Student # (use if pages get separated) _____

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
MIDTERM EXAMINATION, MARCH 15, 2012

Second Year

ECE243H1 S – COMPUTER ORGANIZATION

Examiners – Phil Anderson, Natalie Enright Jerger, Andreas Moshovos

Instructions

This is a type D exam. You are allowed to use any printed/hand-written material including your course notes. The use of calculators is not permitted.

Last Name (Print Clearly): _____

First Name: _____

Student Number: _____

<<CIRCLE THE NAME OF YOUR INSTRUCTOR IN THE HEADING>>

General Instructions:
State your assumptions. Show your work. Comment your code. Solutions that are judged significantly inefficient will lose some marks. The exam is printed on two sides of the page. The last pages are blank and can be used for answers or calculations.
Make your answers clear.

There are **9** questions and a total of **117** marks. There are **7** pieces of paper in the exam, this one included, printed both sides. The page numbering is **1-14**.

Student # (use if pages get separated) _____

Question 1 – Basic Understanding of Assembly [16 Marks]

[16] Basic Instruction Semantics: Assume the following initial state for registers:

```
R8 =  0x01020304
R9 =  0xFFFF8000
R10 = 0x12345678
R11 = 0xABCD1234
R12 = 0x0000FFFE
```

For the following list of instructions, determine what is resulting value in R2 in **hexadecimal, using all eight digits** (that is write 0x00000000 and not 0x0). 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'. Assume each instruction is independent and starts with the initial state given above.

a. `ADDI R2, R10, -5` R2 = _____0x12345673_____

b. `ADDI R2, R0, 0xFFFF` R2 = _____0xFFFFFFFF_____

c. `AND R2, R10, R12` R2 = _____0x00005678_____

d. `MOVIA R2, 0x12340010` R2 = _____0x12340010_____

e. `XOR R2, R8, R9` R2 = _____0xFEFD8304_____

f. `SLLI R2, R11, 2` R2 = _____0xAF3448D0_____

g. `CMPGTU R2, R11, R8` R2 = _____0x00000001_____

h. `CMPGTI R2, R11, -1` R2 = _____0x00000000_____

-2 for any that is wrong

Question 2 – Little vs. Big Endian [14 Marks]

Assume that r8=0x00001004, r9=0x12345678 and memory contains these values:

Address	+0	+1	+2	+3
0x1000	0xA1	0xFE	0x80	0xCD
0x1004	0x19	0x32	0x89	0xFE

What will be the value returned by each of the following loads for little and big-endian systems? Write your answer in hexadecimal using all 8 digits. If an operation is illegal, please indicate with "illegal". Assume that memory is as shown above before executing each sequence of instructions (Note the final row is a sequence of 2 instructions, all others are 1 instruction).

	Little-Endian	Big-Endian
`LDW R2, 0(R8)`	0xFE893219	0x193289FE
`LDB R2, -1(R8)`	0xFFFFFFCD	0xFFFFFFCD
`LDBU R2, -2(R8)`	0x00000080	0x00000080
`LDH R2,2(R8)`	0xFFFFFE89	0xFFFF89FE
`LDB R2,-4(R8)`	0xFFFFFFA1	0xFFFFFFA1
`LDH R2,1(R8)`	Misaligned	Misaligned
`STH R9,-2(R8)` `LDW R2,-4(R8)`	0x5678FEA1	0xA1FE5678

-1 for any that is wrong
No part marks

Question 3 – Subroutines [12 Marks]

Mr. Andares Vosshomo who you are considering to hire for your startup has written the following code where mysub() calls F1() which then calls G1(). Mysub() is called from main.

```
Mysub:
M1:       movi r16, 1
M2:       call F1
M3:       add r2, r2, r16
M4:       ret

F1:       call G1
F2:       add r2, r2, r0
F3:       ret

G1:       movi r2, 7
G2:       ret
```

(i) As written, what is the sequence of instructions that will be executed? Complete the sequence below:

M1, M2, … F1, G1, G2, F2, F3, F2, F3, F2, F3 …

6 marks: -1 for any wrong

(ii) Is the code correct? Did your star candidate forget something? If so, show the changes.

4 Marks

Save/restore ra in mySub

Save/restore ra in F1

Save/restore r16 across the call to F1

-1 for each of the above, -1 if no reasonable attempt whatsoever

(iii) What will be the value of r2 when M4 gets executed after you implement your change?

R2 = 8

2 Marks: -2 if wrong

Student # (use if pages get separated) _____

Question 4 – Programming Conventions and Hardware Enforcement [8 Marks]

The following registers are given specific designations according to the chart you were given. Indicate whether the designation is by programming convention or by hardware enforcement.

Register	Convention (check for yes)	Hardware Enforcement? (check for yes)
r0 - zero		x
r2 – return value	X	
r4 – register argument	X	
r8 – caller-saved register	X	
r16 – callee-saved register	X	
r27 – sp, stack pointer	x	
r29 – ea, exception return address	Ok	X
r31 – ra, return address	ok	X

-1 for any wrong
Ok if ea and ra marked as convention

Question 5 -- Coding [10 Marks]

For the following "C" constructs, write equivalent assembly code. Assume the commented areas (like "//body") represent other C code. Include these commented areas in your code as well in the proper place. All variables are unsigned integers. You can assume that `i` is in `r8`, `thisVal` in `r8`, and `thatVal` in `r9`. Function **foo** does not overwrite any caller-save registers.

a.
```
for((a)i=5; (b)i>0; (c)i--, (d)thatVal++){
// body
}
```

```
movi r8, 5
ble r8, r0, Q # Optional because we know r8 = 5
L: # body
subi r8, r8, 1
addi r9, r9, 1
bgt r8, r0, L
Q: …
```

5 marks
-1 for any of the (a)-(d) that is missing or not implemented correctly
-1 for any edge of the control flow that is not implemented correctly
-1 for referencing names instead of registers

b.
```
if((a)thisVal > (b)foo(thatVal))
{ // ifpart (c)
} else
{ //elsepart (d)
}
```

```
#Assume ra is already saved
mov r4, r9
call foo
ble r8, r2, EP # if thisVal <= ret_value, goto elsepart
IP: # ifpart
br Q
EP: #elsepart
Q: …
```

-1 for not passing the argument correctly
-1 for not using r2 as the second register to compare
-1 for not calling foo
-1 for not comparing thisVal
-1 for not getting the condition right
-1 for the THEN edge
-1 for the ELSE edge
-1 for problems with stack

Question 6 – Assembly Programming [20 Marks]

The following assembly code is supposed to implement a subroutine that counts the number of odd values in an array and returns the count to main. Please find and correct 10 errors in the code (there may be more). Use the corresponding numbers to indicate instructions you are correcting. If you need to add additional instructions indicate the numbers of the instructions they will be inserted between. The next page is blank for your use.

```
1.    .section .data
2.              .align 1        #.align 2
3.    arr_size:       .word 10
4.    myarray:        .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

5.    .section .text
6.    .global main
7.    main:
8.              # some code
9.              call count_odd
10.             # other code here

11.   count_odd:
12.             subi sp, sp, 4 # subi sp, sp 16
13.             stw r16, 0(sp)
14.             stw r17, 4(sp) #need to save r18,r19
15.             movia r16, arr_size
16.             ldw r16, 0(r16)
17.             mov r17, 0x0    # movi r17,0x0 or mov r17, r0
18.             movi r18, myarray  # movia, init r2
19.   LOOP:     bge r17, r16, DONE
20.             ldb r19, 0(r18)      # ldw r9, 0(r18)
21.             andi r19, r19, 0x2   # andi r19,r19,0x1
22.             beq r19, 0x0, NEXT   # beq r19, r0, NEXT
23.             addi r2, r2, 1
24.   NEXT:     addi r18, r18, 4
25.             addi r17, r17, 1
26.             br LOOP
27.   DONE:     ldw r16, 4(sp) # ldw r16, 0(sp)
28.             ldw r17, 0(sp) # ldw r17, 4(sp), restore
29.             add sp, sp, 16       # addi sp, sp, 16
30.             ret
```

Start from 0 and keep adding points
+2 for any error appropriately identified
-1 for any errors that are introduced
lines with multiple errors: count each error separately, e.g., add sp, sp, 16 has two errors (addi) and the (16)

Question 7 – Arrays and aggregate data structures [15 Marks]

Here is an array of structures, defined in C. Assume that chars are 1 byte, shorts are 2 bytes, and ints are 4 bytes:

```
struct element
{
    char a;
    unsigned int c;
    unsigned short b;
} array[16];
```

(A) How many bytes are needed to store each element (i.e., how many bytes are there between two consecutive elements in the array?). Explain your answer.

12 bytes - char padded to 4B to align int, short padded to 4B to align each array element

2 marks: either get it or not

(B) Fill in the .align and .space directives in the code below to correctly allocate, and initialize to zero, all the space for the array:

```
            .section .data
            .align __2___
array:      .space __192___
```

2 marks one for each number asked

(C) Implement the following C code in assembly. Assume that the label 'array' points to the base address of the array, register r10 holds 'i', and r11 should hold the 32-bit zero-extended value of 'result' when your code finishes executing.

```
unsigned int result = array[i].b;
```

9 marks

addr = base + i x sizeof + offset to b

-1 for not using base, -1 for not using i, -1 for not using sizeof, -1 for not multiplying i x sizeof, -1 for not using offset, -1 for not adding offset

-1 for not loading from addr

(D) Reduce the memory footprint of the original array of structures by reordering the members a, b, and c. How many bytes is each array element now?

2 marks: either get it or not

```
struct element
{
    unsigned int c;_________
    unsigned short b;_________
    char a;_________
} array[16];
```

New size: __8___ bytes per element

Question 8 -- Interrupts [12 Marks]

In the following code, the interrupt handler increments `cnt` whereas the main program decrements it:

```
            .data
cnt:        .word 9
            .section .exceptions, "ax"
iHdlr:
I1:         ldw       r9, 0(r8)
I2:         addi      r9, r9, 1
I3:         stw       r9, 0(r8)
            # interrupt is acknowledged here
            subi ea, ea, 4
            eret
            .text
main:       # interrupts are initially disabled
            movia r8, cnt
            # interrupts are enabled here
wait:
W1:         ldw r11, 0(r8)
W2:         addi r11, r11, -1
W3:         stw r11, 0(r8)
W4:         br wait
```

If we could observe the sequence of writes to `cnt` done by the stores at I3 and W3, which of the following value sequences are possible? If so, explain why by showing the sequence of instructions that get executed (e.g., W1, W2, W3, W4, W1, W2, W3, …, etc.; if not, give a brief explanation as to why not).

3 general marks: do they seem to understand that the interrupt may execute at arbitrary points: I expect most students to get this if not all

1. `10, 11, 12, 11`

   Yes. I1-3, I1-3, I1-3, W1-3

   3 marks: -1 for any instruction that is wrong
   Min 1 if they get the yes/no answer right

2. `10, 9, 10, 8, 9`

   Yes. I1-3 (10), W1-4 (9), W1, I1-3(10), W2-3(8), H1-3(9)

   3 marks: -1 for any instruction that is wrong
   Min 1 if they get the yes/no answer right

3. 10, 12, 13, 12

No. Can't get from 10 to 12 without going through 11, because ISR can't be interrupted. (10, 12 can't even appear as a sub-sequence.)

3 marks: -1 for any instruction that is wrong
Min 1 if they get the yes/no answer right

Question 9 – Synthesizing operations on wider datatypes [10 Marks]

We want to implement 64-bit unsigned integer addition. Assume we are presenting 64-bit integers using a 32-bit value pair `(lo, hi)`, where `lo` contains the lower and `hi` the upper 32-bits of the 64-bit number respectively. Write a NIOS II instruction sequence that adds the 64-bit number stored in `(r8, r9)` with the 64-bit number stored in `(r10, r11)`. The resulting 64-bit number should be stored in `(r12, r13)`.

```
add r12, r10, r8     # add LSWs
cmpltu r13, r12, r8 # check LSWs for carry, store result (0/1) in MSW
add r13, r13, r11     # add one operand to MSW
add r13, r13, r19    # add other operand to MSW
```

2 marks: they do a.lo + b.lo
2 marks: they do a.hi + b.hi
4 marks: they correctly detect when the is a carry that needs to be carried from the lo addition to the hi
Incomplete conditions for carry: 2/4
2 marks: they appropriately adjust the hi part
-2 if treat r8, etc as addresses
-1 if do not use unsigned compare
-1 if switch lo and hi

Student # (use if pages get separated) _____

Question 1	**16**
Question 2	**14**
Question 3	**12**
Question 4	**8**
Question 5	**10**
Question 6	**20**
Question 7	**15**
Question 8	**12**
Question 9	**10**
Total	**117**