is pending it is acknowledged once the data register is read or written respectively, i.e., the device lowers the pending interrupt.

The device is guaranteed to work as long as registers are accessed as words. No guarantees are made if byte or half-word accesses are used instead.

a) [10] Fill in the main routine below to configure mydrive for an interrupt on ready to **write,** and to enable the interrupt so that it can occur as soon as the processor reaches label PSIT. Assume that the code after PSIT does not change the interrupt configuration of the processor or the device, but of course, it can read and write any registers and memory as allowed by the programming conventions for NIOS II.

```
      .section .text
      .global main
main:
     # your code goes here

     movi r8, 0x80 # enable IRQ7
     wrctl ctl3, r8

     movi r8, 0x20 # enable write interrupts on the device
     movia r9, 0xface00
     stwio r8, 4(r9)

     movi r8, 0x1 # enable interrupts on the CPU
     wrctl ctl0, r8

PSIT:      # Unknown code follows at this point
     ....
```

3marks

3marks

3marks

1mark for enabling pie bit last

b) [15] Assuming your answer in (a), the interrupt handler below will be invoked whenever mydrive is ready to be written to. Fill in the interrupt handler below to call the function `get_next_value()`, which has no input parameters, and returns the next value that you are to write to mydrive. After writing the value to mydrive, poll mydrive until ready to be read from, read a byte (an error code), and if the byte is non-zero then call the function `error()`(which does not return anything). Write the instructions for each part of the handler under the associated comment. Note that the code for `get_next_value()` and `error()` are assumed to be written for you, they use only callee-save registers, but are otherwise not shown.

```
      .section .exceptions,"ax"
IHANDLER:
# verify that mydrive raised the interrupt, return if not
     Rdctl ctl4,et
     andi et,0x80   # check bit7
     bne et,r0,EXIT_IHANDLER

# prologue/initialization for the handler (if necessary)
     # must save all registers that are used (except et)
     Subi sp,sp,4
     Stw r2,0(sp)

#write the data register
     movia et, 0xFACE00
     call get_next_value
     stwio r2, 0(et)

# poll mydrive until it is ready to read
POLL: ldwio r2,4(et)
     Andi r2,r2,0b100  # check bit2 (read ready)
     Beq r2,r0,POLL

# read and process the error code from mydrive
     Ldwio r2,0(et)
     Andi r2,r2,0xff  # isolate the data value
     Beq r2,r0,RESTORE
     call error          # error code non-zero

# epilogue
RESTORE:
     Ldw r2,0(sp) # restore registers
     Addi sp,sp,4

# return from the interrupt handler
     EXIT_IHANDLER:
     Subi ea,ea,4
     eret
```

3marks for this and "return" code below

3marks for this and epilogue below

3marks

3marks

3marks

Part 7. [20] Register Conventions / Stack Use

The assembly programmer is implementing the C-like pseudocode below. The part of the pseudocode in larger, bold font is implemented as shown in the larger bold font on the next page. Fill in the code to complete the implementation. You must use / can assume the register conventions for the call, return, etc. Note that registers are used in place of local variables **minvalptr, aptr and MinSoFar**. Variable **i**, in the "for" loop, should be implemented with r16.

C-Like Code

```
/* function, returns pointer to minimum
signed byte value in array */
*char getminval(*char ArrayPtr, unsigned int
ArraySize)
{
   char *minvalptr, *aptr;
   int i;
   char MinSoFar;

   aptr = minvalptr = ArrayPtr;
   MinSoFar = 0x7F; // largest positive number
   for(i=0;i<ArraySize,i++)
   {
      if (MinSoFar > *aptr)
       {
          minvalptr = aptr;
          MinSoFar = *aptr;
       }
      aptr++;
   }
   return(minvalptr);
}
```

Use this area for notes, rough drafts

Students should realize from the code given:

r8 is aptr
r9 is MinSoFar
r11 is minvalptr

r4 ArrayPtr
r5 ArraySize

Assembly Code COPY1 (only one copy will be graded, cross out the other copy)

[note: must have correct mapping of vars to regs, -1 for each wrong]

getminval: **# [4marks]**
put code here that deals with entry into a subroutine
subi sp,sp,4
stw r16,0(sp) #must save callee save register used later

[5marks]
#put code here for " aptr = minvalptr = ArrayPtr;" and " MinSoFar = 0x7F;"
mov r8,r4 #aptr = ArrayPtr
mov r11,r4 #minvalptr = ArrayPtr
movi r9,0x7f #MinSoFar = 0x7f

[3marks]
#put code here for start of "for" loop. **Use r16 for the variable "i"**
mov r16, r0
ForLoop: #this name is up to the students
bge r16,r5,Continue

ldb r12,0(r8)
bge r12,r9,NoChange
mov r11,r8
mov r9,r12
NoChange:
addi r8,r8,1

put code here for end of "for" loop [3marks]
addi r16,r16,1
br ForLoop
Continue:

#put code here to properly exit the subroutine [5marks]
ldw r16,0(sp) #pop saved register value
addi sp,sp,4
mov r2,r9
ret

Assembly Code

Assembly Code COPY2 (only one copy will be graded, cross out the other copy)

getminval:

put code here that deals with entry into a subroutine

#put code here for " aptr = minvalptr = ArrayPtr;" and " MinSoFar = 0x7F;"

#put code here for start of "for" loop. **Use r16 for the variable "i"**

```
   ldb r12,0(r8)
   bge r12,r9,NoChange
   mov r11,r8
   mov r9,r12
NoChange:
   Addi r8,r8,1
```

put code here for end of "for" loop

#put code here to properly exit the subroutine