Print:

First Name:. . . . . Last Name:. . . . .

University of Toronto
Faculty of Applied Science and Engineering

Midterm Examination – March 4, 2014

ECE243 – Computer Organization

Examiners – **Phil Anderson, Natalie Enright Jerger, Glenn Gulak, Hamid Timorabadi**

Circle (above) the instructor whose section you will pick your exam up in.

1. There are 9 questions and **14** pages. Do **all** questions. The total number of marks is 100. The duration of the test is 120 minutes.
2. **ALL WORK IS TO BE DONE ON THESE SHEETS!** Use the back of the pages if you need more space. You may also use the two blank sheets included at the end of the exam. Be sure to indicate clearly if your work continues elsewhere.
3. Please put your final solution in the box if one is provided.
4. **Clear and concise** answers will be considered more favourably than ones that ramble. Do not fill space just because it exists!
5. The exam is open book/open notes.
6. Calculators are not permitted.
7. Always give some explanations or reasoning for how you arrived at your solutions to help the marker understand your thinking.
8. To get **full marks** you must **comment** your code.
9. State your assumptions. Show your work. Use your time wisely as not all questions will require the same amount of time. 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.
10. Only exams written in pen can be considered for remarking.

Print:

Student Number:. . . . .

This page is for grading purposes only. The marks breakdown is given for each question.

1 [15]	
2 [12]	
3 [20]	
4 [5]	
5 [10]	
6 [10]	
7 [8]	
8 [10]	
9 [10]	
Total [100]	

[15 marks] 1. Circle the correct answer:

(a) Convert the following decimal numbers into signed integer 32-bit representation (2's complement for negative numbers).

i. -60

```
(a) 1000 0000 0000 0000 0000 0000 0011 1100

(b) 1111 1111 1111 1111 1111 1111 1100 0011

(c) 1111 1111 1111 1111 1111 1111 1100 0100

(d) 1000 0000 0000 0000 0000 0000 0011 1011

(e) none of these
```

c

ii. 72

```
(a) 1000 0000 0000 0000 0000 0000 0101 1001

(b) 0000 0000 0000 0000 0000 0000 0100 1000

(c) 0000 0000 0000 0000 0000 0000 0100 0010

(d) 0000 0000 0000 0000 0000 0000 1000 0001

(e) none of these
```

b

(b) Convert the following binary number (unsigned 32-bit integers) into decimal:

i. `0000 0000 0000 0000 0000 0000 0100 0111`

```
(a) 107

(b) 142

(c) 35

(d) 71

(e) none of these
```

d

(c) Convert the following binary number (32-bit floating point IEEE-754) into decimal:

i. `0100 0001 1000 0000 0000 0000 0000 0000`

```
(a) 16.0

(b) 32.0

(c) 64.0

(d) 128.0

(e) none of these
```

a

(d) Convert the following decimal number into binary (32-bit floating point IEEE-754):

i. -12.0

```
(a) 0100 0001 0100 0000 0000 0000 0000 0000

(b) 1100 0001 0100 0000 0000 0000 0000 0000

(c) 1100 0001 0000 0000 0000 0000 0000 0000

(d) 1100 0000 1100 0000 0000 0000 0000 0000

(e) none of these
```

b

[12 marks] 2. The contents of a few NIOS-II registers and the content of the memory in hexadecimal are shown in the following two Figures. Write the values (in hexadecimal) of the registers after each of the following instructions are executed in space provided (each part is independent of the others; in other words, when answering part (b) do not assume that part (a) has executed).

Register	bit 31			bit 0
`r8`	00	00	00	68
`r9`	FF	FF	FF	FF
`r10`	00	00	00	00
`r11`	00	00	00	00
`r12`	FF	FF	FF	FF

Address	Content (bit 7 … bit 0)
0x0000 00064	01
0x0000 00065	FF
0x0000 00066	03
0x0000 00067	04
0x0000 00068	F1
0x0000 00069	EE
0x0000 0006a	03
0x0000 0006b	05
0x0000 0006c	09
0x0000 0006d	AA
0x0000 0006e	F0

(a) `ldh r9, 4(r8)`

Register	31			0
`r8`	00	00	00	68
`r9`	FF	FF	AA	09

(b) `ldb r10, 5(r8)`

Register	31			0
`r10`	FF	FF	FF	AA

(c) `ldb r12, 0(r8)`
`addi r12, r12, 16`

Register	31			0
`r12`	00	00	00	01

(d) `addi r8, r8, 8`
`ldw r11, -12(r8)`

Register	31			0
`r8`	00	00	00	70
`r11`	04	03	FF	01

[20 marks] 3. You are given the following code where the data section starts at 0x1000 and the text section starts at 0x2000:

```
        .data
        .equ SMALLNUM, 10
MyBytes:
        .byte 10,15,18,20
MyString:
        .string "9876543210"       #note: ASCII of 0 is 0x30
        .equ BIGNUM, 0x50000
        .align 2
MyVar:  .word 6
        .text
_start:
        movi r16,SMALLNUM
        movia r15,MyString
        mov r17,r0
        ldw r18,0(r15)
LOOP:
        subi r16,r16,1
        addi r15,r15,1
        ldbu r18,0(r15)
        bgtu r17,r18,NOCHNG
        mov r17,r18
NOCHNG:
        bne r16,r0,LOOP
ENDLOOP:
        br ENDLOOP
```

(a) What is the value of `MyString`?
0x1004

(b) What is the value of `MyBytes`?
0x1000

(c) What is the value of `MyVar`?
0x1010

(d) At what address does the first byte of the string 9876543210 start?
0x1004

(e) What is the value of `LOOP`?
0x2014

(f) Fill in the values in the following table for the values at `LOOP`:

	r15	r16	r17	r18
First time program reaches `LOOP`	0x1004	0xA	0x0	0x36373839
Second time program reaches `LOOP`	0x1005	0x9	0x38	0x38
Last time program reaches `LOOP`	0x100D	0x1	0x38	0x30

(g) What would be the consequences of leaving out `ENDLOOP: br ENDLOOP` at the end of our program?
If you don't put anything at the end of your code then the processor will just keep executing the instructions it finds in memory, which is just random data and thus random instructions.

[5 marks] 4. Fill in the NIOS-II assembly language program for the following segment of the code. Remember you must comment your code for full marks.

```
int i=0;
do{
     i++
} while(i < n);
```

```
.data
n: .word 10
i: .word 0

.text
main:
/* ADD YOUR CODE HERE */


    movia r8, i
    movia r9, n
    ldw r10, 0(r8)
    ldw r11, 0(r9)
DO: addi r10, r10, 1
    stw r10, 0(r8)
    blt r10, r11, DO
```

[10 marks] 5. Consider the following code for subroutine `boo`. (Line numbers are provided for your convenience):

```
( 1) boo:
( 2)        movi r8, 0
( 3) LOOP:  bge r8, r4, DONE
( 4)        add r10, r8, r8
( 5)        add r10, r10, r10
( 6)        add r9, r5, r10
( 7)        mov r4, r8
( 8)        call fun
( 9)        add r16, r16, r2
(10)        stw r2 0(r9)
(11)        add r8, r8, 1
(12)        br LOOP
(13) DONE:
(14)        mov r2, r16
(15)        ret
```

Code to save/restore registers has intentionally been omitted. List all registers that you will need to save on the stack prior to the instruction `call fun`. Also give the reason you must save the registers and the line number before which you would insert the code to save that register.

Register	**Reason why it needs to be saved**	**line number**
r8, r9	Caller-saved, used after call	8
r4, r5	Caller-saved, used in loop iteration after call	8
r16	Callee-saved and used in `boo`	2
ra	return address, must save if calling subroutine	2

[10 marks] 6. Devices and interrupts.

- A temperature sensor is connected to IRQ24 and is memory mapped at address 0x80000.
- The device's control register (CR) is at 0x80008.
  - Bits 0 to 7 set the low temperature threshold.
  - Bits 8 to 15 set the high temperature threshold.
  - If enabled for interrupts, an interrupt will trigger if the temperature falls below the low threshold or exceeds the high threshold.
  - Bit 16, if set enables interrupts.
  - The rest of the CR register contains several other bits that must be left unchanged when writing other bits. Reading the control register returns the current state of all bits.

Assuming that initially all interrupts are disabled, provide a sequence of instructions that configures this device and enable interrupts from this device. The low temperature threshold should be configured to be 20 and the high temperature should be configured to be 30. After your code executes, the processor should be accepting interrupt requests from this device only. Remember you must comment your code for full marks.

```
movia r8, 0x80008
movia r9, 0x00011E14 # threshold value + bit 16 for interrupt
ldwio r10, 0(r8)
movia r11, 0xFFFF0000
and r10, r10, r11
or r10, r10, r9
stwio r10, 0(r8)
movia r9, 0x01000000 # IRQ line 24
wrctl ctl3, r9
movi r9, 1
wrctl ctl0, r9 # set PIE bit
```

7. Consider the following sequence of assembly language instructions (assume 0 is a valid memory address)

```
movi r11,0x12
movi r12,0x77
movi r13,0x23
stb r11,0(r0)
stb r12,1(r0)
stb r0,2(r0)
stb r13,3(r0)
ldw r14,0(r0)
```

[4 marks] (a) Fill in the following memory addresses with their correct content after this sequence of instructions is executed by the NIOS II processor:

Address	Content
0x0003	0x23
0x0002	0x0
0x0001	0x77
0x0000	0x12

[4 marks] (b) What is the value of r14?

0x23007712

[10 marks] 8. Write a subroutine.

You are to write a subroutine that finds the minimum value from a list of N 32-bit unsigned numbers. The subroutine declaration is as follows:

```
void FINDMIN(int *addr_first_num, int *addr_list_size);
```

The subroutine takes two parameters. The first parameter is the address of the first number in the list. The numbers are stored in memory as an array. The second parameter is the address where the number of items in the list (N) is stored. These parameters will be passed via the stack (Note: this does not follow the ABI convention used in lab/lecture for passing parameters). The subroutine should store the result at memory address MIN. Remember you must comment your code for full marks.

```
findmin:
         ldw r8, 0(sp)   # first parameter (addr_first_num)
         ldw r9, 4(sp)   # second parameter (addr_list_size)
         ldw r11, 0(r9) # N
         ldw r10, 0(r8)
         movi r13, 1
         ble r11, r13, DONE  # check if list has more than 1 element
LOOP:    addi r8, r8, 4
         ldw r12, 0(r8)
         bgeu r12, r10, NEXT
         mov r10, r12
NEXT:    subi r11, r11, 1
         bgt r11, r0, LOOP
         movia r9, MIN
DONE:    stw r10, 0(r9)
         ret
```

9. Devices

The following device information is provided for you:

```
.equ ADDR_SLIDESWITCHES, 0x10000040 # base address for switches
.equ ADDR_GREENLEDS, 0x10000010 # base address for green LEDs
.equ ADDR_TIMER, 0x10002000 # base address for timer
```

[5 marks] (a) Write a simple program in assembly language that assigns the up/down positions of 8 switches to 8 Green LEDs to turn them On/Off. Your code only needs to capture these positions once. Remember you must comment your code for full marks.

```
movia r8, ADDR_SLIDESWITCHES
movia r9, GREENLEDS
ldwio r11, 0(r8)
andi r11, r11, 255
stwio r11, 0(r9)
```

[5 marks] (b) Use the timer that at one-second intervals inverts the state of the LEDs from part a (the word invert means that all LEDs that are ON should be turned OFF, and all LEDs that are OFF Should be turned ON). You do not need to consider changes to the switches; the position of the switches was captured in part a. Remember you must comment your code for full marks.

```
      .equ PERIOD, 50000000
      movia r10, ADDR_TIMER
      movi r13, %lo(PERIOD)
      stwio r13, 8(r10)
      movi r13, %hi(PERIOD)
      stwio r13, 12(r10)
      movi r12, 6
      stwio r12, 4(r10)
POLL: ldwio r13, 0(r10)
      andi r13, r13, 1
      beq r13, r0, POLL
      stwio r0, 0(r10)
      xori r11, r11, 255
      stwio r11, 0(r9)
      br POLL
```

This page is intentionally blank.

This page is intentionally blank.