COMP 4300
SIMPLE RISC CPU INSTRUCTION SET

Memory Organization
The memory of the SimpleRisc system is word-addressable. That is, a 4-byte (32-bit) value is stored at each address. This is done for simplicity: you only have to increment the PC by one to read the next instruction rather than by 4.

Instruction Format
ALU instructions are 1 word (32-bits) long. All other instructions (load, store, and jumps) need an address, so they are two words long. The first word holds the instruction, and the second word holds the address to load, to store, or to jump to, depending on the type of the instruction.

There are 32 general purpose registers in the SimpleRisc CPU, so a register number (Operand1, Operand2, or Destination) is specified as an unsigned 5-bit binary number (e.g. register 7 is 00011).

Opcode	Operand1	Operand2	Destination	Function	Not used
31 - 26	25 - 21	20-16	15 - 11	10-7	6-0

ALU Operation (Opcode = 000010, Mnemonic based on Function code)
Perform the ALU operation specified by the Function code on the registers in numbers Operand1 and Operand2, and store the result in register Destination. Note that these function codes are the same width and the same meaning as the ALU of the SimpleRisc which was developed in a prior lab assignment.

Function codes for Opcode 000010
0000 = unsigned add (mnemonic UADD)
0001 = unsigned subtract (USUB)
0010 = two's complement add (ADD)
0011 = two's complement subtract (SUB)
0100 = two's complement multiply (MUL)
0101 = two's complement divide (DIV)
0111 = bitwise AND
1001 = bitwise OR
1011 = bitwise NOT of operand1 (ignore operand2)
1100 = pass operand1 through to the output (OP1)
1101 = pass operand2 through to the output (OP2)
1110 = output all zero's (ZERO)
1111 = output all one's (ONE)

Example of ALU operation:

ADD R3, R1, R2

000010 00001 00010 00011 0010 0000000

Load Operation (Opcode = 000000, Mnemonic = LD)
Add (unsigned) the contents of register Operand1 to the base address specified in the second word of the instruction to get the address to load. Load the contents of that address (base + offset) into the register specified as Destination. For this instruction Operand2 and the Function bits are not used.

Example of LD Operation:

LD R1, R2(0x1234)

000000 00010 00000 00001 0000 0000000
00000000 00000000 00010010 0011 0100

Store Operation (Opcode 000001, Mnemonic = STO)
Add (unsigned) the contents of register Destination to the base address specified in the second word of the instruction to get the address to load. Store the contents of register Operand1 into the address (base + offset). For this instruction Operand2 and the Function bits are not used.

Example of STO operation:

STO R1, R2(0x1234)

000001 00001 00000 00010 0000 0000000
00000000 00000000 00010010 0011 0100

Jump Operation (Opcode 000011, Mnemonic = JMP)
The next instruction should be loaded from the address in the second word of the instruction added (unsigned) to the contents of register specified by operand1.
For this instruction Operand2, Destination, and Function bits are not used.

Example of JMP operation:

JMP R1(0x00001234)

000011 00001 00000 00000 0000 0000000
00000000 00000000 00010010 00110100

Jump If Zero Operation (Opcode 000100, Mnemonic JZ)
If the register specified in operand2 (written first in assembler) is zero, jump to the address that is the sum of the base address (in the second word of this instruction) and the register specified in operand1, otherwise (if register value of operand2 is non-zero) execute the next instruction in order.

Example of JZ operation

JZ R1, R2(0x00001234)
(note that R1 = operand2, R2 = operand1) in assembler syntax

000100 00010 00001 00000 0000 0000000
00000000 00000000 00010010 00110100

Jump If Not Zero Operation (Opcode 000101, Mnemonic JNZ)
If the register specified in operand2 is not zero, jump to the address that is the sum of the base address(in the second word of this instruction) and the offset which is the value of operand1, otherwise (that is if operand1 holds a zero) execution the next instruction in order.

Example of JNZ operation

JNZ R1, R2(0x00001234)
(note that R1 = operand2, R2 = operand1 in assembler syntax)

000101 00010 00001 00000 0000 0000000
00000000 00000000 00010010 00110100