EE 457 HW 3
Instruction Sets
Redekopp • Puvvada

Name: _____

Due: See Website **Score: _____**

Please post any questions regarding HW problems on Piazza.

Refer to the MIPS32 instruction descriptions available at http://ee.usc.edu/~redekopp/ee457/MIPS_Vol2.pdf. A few of the more important instructions are listed below but others will also be needed.

Instruction	PDF Page / Page in MIPS_Vol2
ADD	39 / 31
ADDI	42 / 34
ADDU	44 / 36
LUI	137 / 129
ORI	185 / 177
SLT	203 / 195
SLTU	206 / 198
SUB	213 / 205
SUBU	215 / 207

This homework contains some problems from the 1st, 2nd, and 3rd edition of "Computer Organization and Design" by Hennessy and Patterson. We reproduce the problems below.

For the following questions, find the shortest MIPS instruction sequence (a single instruction if possible) that performs the given operation. Many of these instructions are taken from the DEC VAX instructions set.

1. **(2) [1st Ed., Q3.1]** In some cases a simple instruction set like MIPS can synthesize instructions found in richer instructions sets such as the DEC VAX. The following VAX instruction decrements register $5:
```
decl $5    # register $5 = $5 - 1
```

2. **(3) [1st Ed., Q3.2]** This VAX instruction now clears register $5:
```
clr $5    # register $5 = 0
```

3. **(4) [1st Ed., Q3.3]** This VAX instruction now clears memory location 1000:
```
clr 1000    # memory[1000] = 0
```

4. **(6) [1st Ed., Q3.4]** This VAX instruction adds 1 to register $5, placing the sum back in register $5, compares the sum to register $6, and then branches to L1 if $5 < $6:
```
aoblss $6, $5, L1   # $5 = $5 + 1; if($5 < $6) goto L1;
```

5. **(6) [1st Ed., Q3.5]** This VAX instruction subtracts 1 from register $5, placing the difference back in register $5, then branches to L1 if $5 > 0:

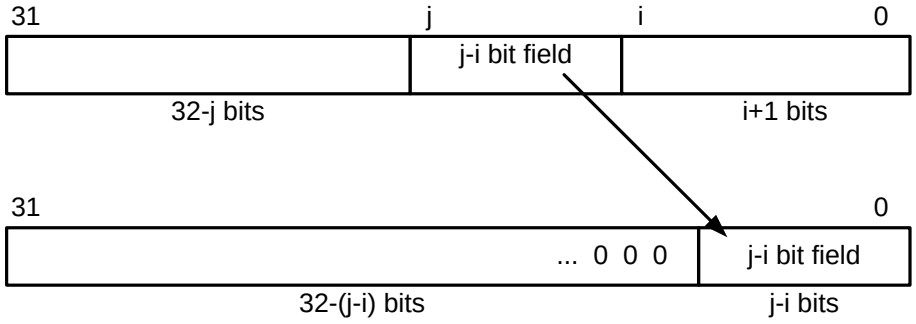
```
sobgtr $5, L1   # $5 = $5 - 1; if($5 > 0) goto L1;
```

6. **(6) [3rd Ed., Q2.59]** Show the minimal MIPS instruction sequence for a new instruction called 'SWAP' that exchanges the values in two registers. After the sequence completes, the destination register has the original value of the source register, and the source register has the original value of the destination register. Convert this instruction (write an equivalent MIPS sequence):

```
swap $s0, $s1
```

   The hard part is that this sequence must use only these two registers! [Hint: It can be done in three instructions that involve XOR operations. Consider what value results from: A xor B xor A].

7. **(6) [3rd Ed., Q2.6]** Some computers have explicit instructions to extract an arbitrary field from a 32-bit register and to place it in the least significant bits of a register. The figure below shows the desired operation:



   Find the shortest sequence of MIPS instructions that extracts a field for the constant values of i=5 and j=22 from register $t3 and places it in register $t0. (Hint: It can be done in two instructions using 'sll' and 'srl'.)

For the following problems, assume that you only have the MIPS instructions: **bne, beq, slt, and sltu**. Assume you DO NOT have the MIPS instructions like bgez, bltz, etc.

Assume that **i** and **j** are **SIGNED** numbers represented in the 2's complement system. Note that 1111 is less than 0111 in 4-bit signed numbers since 1111=-1 decimal while 0111=+7 dec. You may assume that i and j are stored in registers $2 and $3, respectively.

8. **(4) [1st Ed., Q3.14]** Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i < j) goto L1;
```

9. **(4) [1st Ed., Q3.15]** Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i <= j) goto L1;
```


10. (4) [1st Ed., Q3.16] Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i > j) goto L1;
```

11. (4) [1st Ed., Q3.17] Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i >= j) goto L1;
```

Repeat the problems but now assuming that at **i** and **j** are **UNSIGNED** numbers. Note that now 1111 is greater than 0111 in 4-bit unsigned numbers since 1111=+15 decimal while 0111=+7 dec.

12. (2) Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i < j) goto L1;
```

13. (2) Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i <= j) goto L1;
```

14. (2) Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i > j) goto L1;
```

15. (2) Find the shortest MIPS sequence (using the restrictions above) to achieve the following operations.

```
if(i >= j) goto L1;
```

16. (12) Consider how to implement the following CISC instruction (of some fictitious CISC processor) using MIPS instructions. Write a sequence of MIPS instructions to perform the same operation. [**Hint**: Use LUI, ORI, LW, ADD, SW instructions.]

```
add C, A, B  # (C) = (A) + (B)
```

In this instruction, A and B are the addresses of locations in memory containing the source operands. The address of the destination location is C. All addresses point to 32-bit word values. In your answer you can represent the upper and lower 16-bits of A as "A-upper" and "A-lower", respectively, and similarly for B and C.

17. (6) The MIPS '**LW**' instruction: **`lw $rt, offset16($rs)`**, allows specifying a 16-bit offset only. Devise a sequence of MIPS instructions, which will be equivalent to the instruction:

```
lw $rt, offset32($rs)
```

where offset32 is a 32-bit offset. Hint: perform the address computation **`offset32($rs)`** using separate instructions.

18. (3) MIPS does not support a '**NOT**' instruction to perform logic inversion of all bits in an operand. Find a suitable substitute for the fictitious instruction:

```
not $3, $4  # $3 = ~$4
```

Note that MIPS does provide an XOR instruction and a NOR instruction. Select the shortest possible sequence to achieve the operation.

Little vs. Big Endian and Double-precision Arithmetic

19. (6) Mr. Hasty wrote code to perform 64-bit addition using the 32-bit addition MIPS instructions. The two source operands are in the memory at the (symbolic) addresses A and B. The result is to be put in memory at the (symbolic) address C. Mr. Hasty did NOT check whether the system uses little- or big-endian format. Mr. Hasty's code did not work. Explain possible reasons why and illustrate your reasoning with a brief example.

20. (4) Suppose that the content of a 32-bit word is 0x12345678 and its address is 0x00000040. What is the content of the byte at 0x00000042 if we use a little-endian system? What would be the content of the mentioned memory location if big-endian system was used? (Clearly label your answer for each endian-ness)

21. (12) Refer to the descriptions of ADD, ADDU, SUB, SUBU, SLT, and SLTU instructions. Understand the difference between ADD and ADDU and SUB and SUBU. Next, understand the double-precision (64-bit) unsigned addition code below. Then, write a double-precision unsigned subtraction method.

```
// Assume little-endian system for 64-bit integers
// $11 and $10 together hold a 64-bit operand
// $11 has the upper 32-bits; $10 has the lower

// code to perform unsigned 64-bit addition
// $15:$14 = $13:12 + $11:$10

ADDU $14, $12, $10 // $14 = $12 + $10
SLTU $15, $14, $10 // if $14 < $10 Then $15=1
                   // else $15=0
ADDU $15, $15, $11
ADDU $15, $15, $13

// Use ADD instead of ADDU for the last two ADDs if
// overflow "exception"/"trap" is needed.
```

Write a double-precision (64-bit) unsigned subtraction code to perform

```
// $15:$14 = $13:12 - $11:$10
```

Do not bother about overall unsigned overflow beyond 64-bits but you do need to check if a BORROW needs to be propagated across the lower and upper portions of the subtraction by comparing $12 with $10.

