EC413 Computer Organization, Spring 2021
Homework 2 – Assembly Language Practice
Out: 2.8.21
Due: 2.16.21

1. [Instruction executions]
For each instruction given below provide the following:
i) Specify if the instruction is legal.
ii) If it is legal, evaluate it with the provided initial register values and give the value of $10 after each instruction execution, in HEX.

$1 = 0x0000D0A0 $5 = 0x00012BAD
$2 = 0x00000073 $6 = 0x0234EF01
$3 = 0xFFFFFF00 $7 = 0x00007B01
$4 = 0x00000FFF $8 = 0x00000005

a) add $10, $1, $2
**Legal**
**R 10 = 0x0000D0A0**
**+ 0x00000073**
**R10 = 0x0000D113**

b) sub $10, $2, $1
**Legal**
**R10 = 0x00000073**
**- 0x0000D0A0**
**= 115 - 53408**
**= -53293**
**R10 = FFFF2FD3**

c) and $10, 0x00000345, 0x00000001
**illegal**

d) lw $10, $8($7)
**illegal**

e) add $10, $3, -10
**illegal**

f) or $10, $3, 0x000000AB
**illegal**

g) and $10, $5, 0x000000AB
**illegal**

h) xor $10, $6, $5
**Legal**
**R10 = 0x0234EF01**
**Xor 0x00012BAD**

**=0x023-0001/0100->0101->5-00010/1110->1100->12/C-1111/1011->0100->9-A-0001/1101->1100->12/C**

**r10 =0x0235C09AC**

i) sll $10, $7, 3

**Legal**

**R10 = 0x00007B01**

**= 0x0000-0111-1010-0000-0001**

**=0011-1101-0000-0000-1000**

**r10=0x0003D008**

j) sra $10, $6, 2

**Legal**

**R10**

**= 0x0234EF01**

- **Calc done in c**
  - **= 0x008D3BC0**

k) sra $10, $3, 1

**Legal**

**R10 0xFFFFFF00**

**>>3**

**= 0xFFFFFF80**

2. [Memory Access]

- Given the following memory map, for each instruction:

i) Describe what it does

ii) Write down the address of the memory operand

iii) Write down what value is written to the destination register or memory location.

iv) If the instruction is illegal or results in an error, say so.

- Treat the instructions as a program, not individual instructions. Assume little endian.
- Write the final content of the memory and registers.

LABEL ADDRESS INITIAL CONTENTS

a: 0 0xA0
1 0xA1
b: 2 0xA2
3 0xA3
c: 4 0xA4
5 0xA5
6 0xA6
7 0xA7
d: 8 0xA8
9 0xA9
10 0xB0
11 0xB1
e: 12 0xB2

f: 13 0xB3
g: 14 0xB4
h: 15 0xB5
i: 16 0xB6
17 0xB7
18 0xB8
19 0xB9

LA $t0,a**// $t0 a:0||v:unf ||lLL:no**
LW $s0,d **// $s0 a:unf ||0xA8A9B0B1|| lLL:no**
SB $t0,13($zero)**// a||0||0 iIl:no**
LA $t3,f **//$t3 a:13||v:unf ||lLL:no**
LB $t4,0($t3)**// $t4 a:unf ||v:B3 ||lLL:no**
SW $t0,i **// a||0||B6B7B8B9 iIl:no**
LH $t1,0($t3)**//$t4 a:unf ||v:0xB3B4 ||lLL:no**
SH $t1,g **lLL :yes**
LW $t5,17($zero) **//$t5 a:unf ||v:B6B7B8B9 ||lLL:no**
LBU $t5,f **//$t5 a:unf ||v:0xB3 ||lLL:no**

3. [Translating C code fragments to MIPS assembly]
Given the following C declaration and code, write the appropriate MIPS assembly code. Use the appropriate loads and stores. You do not need to give the assembly language declarations.

```
main()
{
int ia = 7;
int ib = 0x23;
int ic,id,ie,ig;
ia = 0x1234;
ib = ia;
ic = ia + ib;
id = ic | ib & 17;
ie = ~ig;
ig = (ia - ib) ^ (ic + id);
}
```

```
.data
ia:     .word 7 #int ia = 7;
ib: .word 35 #int ib = 0x23;
ic: .word 0 #initialize all non predone numbers to 0 int ic,id,ie,ig;
id:     .word 0
ie: .word 0
ig: .word 0
```

```
.main
        la $t0,ia
        sw $t0,0x1234 #ia = 0x1234;

        lw $t1,ia #ib = ia;
        la $t2,ib
        sw $t2,$t1

        la $t3,ic #ic = ia + ib;
        add $t4,$t1,$t1
        sw $t3,$t4

        la $t5,id #id = ic | ib & 17;
        or $t6,$t4,$t1
        andi $t6,$t6,17
        sw $t5,$t6

        la $t7,ie  #ie = ~ig;
        lw $t8,ig
        nor $t8,$t8,$t8
        sw $t7,$t8

        la $t8,ig  #ig = (ia - ib) ^ (ic + id);
        sub $t0,$t1,$t1
        add $t1,$t4,$t6
        xor $t1,$t1,$t2
        sw $t8,$t1
```

4. [Translating C to MIPS assembly – Conditionals]
Rewrite the following C code in MIPS assembly language. Be sure to include all the code needed to access memory. You do not need to give the assembly language declarations.

```
int a = 4;
int b = 30;
int c = 20;
int d = 10;
if (a == b) c = 33;
else if (b != c) a = 20;
else {
 if (a > b) b = 10;
 else if (c <= 10) c = 12;
 else a = 5;
}
```

```
        main:
```

```
        lw $t0,a
        lw $t1,b
        lw $t2,c
        lw $t3,d
        bne $t0,$t1,else_1
        la $t4,c
        sw $t4,33
        j done1
else_1:
        beq $1,$t2,else_2
        la $t4,a
        sw $t4,20
        j done1
else_2:
        ble $0,$t1,else_3
        la $t4,b
        sw $t4,10
        j done1
else_3:
        bgt $t2,10,else_4
        la $t4,c
        sw $t4,12
        j done1
else_4:
        la $t4,a
        sw $t4,5
        j done1
done1:

Exit:
   jr $ra
```

5. [Memory and data references]

a) What's the difference between LI, LA, and LW?

- Load address stores the address in a designated register,
- Load word stores the word value in the register
- Load immediate stores a constant in a register

b) Translate C to AL (you will need LI, LA, and LW!).

You do not need to give the assembly language declarations.

```
        int VarA,VarB,VarC;
        int *VarE, *VarF;
        int VarD = 10;
        int ArrayA[100];
```

```
        VarA = 20;
        VarB = VarA;
        ArrayA[1] = VarB; // careful with this one (from previous year's mid-term)
        VarF = &VarE;
```

```
main:
      la    $t0,VarA     # load $t1 with the address of array numbs
      Sw   $t0,20         # t0 is is a counter initializer
      la    $t1,VarB
      lw    $t2,VarA
      sw   $t1,$t2

      la    $t3,4(ArrayA)
      sw  $t3,$t2

      la    $t4,VarE
      la    $t3,VarF
      sw   $t3,$t4
Exit:
   jr $ra
```

6. [Loops]
In class, we talked about two methods of doing data transfers using for loops, (i) computing the A[i] and B[i] every iteration and (ii) using pointers which were incremented every iteration. In this problem, code data transfer with a loop using a different variation: Use A and B in the displacement slot and use a register to be the offset. For example, your memory operands for SW and LW could be A($t0) and B($t0), respectively. Demonstrate with array A containing 6 words (e.g. 1,2,3,4,5,6), that are copied to array B.

```
        .data
A       .word 1,2,3,4,5,6
B       .space 24
Index  .word 0

        .text
Main:
        Lw $t0,index
        La $t2,B
Beg_loop:
        Beq $t0,6,Loop_over
        Lw $t1,$t0(A)
        Sw $t0($t2),$t1
        Add $t0,$t0,4
```

**J Beg_loop**

**Loop_over:**
**Exit:**
**jr $ra**

7. [Shifts and logicals]
Give the instruction or instruction sequence to perform the following functions.
Assume little endian.

a) Negate $s0
**Add $s0,0x80000000**
b) Take the two's complement of $s1
**Xori $t0,$s1,0xFFFFFFFF**
**Addi $s1,$t0,0x00000001**
c) Set bits 7, 14, 15, 26, and 29 in $s2
**Ori $s2,$s2,0x1400A080**
d) Clear bits 2, 5, and 11 in $s3
**Xori $s3,$s3, 0x00000812**
e) Branch if and only if bits 1, 12, and 13 of $s4 are set
**And $t0,$s0,0x00001801**
**Bgt $t0,0,dest**
f) Shift $s5 to the right by 6 while preserving the sign
**Sra $s5,6**
g) Move bits 5, 6, and 7 of $s6 to bits 0, 1, and 2 while clearing all other bits.
**Sll $s5,1**
**Srl $s5,5**