UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
MIDTERM EXAMINATION, MARCH 2007

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

A. Moshovos and G. Steffan

This is a type D exam. You are allowed to use any printed material but not a calculator.

Last Name (Print): _____

First Name: _____

Student Number: _____

	Marks	Max. Marks
1		20
2		10
3		10
4		10
5		10
6		10
7		10
8		10
9		10
Total		100

Please:
State your assumptions. Show your work. Comment your code. **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.

 Last Name (in case pages get detached):_____

Marking starts at the maximum grade per question and you deduct as per the instructions. If you reach 0 you stop.

1. [20] Basic Instruction Semantics: Assume the following initial state for registers:

```
R8  = 0x12345678
R9  = 0xFAFB12AE
R10 = 0x01000100
R11 = 0xFFFFFFFE
R12 = 0xFFFFFFFF
```

In the following lists 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.

a. ADDI R2, R10, 2 R2 = _____0x01000102_____

b. ADD R2, R9, R12 R2 = _____0xFAFB12AD_____

c. ADD R2, R9, R11 R2 = _____0xFAFB12AC_____

d. OR R2, R8, R10 R2 = _____0x13345778_____

e. ORHI R2, R8, 0xFFFF R2 = _____0xFFFF5678_____

f. XORI R2, R8, 0x5678 R2 = _____0x12340000_____

g. ANDI R2, R11, 0x1111 R2 = _____0x00001110_____

h. MOVIA R2, 0x90000000 R2 = _____0x90000000_____

i. ADDI R2, R0, 0x0012 R2 = _____0x00000012_____

j. ADDI R2, R0, 0x8000 R2 = _____0xFFFF8000_____

-2 for each incorrect answer
Max -2 per point.
No partial marks for some digits being correct.

2. [10] Memory Operations: Assume the following initial state:

```
R8 = 0x01000100
R9 = 0x897601FE
```

What is the value of R2 when each of the following instruction sequences executes for little- and big-endian machines?

	Little-Endian	Big-Endian
STW R9, 0(R8) LDW R2, 0(R8)	0x897601FE	0x897601FE
STW R9, 0(R8) LDH R2, 2(R8)	0xFFFF8976	0x000001FE
STW R9, 0(R8) LDH R2, 0(R8)	0x000001FE	0xFFFF8976
STH R9, 0(R8) LDB R2, 1(R8)	0x00000001	0xFFFFFFFE
STH R9, 0(R8) LDBU R2, 1(R8)	0x00000001	0x000000FE

-1 for each incorrect answer
Max -1 per point.
No partial marks for some digits being correct.

3. [10] Instruction Semantics: if, initially, R8 = N, where N is a positive integer, how many times does the following loop iterate?

```
KOKO:   ADDI R8, R8, 1
        BGT  R8, R0, KOKO
```

Number of Iterations = _0x7FFFFFFF - N _

-5 if the answer is not exact
    +/- 1 is OK
-5 if the answer is not of the form CONSTANT - N

4. [10] Instruction Semantics: Please write the sequence of instructions that will execute given that execution starts at label KOKO below (instruction A). List instructions as A, B, and C as indicated to the left of each instruction in parentheses. We already listed the first execution of A for you.

```
(A)  KOKO:   CALL LALA
(B)          CALL KOKO
(C)  LALA:   RET
```

Sequence of executed instructions:

A, _C, _B, _A, _C, _B, _A, _C, _B, _A, _C, _B, _A, _C, _B, _A,

-2 if the second instruction is not C
-2 if the third instruction is not B
-8 if eventually they do not show the repeated sequence A, C, B

This scheme allows for slightly “shifted” execution sequences.

5. [10] Basic C and Assembly: Given the following assembly sequence:

```
     BEQ  R9, R10, KOKO
     BLT  R9, R10, KOKO
     BEQ  R0, R0, LALA
KOKO:
     ADDI R9, R9, 1
     ADD  R9, R9, R10
LALA:
     SUB  R9, R9, R10
```

Which C statement does it implement? Circle the appropriate sequence below. Finding what registers a and b correspond to is part of the question.

```
(a)  if (a <= b)
          a = a + 1;
     else a = a – b;

(b)  if (a > b)
          a = a + 1;
     else a = a – b;

(c)  if (a >= b)
          a = a + 1;
     else a = a – b;

(d)  if (a <= b)
          a = a + b + 1;
     else a = a – b;

(e)  if (a > b)
          a = a + b + 1;
     else a = a – b;

(f)  if (a >= b)
          a = a + b + 1;
     else a = a – b;

(g)  None of the above.
```

-10 if the answer is (c), (e), (f) or (g)
-8 if the answer is (b)
-9 if the answer is (d)

6. [10] Datastructures: Given the following definition:

```
struct    node_t {
     int                  data;
     struct    node_t     *left;
     struct    node_t     *right;
} array[10];
```

Assuming that R8 = base address of `array` in memory, and R9 = i (an integer variable), and the following assembly code:

```
ADD  R10, R9, R9
ADD  R10, R10, R10
ADD  R11, R10, R10
ADD  R10, R11, R10
ADD  R10, R8, R10
LDW  R10, 8(R10)
LDW  R10, 0(R10)
```

Which of the following statements does this assembly sequence implement? Circle the correct one:

(a) `array[i].data`

(b) `array[2 * i].data`

(c) `array[i].right`

(d) `array[i].left`

(e) `(array[i].right)->data`

(f) `(array[i].left)->data`

-8 if the answer is incorrect
-2 if the array index is not [i].
-2 if the term .left does not appear

7. [10] Calling Convention: Assume the following C declarations:

```
int foo (int a, int b, int c, int d);
int boo (void)
```

Fill in the prologue, post-call, and epilogue sections in the following assembly code. In the prologue you should determine which registers should be saved on the stack. In the post-call and the epilogue you should restore them. You should also determine the constants by which the stack pointer should be adjusted by. **Save registers on the stack so that they appear in numerical order with the register with the lower number appearing on the top of the stack. If a section is empty (there is no need for instructions in that section) then just write the comment "#empty".**

```
FOO:    ADDI SP, SP, __-12__
        # PROLOGUE – REGISTER SAVING STORES GO HERE
        STW  ra_, 8(SP)
        STW  r16_, 4(SP)

        # MAIN FUNCTION BODY
        ADD  R16, R5, R6
        ADD  R16, R16, R4
        # PRE-CALL, REGISTER SAVING STORES GO HERE
        STW  r7_, 0(SP)
        CALL BOO
        # POST-CALL, REGISTER RESTORING LOADS GO HERE
        LDW r7_, 0_(SP)
        ADD  R2, R7, R2
        ADD  R16, R16, R2
        ADD  R16, R16, R2
        LDW  R2, 0(R16)
        # EPILOGUE – REGISTER RESTORING LOADS GO HERE
        LDW r16_, _4_(SP)
        LDW ra_, _8_(SP)
        ADDI SP, SP, _12__
        RET
```

-2 for any register not saved
-2 for any register not restored at the right place or not restored at all
-2 for incorrect indexing in the stack
-1 for any incorrect stack adjustment (e.g., if +6 appears in the prologue and -6 in the epilogue subtract -2)
-1 for mistakes that cause wrong execution
OK if ADDI was changed to SUBI
Ignore the actual ordering of registers in the stack as long as it is consistent.

8. [10] Interrupt Handling: There are several steps to properly initializing an interrupt. Furthermore, when an interrupt happens several events occur. Put the following actions/events in proper order for both initialization and events that follow the occurrence of the interrupt (but before the interrupt handler starts executing). Simply give the ordered list of letters for each. Note that there are multiple correct orders, and that some items should be left out of the solution (i.e., there are decoys). Assume that PC contains the address of the instruction that was in the middle of executing when the interrupt was taken.

A) the value of PC+4 is written to the ea register
B) the value of PC is written to the ea register
C) the ea register is copied to the PC
D) ctl0 is copied to ctl1
E) ctl1 is copied to ctl0
F) the current instruction is aborted
G) the current instruction completes execution
H) interrupts are enabled at the device
I) the PIE bit in ctl0 is set to 0
J) the PIE bit in ctl0 is set to 1
K) the appropriate bit of ctl3 (ienable) is set
L) the PC is set to 0x1000020
M) ctl4 (ipending) is set to the address of the interrupt handler.

```
Initialization:
```
H-K-J

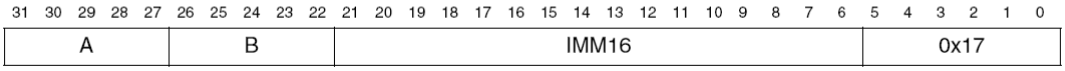
```
When interrupt occurs:
```

F/G-D-I-L-A

-1 per letter
(a letter is considered correct only and only if it appears in the right list or it does not appear at all if it shouldn't)

9. [10] Instruction Encoding: The following figure shows how LDW is encoded:

Instruction Fields: A = Register index of operand rA
B = Register index of operand rB
IMM16 = 16-bit signed immediate value

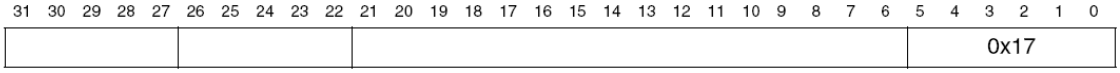


Instruction format for `ldw`

Where rB is the register written to with the value read from memory and rA is the base register used in the address calculation.

Encode the following instructions – Fill in the field in **binary**

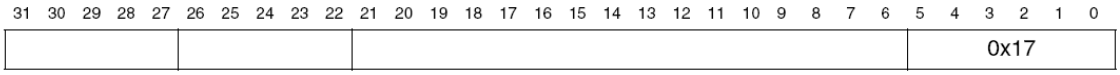
LDW R10, 0x100(R8)



Instruction format for `ldw`

01010 01000 0000 0001 0000 0000 0001 0111

LDW R15, -4(R9)



Instruction format for `ldw`

01111 01001 1111 1111 1111 11100 0001 0111

-2 per field that is wrong/ there are three fields per instruction

If all fields but one were wrong the total is 1