Linh Nguyen

B04

ENCM 369 – Computer Organization

Lab 6

Ex. A) Part I

1. 160 is 10\*16, so the 16-bit representation of 160 is

0000\_0000\_1010\_0000

And the 16-bit representation of -160 is

1111\_1111\_0101\_1111 + 1 = 1111\_1111\_0110\_0000

$sp ($29) is both the source and destination GPR, therefore both register fields are 11101. The op-code for addiu is 001001.

Therefore, the machine code for this instruction in base two is

001001\_11101\_11101\_1111\_1111\_0110\_0000

1. The adder inputs are the $sp value and the 32-bit sign-extension of -160:

Carry-in: 1111\_1111\_1111\_1111\_1111\_1111\_1000\_0000

$sp: 0111\_1111\_1111\_1111\_1110\_1000\_1100\_0000

-160: 1111\_1111\_1111\_1111\_1111\_1111\_0110\_0000

--------------------------------------------------

output: 0111\_1111\_1111\_1111\_1110\_1000\_0010\_0000

Part II

1. To use the address of 0x1002\_b000, the offset for the lw instruction would need to be the unsigned 16-bit number 0xb000, but offsets in load and stores are signed 16-bit integers, with the largest number in that set being 0x7fff which is smaller than 0xb000.
2. The pseudoinstruction can be implemented by giving lui an address greater than 0x1002\_0000 to put into $at and then using a negative offset with lw to get the desired address of 0x1002\_b000.

If the address given to lui is 0x1003, then the negative offset needed to get to 0x1002\_b000 would be -0x5000. In base ten, that is –(5\*(16^3)) = -20480. Therefore the correct instructions is

lui $at, 0x1003

lw $s0, -20480($at)

Ex B)

Part I

Carry in: 0000\_0000

a: 1001\_0000

b: 1100\_0000

----------

Sum: 0101\_0000

Signed overflow: Yes. a and b are both negative numbers, but sum is a positive number, therefore there is overflow.

Unsigned overflow: Yes. Sum < a

Part II

Carry in: 0000\_0000

a: 0100\_0101

b: 0011\_1000

----------

Sum: 0111\_1101

Signed overflow: No. a and b are both positive numbers, and so is sum.

Unsigned overflow: No, sum > a

Part III

Carry in: 1000\_0000

a: 0100\_0101

b: 0100\_1000

----------

Sum: 1000\_1101

Signed overflow: Yes. a and b are both positive numbers, but sum is a negative number, therefore there is overflow.

Unsigned overflow: No. Sum > a

Part IV

Carry in: 1110\_0100

a: 1111\_0010

b: 1111\_0011

----------

Sum: 1110\_0101

Signed overflow: No. a and b are both negative numbers, and so is sum.

Unsigned overflow: Yes. Sum < a

Ex E)

Part I

Carry in: 1110\_0001

a: 0111\_0000

inverted bits of b: 0001\_0110

----------

result: 1000\_0111

Signed overflow: Yes. a is positive and b is negative, but the result seems to be negative.

Unsigned overflow: Yes, result > a

Part II

Carry in: 0000\_0111

a: 1001\_0011

inverted bits of b: 1110\_1011

----------

result: 0111\_1111

Signed overflow: Yes. a is negative and b is positive, but the result seems to be positive.

Unsigned overflow: No, result < a

Part III

Carry in: 1001\_1011

a: 1100\_0101

inverted bits of b: 0100\_1101

----------

result: 0001\_0011

Signed overflow: No. a is negative and b is negative, so the result could be negative or positive.

Unsigned overflow: No, result < a

Part IV

Carry in: 0000\_0011

a: 0100\_0001

inverted bits of b: 0000\_0001

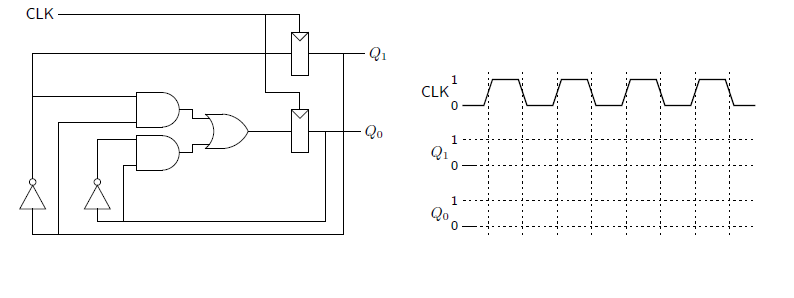
----------

result: 0100\_0011

Signed overflow: No. a is positive and b is negative, and the result is to be positive.

Unsigned overflow: Yes, result > a

Ex F)

 Part I

Part II

