ST. CLOUD STATE
UNIVERSITY

CSCI 220: Computer Architecture-I
Instructor: Pranava K. Jha

Exercises in MIPS

Q. 1. [15 points] Convert the following MIPS assembly language code into machine code. Write the instructions in hexadecimal.

```
add  $t0, $s0, $s1
lw   $t0, 0x20($t7)
addi $s0, $0, -10
```

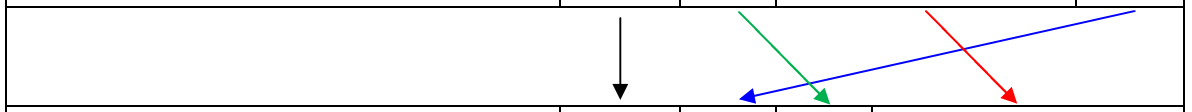
First instruction: `add $t0, $s0, $s1`

Note: This instruction computes the sum of the contents of (source) registers $s0 and $s1 and puts the result in the (target) register $t0.

Basic format of the assembly instruction:	add	$t0	$s0	$s1	shamt	funct
Binary representation of various fields:	000000	01000	10000	10001	00000	100000
	↓	↘	↘	↓	↓	↓
Binary representation in the machine format:	000000	10000	10001	01000	00000	100000
The resulting 32-bit string:	00000010000100010100000000100000					
The corresponding hexadecimal string:	02114020					
`Final answer:`	`0x02114020`					

Second instruction: `lw  $t0, 0x20($t7)`

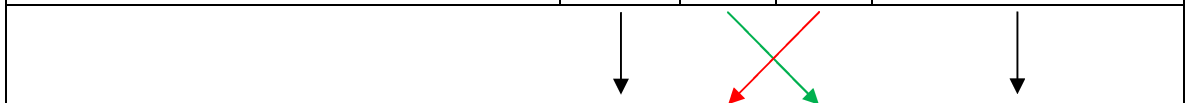
In this case, register $t0 is to be loaded with the word that is located at the memory location whose base address is in register $t7, the offset being 0020 (hexadecimal).

Basic format of the assembly instruction:	lw	$t0	Offset: 0x0020	$t7
Binary representation of various fields:	100011	01000	0000000000100000	01111
				
Binary representation in the machine format:	100011	01111	01000	0000000000100000
The resulting 32-bit string:	10001101111010000000000000100000			
The corresponding hexadecimal string:	8de80020			
`Final answer:`	`0x8de80020`			

Third instruction: `addi $s0, $0, -10`

In this case, sum of the content of register $0 (that is the constant value 0) and the constant value (-10) will appear in register $s0.

Note that the two's-complement representation of (-10) using 16 bits is 1111 1111 1111 0110.

Basic format of the assembly instruction:	addi	$s0	$0	-10
Binary representation of various fields:	001000	10000	00000	1111 1111 1111 0110
				
Binary representation in the machine format:	001000	00000	10000	1111 1111 1111 0110
The resulting 32-bit string:	00100000000100001111111111110110			
The corresponding hexadecimal string:	2010fff6			
`Final answer:`	`0x2010fff6`			

Q. 2. [20 points] Convert the following program from machine language into MIPS assembly language. The numbers on the left are addresses in memory, and those on the right are instructions at the respective addresses.

Address	Instruction
0x00400000	0x20080000
0x00400004	0x20090001
0x00400008	0x0089502a
0x0040000c	0x15400003
0x00400010	0x01094020
0x00400014	0x21290002
0x00400018	0x08100002
0x0040001c	0x01001020

First instruction: The binary representation of 0x20080000 is given by 0010 0000 0000 1000 0000 0000 0000 0000.

The group of leftmost six bits (001000) corresponds to the op code `addi`.
Following is the method of attack for getting the corresponding assembly instruction.

Binary representation in the machine format:	001000	00000	01000	0000000000000000
	↓	↘ ↙	↓	
Binary representation in the assembly format:	001000	01000	00000	0000000000000000
Assembly instruction	addi	$t0	$0	0
Final answer:	`addi $t0, $0, 0`			

Remark: The 16-bit string 0000000000000000 denotes decimal 0 in two's-complement representation.

Second instruction: The binary representation of 0x20090001 is given by 0010 0000 0000 1001 0000 0000 0000 0001.

Again, the group of leftmost six bits (001000) corresponds to the op code `addi`.

Binary representation in the machine format:	001000	00000	01001	0000000000000001
	↓	↘ ↙	↓	
Binary representation in the assembly format:	001000	01001	00000	0000000000000001
Assembly instruction	addi	$t1	$0	1
Final answer:	`addi $t1, $0, 1`			

Third instruction: The binary representation of 0x0089502a is given by 0000 0000 1000 1001 0101 0000 0010 1010.

The group of leftmost six bits (i.e., 000000) indicates that it is an R-type instruction that, in turn, suggests that we must examine the group of last six bits (i.e., 101010) for `funct`. It turns out that 101010 corresponds to `slt`.

Binary representation in the machine format:	000000	00100	01001	01010	00000	101010
Binary representation in the assembly format:	000000	01010	00100	01001	00000	101010
Assembly instruction	slt	$t2	$a0	$t1	--	--
Final answer:	`slt $t2, $a0, $t1`					

Fourth instruction: The binary representation of 0x15400003 is given by 0001 0101 0100 0000 0000 0000 0000 0011.

The group of leftmost six bits (i.e., 000101) corresponds to the op code `bne`.

	bne	Rs	Rt	Imm
Binary representation in the machine format:	000101	01010	00000	0000000000000011

At this point, recall the assembly language format of this instruction and its correspondence with the machine language format:

```
bne Rs, Rt, Label #If [Rs]!=[Rt] then PC = PC + se Imm << 2
```

In this particular case, Imm << 2 in binary is given by 0000000000001100 that is equal to 0x0000000c. Further, PC currently holds 0x00400010 (address of the next instruction). Accordingly, `Label` itself will correspond to the memory address 0x00400010 plus 0x0000000c that is 0x0040001c. Let L2 denote the address 0x0040001c.

Binary representation in the assembly format:	000101	01010	00000	0000000000011100
Assembly instruction	bne	$t2	$0	L2
Final answer:	`bne $t2, $0, L2`			

Fifth instruction: The binary representation of `0x01094020` is given by 0000 0001 0000 1001 0100 0000 0010 0000.

The group of leftmost six bits (i.e., 000000) indicates that it is an R-type instruction that, in turn, suggests that we must examine the group of last six bits (i.e., 100000) for `funct`. It turns out that 100000 corresponds to `add`.

Binary representation in the machine format:	000000	01000	01001	01000	00000	100000
Binary representation in the assembly format:	000000	01000	01000	01001	00000	100000
Assembly instruction	add	$t0	$t0	$t1	--	--
Final answer:	`add $t0, $t0, $t1`					

Sixth instruction: The binary representation of `0x21290002` is given by 0010 0001 0010 1001 0000 0000 0000 0010.

The group of leftmost six bits (001000) corresponds to the op code `addi`.

Binary representation in the machine format:	001000	01001	01001	0000000000000010
Binary representation in the assembly format:	001000	01001	01001	0000000000000010
Assembly instruction	addi	$t1	$t1	2
Final answer:	`addi $t1, $t1, 2`			

Seventh instruction: The binary representation of `0x08100002` is given by 0000 1000 0001 0000 0000 0000 0000 0010.

The group of leftmost six bits (i.e., 000010) corresponds to the op code `j` for jump.

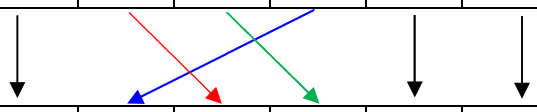
	j	Imm
Binary representation in the machine format:	000010	00000100000000000000000010

Load the PC with the address formed by concatenating the first four bits of the current PC with the value in the immediate field padded at the rightmost end by two 0s. Now, PC currently holds 0x0040001c (address of the next instruction). Accordingly, jump will be made to the memory address 00000000010000000000000000001000 that is 0x00400008. Let `L1` denote this address.

Final answer:	`j L1`

Eighth instruction: The binary representation of `0x01001020` is given by 0000 0001 0000 0000 0001 0000 0010 0000.

The group of leftmost six bits (i.e., 000000) indicates that it is an R-type instruction that, in turn, suggests that we must examine the group of last six bits (i.e., 100000) for `funct`. It turns out that 100000 corresponds to `add`.

Binary representation in the machine format:	000000	01000	00000	00010	00000	100000
						
Binary representation in the assembly format:	000000	00010	01000	00000	00000	100000
Assembly instruction	add	$v0	$t0	$0	--	--
Final answer:	`add $t0, $t0, $t1`					

Here is the complete assembly code corresponding to the given machine code:

```
     addi $t0, $0, 0
     addi $t1, $0, 1
     slt $t2, $a0, $t1
L1:  bne $t2, $0, L2
     add $t0, $t0, $t1
     addi $t1, $t1, 2
     j L1
L2:  add $t0, $t0, $t1
```

Q. 3. [20 points] Implement the following high-level code segments using the `slt` instruction. Assume that the integer variables `g` and `h` are in registers `$s0` and `$s1`, respectively.

```
(a)  if (g > h)
         g = g + h;
       else
         g = g – h;

(b)  if (g >= h)
         g = g + 1;
       else
         h = h – 1;
```

```
(a)
                # $s0 = g, $s1 = h
                slt $t0, $s1, $s0   # if h < g, $t0 = 1
                beq $t0, $0, else   # if $t0 == 0, do else
                add $s0, $s0, $s1   # g = g + h
                j done              # jump past else block
    else:       sub $s0, $s0, $s1   # g = g - h
    done:


(b)
                # $s0 = g, $s1 = h
                slt $t0, $s0, $s1   # if g < h, $t0 = 1
                bne $t0, $0, else   # if $t0 != 0, do else
                addi $s0, $s0, 1    # g = g + 1
                j done              # jump past else block
    else:       addi $s1, $s1, -1   # h = h - 1
    done:
```

———————o———————