CISC 260 Machine Organization and Assembly Language (Spring, 2018)

Assignment # 3

(Due: March 13, 2018

The following instruction set is supported by a simple processor, which is similar to what we discussed in the class, with a few new instructions added. The format of most instructions is defined as follows.

bits	15:14	13:10	9	8:6	5:3	2:0
field	unused	opcode	w	src1	src2	dst

where the fields are defined as follows.

opcode : operation to be performed by the processor
w: write back ALU output to register file (1 = yes, 0 = no)
src1: address of the first ALU operand in the register file
src2: address of the second ALU operand in the register file
dst: address in the register file where the output is written

For opcodes BEQ, BLEZ and JUMP, the 6 least significant bits (5:0) give an address in the instruction memory, which is byte-addressed. The opcode HALT has all operand bits (9:0) being 0. When an instruction has only two operands, the field for the unused operand is filled with 0-bits. For example, bits (5:3) for SLL are all zero because src2 is not used.

The opcode and meaning of these instructions are listed in the following table.

opcode	Binary encoding	Operation
ADD	0x0	R[src1] + R[src2] → R[dst]
SUB	0x1	R[src1] – R[src2] → R[dst]
SLL	0x2	R[src1] << 1 → R[dst]
SRL	0x3	R[src1] >> 1 → R[dst]
INV	0x4	~R[src1] → R[dst]
XOR	0x5	R[src1] ^ R[src2] → R[dst]
OR	0x6	R[src1] | R[src2] → R[dst]
AND	0x7	R[src1] & R[src2] → R[dst]
INCR	0x8	R[src1] + 1 → R[dst]
BRZ	0x9	If R[src1] = 0, branch to BAddr
BLEZ	0xA	If R[src1] ≤ 0, branch to BAddr
JUMP	0xE	Jump to JAddr
HALT	0xF	Stop execution

1. In this part you are asked to recognize what a given segment of machine code does, by translating it into assembly code, and annotating it with expressions in a high level language like C. This process,

from binary code back to source code, is called "disassembling." Disassemble the following machine code into operations and arguments, e.g., ADD R1, R2, R3. Explain what the whole program does, either in plain English (e.g., it calculates the product of R1 and R2) or in C type of pseudo code (e.g, R3 = R1 * R2.) Note: you need to make up unique labels for your assembly code.

What do you mean by "unique label"

Addr:	Code
0x00:	07 FF
0x02:	06 08
0x04:	28 08
0x06:	38 02
0x08:	02 47
0x0A:	3C 00

2. If the register files (8-bit, byte addressed) have the initial image as the following, what are the values of these 8 registers when the execution of the program in part 1 stops?

R0: 0001 1101
R1: 0000 0111
R2: 0000 0000
R3: 0000 0000
R4: 0000 0000
R5: 0000 0000
R6: 0000 0000
R7: 0001 0000

3. Translate the following segment of C program into assembly code for the machine given above. Note: you may assume two positive integers a and b are already given in register R0 and R1. By convention, put the returned value in R7.

```
while (a!=b) {

        if (a > b) {
                a = a - b;
        } else {
                b = b – a;
        }
}
return a;
```