Computer Science 605.611

Problem Set 2

(Appendix A of the textbook defines and describes all MIPS instructions. Instructions not identified as pseudo-instructions in appendix A are true-op instructions.

To demonstrate your understanding of the concepts, your answers on problem sets should be produced manually without the use of a simulator or other outside source.)

1. Recall that “**true-op”** instructions are those that can be translated by the assembler into a single 32-bit built-in machine instruction. True-op instructions can only use addressing modes that are supported by the built-in machine instructions and cannot use immediate operands outside the range -32768 to +32767.

a) (3) Write down one or more MIPS true-op instructions (no pseudo-instructions) to add the decimal constant +250000 to register $t0. Any constants you use should be decimal.

To start I will mention that 250000 is too large to be moved all at once, thus

**li $t0, 32767 (load $t0 with max number)**

**mul $t0, $t0, 7 (multiply 32767 by 7 which is 229369)**

**add $t0, $t0, 20631 (add the remaining value to make it 250000)**

b) (3) Write down one or more MIPS true-op instructions (no pseudo-instructions) to add the decimal constant -250000 to register $t0. Any constants you use should be decimal.

To start I will mention that -250000 is too small to be moved all at once, thus

**mul $t0, 32767, 7 (multiply 32767 by 7 which is 229369)**

**add $t0, $t0, 20631 (add the remaining value to make it 250000)**

**sub $t0, $0, $t0 (subtract 250000 from 0 leaving us -250000)**

c) (3) Write down one or more MIPS true-op instructions (no pseudo-instructions) to add the decimal constant 32768 to register $t0. Any constants you use should be decimal.

This value is just outside of the range, thus

**li $t0, 32767 (load $t0 with max number)**

**add $t0, $t0, 1 (add 1 to $t0 value and store it in $t0, thus 32768)**

d) (3) Write down one or more MIPS true-op instructions (no pseudo-instructions) to add the decimal constant -32768 to register $t0.

This is within the range of MIPS, thus

**li $t0, -32768 (load $t0 with lowest number)**

2. For each of the decimal integers listed below, what is the minimum number of bits required to represent that integer in two’s complement form if only one sign bit is included?

a) (3) 30800

To start we know that one sign bit is required,

If we start to convert the number itself we can get to 30800 in only 15 bits

**16 bits is the minimum**

b) (3) -32768

Always add one for sign bit

This is actually the max number if you do 1000000000000000 to decimal

**16 Bits is the minimum**

c) (3) -6

in twos complement 1010 = -6,

**4 Bits is the minimum**

d) (3) 32768

Since this the negative of B, all we need to do is add an extra bit to the front, 01000000000000000 (cant be represented in 16 bits)

**17 Bit minimum**

3.(5) A system uses 16 bits to represent signed integers in excess-4000 form (i.e., the bias is 4000). Fill in the blanks to show the range of signed integers that can be represented in this excess-4000 system. The range is \_-4000\_\_\_ to \_ 36768\_\_\_.

Since range is -bias to (2^n-1) – bias, then we have -4000 to (2^15)-4000

4. (5) For each of the following instructions, indicate whether the instruction is a true-op or is a pseudo-instruction.

li $4, -72 Pseudo-instruction

lui $5, 0x8000 True op

addiu $,3, $9, 51902 True Op

ori $7, $0, 0xCABE True Op

andi $11, $6, -13634 True Op

5. Assume that floating point register $f8 contains the 32-bit floating point representation of the value -2.5. CPU register $4 contains the 32-bit two’s complement representation of 30. Registers $f8 and $4 are set to these patterns prior to executing each of the instruction sequences listed below.

a) (5) The following pair of instructions places a 32-bit pattern into register $2. What decimal integer has the resulting pattern as its two’s complement representation?

mfc1 $2, $f8

addu $2, $2, $4

We can determine what the decimal integer by looking at what it is doing. The first line copies the floating point into register $2. Then the second line adds the value in integer register $4 to integer in register $2. Or in easier terms it takes -2.5 and adds it to 30 which is **27.5**.

b) (5) The following pair of instructions places a 32-bit pattern into register $2. What decimal integer has the resulting pattern as its two’s complement representation?

mtc1 $4, $f8

xor $2, $4, $0

Again we can determine this by looking at the instructions. The first line copies $4 into $f8 (30 into $f8). The second performs XOR between $4 and $0 and stores it in $2. Since $0 is all 0s the XOR operation does not affect the stored value. So the Decimal integer is **30**

c) (5) The following instruction sequence places a 32-bit pattern into register $2. What decimal integer has the resulting pattern as its two’s complement representation?

cvt.w.s $f6, $f8

mfc1 $2, $f6

xor $2, $4, $2

For this one I will follow the same steps. Line 1 converts the value in $f8 (-2.5) into floating point representation and stores that in $f6. The second line copies that value to register $2. And finally, the final line performs the bitwise XOR operation between $2 and $4 storing the result in $2. So -2.5 is -10.1 and 30 is 11110. The XOR would then be

11110.0

00010.1 (Added 0’s for alignment)

11100.1 or **28.5**

6. (5) Floating point register $f8 contains the 32-bit floating point representation of -8761. The following instruction is then executed:

cvt.s.w $f6,$f8

Use eight hex digits to show the resulting bit pattern contained in $f6.

$f6 = 0xFFFF223D\_\_\_\_\_\_\_\_

Fist we convert -8761 into its 32 bit rep in binary twos complement 1101110111000011

To get that to hex its 4 bits per hex, so 1101 1101 1100 0011, which turns to 223D, we then need to pad the front with 0’s for 32 bits, so FFFF223D

7. (5) If each character is encoded as an 8-bit ASCII character, a 32-bit register can contain 4 characters. Assume that register $3 contains the two’s complement representation of +1094861636 and register $4 contains the two’s complement representation of +538976288. After the instruction addu $2,$3,$4 is executed, show the 4 ASCII characters represented by the 32-bit pattern produced in register $2. Display the ASCII characters (not their 8-bit codes) from left to right (i.e., from the high byte to the low byte within the register).

So to start lets add the two registers together as that is what is stored in $2. 1094861636 + 538976288 = 1633837924. We then need to translate that into its two bit representation, 01100001011000100110001101100100, ASCII is 8 bit codes so lets split that up, 01100001 01100010 01100011 01100100. Then translate those to Hex, 61 62 63 64. Then to ASCII **abcd,** which should be the answer

8. Since the MIPS processor uses 32-bit addresses, it can reference any location within the 4-GB address space. The bytes in memory are numbered consecutively starting from 0x00000000 at the low end of memory up to 0xFFFFFFFF at the upper end of memory.

a) (5) The jump instruction j loop transfers control to the instruction to which the label “loop” is attached (i.e., it jumps to loop). Assume that the machine code for this jump instruction is located at memory address 0x20CE88C0 and that the label “loop” corresponds to memory address 0x2C4E088C. What is the 32-bit machine code for this jump instruction? Express your answer as an 8-digit hex number. The jump instruction is described in modules 1 and 2 as well as in appendix A of the textbook.

There are two parts of the jump address. The Op code and target address, op code is 6 bits and target address is 26 bits. Op code is always 2 or 000010 (6 bits), we have to get the offset for the true target address (loop local – jump local). We end up with 1011011111110111111111001100 which is 28 bits, 2 to large for 26. I truncated the end to have 26 bits total. Thus we have 00001010110111111101111111110011 as the jump code in binary, converting it to hex gives us:

**0xADFDFF3**

b) (5) The conditional branch instruction beq $0, $0, exit always transfers control to the instruction to which the label “exit” is attached (since it compares register $0 with itself). Assume that the branch instruction is located at memory address 0x20CE68C0 and the label “exit” corresponds to memory address 0x20CE88C4. What is the 32-bit machine code for this beq instruction? Express your answer as an 8-digit hex number. The beq instruction is described in modules 1 and 2 as well as in appendix A of the textbook.

The beq instruction is made up of four parts. The first is always 4 (in 6 bit form), then 5 bits of the first register and 5 bits of the second register, plus the offset. In good news for me register $0 is made up of all 0s, so 10 0s. so far we have 0001000000000000. Now we just need the offset, offset is target – current address, so 0x20CE88C4 - 0x20CE68C0 = 2004, which in binary is 0010000000000100. So our final binary address is 00010000000000000010000000000100 which in hex is **0x10002004**

c) (5) A jump instruction transfers control to some 32-bit memory address within the instruction memory. Assume that the 32-bit machine instruction at that address is 0x00000000. This target instruction (0x00000000) is an R-type instruction. Indicate the ALU operation performed by this R-type instruction and indicate the value contained in the result register.

The R-type format is as follows, 6 bits of an op code, 5 bits for each register (3 in total), 5 bits for the shift amount and 6 bits for the function code. If the instruction is all 0’s than only one operation fits that bill (technically 2), **slt or sltu**

9. Assume that CPU register $11 is filled with the pattern 0xCAFEF00D prior to executing each of the instructions listed below. Use 8 hex digits to show the contents of $11 after each instruction is executed.

a) (3) addiu $11, $11, -2 contents of $11 = 0xCAFEF00B\_\_

b) (3) addi $11, $11, -2 contents of $11 = 0xCAFEF009\_\_

c) (3) ori $11, $11, -2 contents of $11 = 0xCAFEF009\_\_

d) (3) xori $11, $11, -2 contents of $11 = 0xCAFEF00B\_\_

e) (3) andi $11, $11, -2 contents of $11 = 0xCAFEF008\_\_

f) (3) sra $11, $11, 6 contents of $11 = 0xF2BFD801\_\_

10. (3) Assume that register $15 contains some signed integer value N (in two’s complement form). Write down one or more MIPS true-op instructions (not pseudo-instructions) that place the two’s complement representation of -N into register $16. That is, the negative of the integer value in $15 is placed into $16.

There is actually quite a nice way to do this, we can think about just subtracting the number from 0 which will get us a negative. Thus:

**sub $16, $0, $15**

will move -N into register $16, this would be assuming that the number is within the limits of MIPS 32 bit size. (i.e. -32768 to 32768)

11. (5) Based on Booth’s algorithm, how many additions and how many subtractions should be performed in multiplying the decimal integer 9040 by the decimal multiplier -592 .

So in this instance 9040 is the multiplicand and -592 is the multiplier.

We need to convert both to binary,

9040 is 10001101010000

-592 in signed twos complement is -1001000000

We should then “record” the multiplier, this becomes: -1 0 +1 -1 0 0 0 0 0 0, Thus we have all the information we need for the amount of additions and subtractions:

**We need two subtractions and one addition**