ECE 366
Project 2: ISA Design

Group 6

Karim Eltahawy
Henry Sampson
Erick Gonzalez

Part A)

1. The name of our architecture is called Small Instruction Count (SIC). The overall philosophy is to utilize program 1 and 2 with 7-bit instruction type. Some goals we had included using our memory space efficiently through our limited number of registers and small instruction count. We achieved by taking advantage of our memory space.
2.

Instruction	**opcode**	**Rx**	**Ry**	**imm**	**Range**	**Function**
lw	000	Rx: _ _	Ry: _ _		Rx & Ry: [R0, R1, R2, R3]	Load word
sw	001	Rx: _ _	Ry: _ _		Rx & Ry: [R0, R1, R2, R3]	Store word
add	100	Rx: _ _	Ry: _ _		Rx: [R0, R1] Ry: [R0, R1, R2, R3]	Add
addi	100	Rx: _ _		_ _	Rx: [R2, R3] imm: [0, 3]	Add w/ imm
xor	110	Rx: _ _	Ry: _ _		Rx: [R0, R1] Ry: [R0, R1, R2, R3]	XOR
and	110	Rx: _ _	Ry: _ _		Rx: [R2, R3] Ry: [R0, R1, R2, R3]	AND
Init	101	Rx: _ _		_ _	Rx: [R0, R1] imm: [-1, 1]	Initialize value
sll	101	Rx: _ _		_ _	Rx: [R2, R3] imm: [0,3]	Bit shift left
j	1110			_ _ _	imm: [0, 7]	jump by (-1)* 2^(imm)
sub	010	Rx: _ _	Ry: _ _		Rx: [R0, R1] Ry: [R0, R1, R2, R3]	Subtract
subi	010	Rx: _ _		_ _	Rx: [R2, R3] imm: [0, 3]	Subtract w/ imm
sltR0	011	Rx: _ _	Ry: _ _		Rx: [R0, R1] Ry: [R0, R1, R2, R3]	If Rx < Ry, R0 == 0, Otherwise, R0 = 1
seqR0	011	Rx: _ _	Ry: _ _		Rx: [R2, R3] Ry: [R0, R1, R2, R3]	If Rx = Ry, R0 == -1, Otherwise, R0 = 1
beqR0	1111			_ _ _	imm: [0,6]	If R0 == 0, PC += 2^(imm)

						If R0 = -1, PC -= 2^(imm) Otherwise, PC++ Note: imm =/= 111
Halt	1111111					Stops the processor

Example: add $r0, $r2 which adds $r0 with $r2 and stores result back to $r0.

3. We have four registers in our ISA design and one of our registers acts as our instruction identifier by combining it without 3 or 4-bit opcode.
4. We used two branch instructions: j and beqR0. The j, which is jump, instruction takes an immediate value ranged from [0,7] and changes PC as so: PC = PC – 2^(imm).

The beqR0 instruction depends on the R0 value so that if R0 = 0, PC = PC + 2^(imm), if R0 = -1, PC = PC – 2^(imm), if R0 = otherwise then PC = PC + 1. The range of imm is from [0,6] and cannot be equal to 7 as it would interfere with the Halt instruction.

Example: j 6 would jump back 2^6 lines by changing PC (PC = PC – 2^6).

5. The way we address the data memory is through load word and store word instructions. Our first address starts from 0x0000 and ends at 0xFFFF which gives 2^(17) – 1 16-bit memory addresses storing 16-bits of data per address. The addresses are calculated based on the register's data. Example:

Assume $r0 = 0 and $r1 = 1 then use the following instruction: lw $r0, $r1
This would store the value '0' into memory address '1' (0X0001).

Part B)

1. Some unique features that our ISA design is the way we identify our instructions using both the op code and the Rx by combining them. This helped us increase the number of instruction count.
2. Some ways we optimized for our two main goals were that we were able to create enough instructions needed for both programs and took advantage of our large memory space and relied on it to store data, rather than just relying on our registers.
3. If we had one more bit we wouldn't have to worry about limitation on register count and number of instructions. If we had one less bit we would have less registers to use and a decrease in range for immediate values for our I-type/J-type instructions.
4. Our team would meet in person to make it more interactive and we focused on building our ideas on the ISA design and how it would apply in our two programs. We also had member assignments that we would have to prepare before meeting up again. We tried our best so that not one member would do more work.
5. If we had to restart our project afresh, we would update our ISA design and modifying our two programs.

Part C)

1. Assembly Code of Program 1:

```
main:
addi $r2, 1 #achieve the number 6
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
init $r0, 0 #set $r0 to memory location 0x0000
sw $r2, ($r0) # 6 is now in memory location 0x0000
#could make another 6 or copy it later
addi $r2, 1 #achieve the number 17
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
addi $r2, 1
init $r0, 1 #set $r0 to memory location 0x0001
sw $r2, ($r0) #17 is now in memory location 0x0001
init $r0, 0 #set $r0 to memory location 0x0000
addi $r3, 1
addi $r3, 1
add $r0, $r3 #set $r0 to memory locatiion 0x0002
subi $r3, 1 #r3 is now set to 1
subi $r3, 1 #r3 is reset to 0
subi $r2, 1
subi $r2, 1
subi $r2, 1
subi $r2, 1
subi $r2, 1
subi $r2, 1
subi $r2, 1
sw $r2, ($r0) #10 is now in memory location 0x0002
init $r0, 0 #set $r0 to memory location 0x0000
lw $r2, ($r0) #pull 6 back out from memory
loop2:
loop:
add $r1, $r2
addi $r3, 1
sub $r0, $r3 #decrement the counter by 1 then 2 then 3. . .
beqR0 2 #if counter is 0 leave the loop
j -4
done:
subi $r3, 1
subi $r3, 1
subi $r3, 1
subi $r3, 1
subi $r3, 1
subi $r3, 1 #reset r3 back to 0
init $r0, 0
addi $r3, 1
addi $r3, 1
addi $r3, 1
add $r0, $r3
sw $r1, ($r0) #store product of 6 times larger number in 0x0003
init $r0, 0
subi $r3, 1
add $r0, $r3 #set r0 to memory location 0x0002
lw $r1, ($r0) #$r1 is now equal to 10
subi $r3, 1
sub $r1, $r3
sw $r1, ($r0) #decrement the  counter and put it back in memory location 0x0002
sltR0 $r1, $r3 #check to see if counter is less than 1
subi $r3, 1
init $r0, 0
addi $r3, 1
addi $r3, 1
addi $r3, 1
add $r0, $r3
lw $r2, ($r0) #load product of 6 times larger number in 0x0003
init $r0, 0
subi $r3, 1
subi $r3, 1
subi $r3, 1 #reset r3
addi $r3, 1
addi $r3, 1
addi $r3, 1
addi $r3, 1
add $r0, $r3 #memory 0x0004 has data value 0
subi $r3, 1
subi $r3, 1
subi $r3, 1
subi $r3, 1
lw $r1, ($r0)
init $r0, 0
lw $r0, ($r0)
beqR0 2
j -25
done2:
init $r0, 0
addi $r3, 1
addi $r3, 1
addi $r3, 1
add $r0, $r3
subi $r3, 1
subi $r3, 1
subi $r3, 1
lw $r1, ($r0) #load in the big number from 0x0003
init $r0, 0
addi $r3, 1
add $r0, $r3
subi $r3, 1
lw $r2, ($r0) #load in 17 from 0x0001
loop3: #mod 17
sub $r1, $r2
sltR0 $r1, $r2
beqR0 2
j -3
else:
sw $r1, ($r0) #answer is in memory location 0x0001
```

Assembly Code for Program 2:

```
        # Need to figure out how to deal with user inputs but we'll assume they are taking up the first
# 23 spaces of memory after 0x0001: (T (0x0002 or 2), best_matching_score (0x0003 or 3),
best_matching_count (0x0004 or 4),
# Pattern_Array (0x0005 - 0x0019 or 5-24)
#Initialize best_matching_score and best_matching_count to -1 as initialization each:
init $r0, 1 # set r0 = 1 temporarily
init $r1, 1 # set r1 = 1
add $r1, $r1 # r1 = 1 + 1 = 2
add $r1, $r0 # r1 = 2 + 1 = 3
init $r0, -1 # set r0 = -1
sw $r0, $r1 # set $r0 val of -1 into mem loc 3 which is 0x0003
init $r0, 1 # set r0 = 1 temporarily
add $r1, $r0 # r1 = 3 + 1 = 4
init $r0, -1 # set r0 back to -1
sw $r0, $r1 # set $r0 val of -1 into mem loc 4 which is 0x0004
init $r0, 0 # reset r0 back to 0
init $r1, 0 # reset r1 back to 0
#Initialize mem loc 0x0000 = 0 and mem loc 0x0001 = 1
sw $r0, $r0 # r0 is already = 0, set $r0 val of 0 into mem loc 0 which is 0x0000
init $r1, 1 # set r1 to 1
sw $r1, $r1 # set $r1 val of 1 into mem loc 1 which is 0x0001
init $r1, 0 # reset r1 back to 0, r0 is still 0 at this poin
#Initializing r2 and r3 to 0 so that all registers = 0 initially
lw $r2, $r0 # r0 is 0 at this point, r2 now = 0
lw $r3, $r0 # r0 is 0 at this point, r3 now = 0
#Initialize user input of T and Pattern_Array (HARD CODED)
addi $r2, 3 # $r2 = 0 + 3 = 3
sll $r2, 2 # $r2 = 3 * 2^2 = 3 * 4 = 12
addi $r3, 2 # $r3 = 0 + 2 = 2
sw $r2, $r3 # store r2 val of 12 into mem loc 2 or 0x0002
```

```
addi $r3, 3 # r3 = 2 + 3 = 5
init r0, 0 # set r0 = 0
lw $r2, $r0 # load 0 into r2 from mem loc 0 or 0x0000
sw $r2, $r3 # store 0 into mem loc 5
addi $r2, 1 # r2 = 0 + 1 = 1
addi $r3, 1 # r3 = 5 + 1 = 6
sw $r2, $r3 # store 1 into mem loc 6
addi $r2, 1 # r2 = 1 + 1 = 2
addi $r3, 1 # r3 = 6 + 1 = 7
sw $r2, $r3 # store 2 into mem loc 7
addi $r2, 1 # r2 = 2 + 1 = 3
addi $r3, 1 # r3 = 7 + 1 = 8
sw $r2, $r3 # store 3 into mem loc 8
addi $r2, 1 # r2 = 3 + 1 = 4
addi $r3, 1 # r3 = 8 + 1 = 9
sw $r2, $r3 # store 4 into mem loc 9
addi $r2, 1 # r2 = 4 + 1 = 5
addi $r3, 1 # r3 = 9 + 1 = 10
sw $r2, $r3 # store 5 into mem loc 10
addi $r2, 1 # r2 = 5 + 1 = 6
addi $r3, 1 # r3 = 10 + 1 = 11
sw $r2, $r3 # store 6 into mem loc 11
addi $r2, 1 # r2 = 6 + 1 = 7
addi $r3, 1 # r3 = 11 + 1 = 12
sw $r2, $r3 # store 7 into mem loc 12
addi $r2, 1 # r2 = 7 + 1 = 8
addi $r3, 1 # r3 = 12 + 1 = 13
sw $r2, $r3 # store 8 into mem loc 13
addi $r2, 1 # r2 = 8 + 1 = 9
addi $r3, 1 # r3 = 13 + 1 = 14
sw $r2, $r3 # store 9 into mem loc 14
addi $r2, 1 # r2 = 9 + 1 = 10
addi $r3, 1 # r3 = 14 + 1 = 15
sw $r2, $r3 # store 10 into mem loc 15
addi $r2, 1 # r2 = 10 + 1 = 11
addi $r3, 1 # r3 = 15 + 1 = 16
sw $r2, $r3 # store 11 into mem loc 16
addi $r2, 1 # r2 = 11 + 1 = 12
addi $r3, 1 # r3 = 16 + 1 = 17
sw $r2, $r3 # store 13 into mem loc 17
addi $r2, 1 # r2 = 12 + 1 = 13
addi $r3, 1 # r3 = 17 + 1 = 18
sw $r2, $r3 # store 14 into mem loc 18
addi $r2, 1 # r2 = 13 + 1 = 14
addi $r3, 1 # r3 = 18 + 1 = 19
sw $r2, $r3 # store 15 into mem loc 19
addi $r2, 1 # r2 = 14 + 1 = 15
addi $r3, 1 # r3 = 19 + 1 = 20
```

```
sw $r2, $r3 # store 16 into mem loc 20
addi $r2, 1 # r2 = 15 + 1 = 16
addi $r3, 1 # r3 = 20 + 1 = 21
sw $r2, $r3 # store 17 into mem loc 21
addi $r2, 1 # r2 = 16 + 1 = 17
addi $r3, 1 # r3 = 21 + 1 = 22
sw $r2, $r3 # store 18 into mem loc 22
addi $r2, 1 # r2 = 17 + 1 = 18
addi $r3, 1 # r3 = 22 + 1 = 23
sw $r2, $r3 # store 19 into mem loc 23
addi $r2, 1 # r2 = 18 + 1 = 19
addi $r3, 1 # r3 = 23 + 1 = 24
sw $r2, $r3 # store 20 into mem loc 24
init r0, 0 # reset r0 = 0
init r1, 0 # reset r1 = 0
lw $r2, $r0 # reset r2 = 0
lw $r3, $r0 # reset r3 = 0
#Storing 20 in memory. This will help for our array loop
addi $r2, 1 # registers are initially at 0, so r2 = 0 + 1 = 1
sll $r2, 3 # r2 = r2 << 3 = 8
sll $r2, 1 # r2 = r2 << 1 = 16
addi $r2, 3 # r2 = 16 + 3 = 19
addi $r2, 1 # r2 = 19 + 1 = 20
addi $r3, 3 # r3 = 0 + 3 = 3
addi $r3, 1 # r3 = 3 + 1 = 4
addi $r3, 2 # r3 = 4 + 2 = 6
add $r0, $r2 # r0 = 0 + 20 = 20
add $r0, $r3 # r0 = 20 + 6 = 26, this is for the mem loc
sw $r2, $r0 # store $r2 val of 20 into mem loc 26 which is 0x001A
#At this point, $r3 = 6, $r2 = 20 and $r0 = 26
#Storing 32 in memory for calculating HD later
addi $r2, 3 # r2 = 20 + 3 = 23
addi $r2, 3 # r2 = 23 + 3 = 26
addi $r2, 1 # r2 = 26 + 1 = 27
add $r0, $r3 # r0 = 26 + 6 = 32
sw $r0, $r2 # store $r0 val of 32 into mem loc 27 which is 0x001B
#At this point, $r3 = 6, $r2 = 27, $r0 = 32
#Set r2 = 20 and r3 = -1 for array loop initialization:
#r2 = 20:
subi $r2, 3 # r2 = 27 - 3 = 24
subi $r2, 3 # r2 = 24 - 3 = 21
subi $r2, 1 # r2 = 21 - 1 = 20
# $r3 = -1
init $r1, 0 # set r1 = 0
lw $r3, $r1 # load val 0 from mem location 0 or 0x0000, r3 = 0
subi $r3, 1 # r3 = 0 - 1 = -1
init $r1, 0 # reset r1 to 0
init $r0, 0 # set r0 to 0
```

```
#At this point, $r3 = -1, $r2 = 20, $r1 = 0, $r0 = 0
#r2 = array decrementer, r0 = r2 required beg. of loop, r3 = array element #, initially -1
a_loop: # array loop
        init $r0, 0 # always sets r0 to 0 first
        add $r0, $r2 # since r0 is always 0 beforehand, r0 = r2 everytime
        beqR0 6 #NOTE: this is NOT negative, so it jumps forward 2^6 to leave a_loop
        subi $r2, 1 # decrement r2
        addi $r3, 1 # increment r3
        lw $r1, $r3 # load array value at mem loc 'r3' into r1; #load T val in 0x0002 into r0
        init $r0, 1 # set r0 = 1
        add $r0, $r0 # r0 = 1 + 1 = 2
        lw $r0, $r0 # load T from mem loc 0x0002 into r0, r0 now = T
        xor $r0, $r1 # XOR of r0 and r1, store result into r0
        init $r0, 0 # set r0 = 0
        sw $r1, $r0 # store XOR result into mem loc 0x0000 temporarily, overwriting '0'
        init $r0, 1 # set r0 = 1
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        add $r0, $r0 # r0 = 4 + 4 = 8
        add $r0, $r0 # r0 = 8 + 8 = 16
        add $r0, $r0 # r0 = 16 + 16 = 32
        init $r1, 1 # set r1 = 1, used to decrement r0 by 1
        sub $r0, $r1 # r0 = 32 - 1 = 31
        add $r1, $r1 # r1 = 1 + 1 = 2
        add $r1, $r1 # r1 = 2 + 2 = 4
        add $r1, $r1 # r1 = 4 + 4 = 8
        add $r1, $r1 # r1 = 8 + 8 = 16
        add $r1, $r1 # r1 = 16 + 16 = 32
        add $r1, $r1 # r1 = 32 + 32 = 64
        sw $r0, $r1 # store r0 val of 31 into mem loc 64; # store array element # in r3 into mem loc 28 or
0x001C
        add $r1, $r1 # r1 = 1 + 1 = 2
        add $r1, $r1 # r1 = 2 + 2 = 4
        init $r0, 1 # set r0 = 1
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        add $r0, $r0 # r0 = 4 + 4 = 8
        add $r0, $r0 # r0 = 8 + 8 = 16
        add $r0, $r0 # r0 = 16 + 16 = 32
        sub $r0, $r1 # r0 = 32 - 4 = 28
        sw $r3, r0 # store r3 = array element # in mem loc 28
        lw $r3, $r1 # load val 1 into r3 from mem location 0x0001
        init $r0, 0 # set r0 = 0 for HD
        init $r1, 1 # set r1 = 1
        add $r1, $r1 # r1 = 1 + 1 = 2
        add $r1, $r1 # r1 = 2 + 2 = 4
        add $r1, $r1 # r1 = 4 + 4 = 8
        add $r1, $r1 # r1 = 8 + 8 = 16
        add $r1, $r1 # r1 = 16 + 16 = 32
        sw $r0, $r1 # store HD into mom loc 32
        init $r1, 1 # set r1 = 1 for masking
m_loop: #masking loop, DO NOT use init instr with r1
        init $r0, 1 # set r0 = 1
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        add $r0, $r0 # r0 = 4 + 4 = 8
        add $r0, $r0 # r0 = 8 + 8 = 16
        add $r0, $r0 # r0 = 16 + 16 = 32
        add $r0, $r0 # r0 = 32 + 32 = 64
        lw $r0, $r0 # load r0 val of 31 init stored in mem loc 64 back into r0
        beqR0 5 #NOTE that is NOT negative and jumps forward 2^5 to leave m_loop
        init $r0, 0 # set r0 = 0 so r3 can access mem loc 0x0000/XOR result
        lw $r3, $r0 # r3 = XOR result which is stored in mem loc 0x0000
        and $r3, $r1 # AND the XOR result (r3) with masking value (r1)
        add $r1, $r1 # shift by 1 for mask value
        init $r0, 0 # set r0 = 0 to compare with AND result
        seqR0 $r3, $r0 # check if $r0 = $r3, if so then r0 = -1
        beqR0 -4 #jumps back -2^4
        init $r0, 1 # set r0 = 1
        lw $r3, $r0 # load val 1 into r3 from mem location 0x0001
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        add $r0, $r0 # r0 = 4 + 4 = 8
        add $r0, $r0 # r0 = 8 + 8 = 16
        add $r0, $r0 # r0 = 16 + 16 = 32
        lw $r0, $r0 # load HD into r0 from mem loc 32
        add $r0, $r3 # HD + 1 (r3)
        sll $r3, 3 # r3 = 1 << 3 = 8
        sll $r3, 2 # r3 = 8 << 2 = 32
        sw $r0, $r3 # store incremented HD back to mem loc 32
        j 7 #jumps back 2^7 to reach a_loop
continue: #continue a_loop
        init $r0, 0 # set r0 = 0
        sw $r0, $r0 # set mem loc 0x0000 back to '0' overwriting XOR result
        # remember that HD is stored in mem loc 32
        init $r0, 1 # set r0 = 1
        addi $r3, 3 # r3 = 1 + 3 = 4
        addi $r3, 1 # r3 = 4 + 1 = 5
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        add $r0, $r0 # r0 = 4 + 4 = 8
        add $r0, $r0 # r0 = 8 + 8 = 16
        add $r0, $r0 # r0 = 16 + 16 = 32
        init $r1, 0 # set r1 = 0
        add $r1, $r0 # r1 = 0 + 32 = 32
        lw $r0, $r0 # load HD into r0 from mem loc 32
        sub $r1, $r0 # r1 = 32 - HD
        sw $r1, $r0 # store x (which is 32 - HD) into mem loc 32, overwriting HD
        init $r0, 0 # set r0 = 0
        lw $r3, $r0 # load val '0' into r3
        addi $r3, 3 # r3 = 0 + 3 = 3
        init $r0, 1 # set r0 = 1 to avoid next cases initially
        lw $r3, $r3 # load best_matching_score val stored in mem loc 3 into r3
        sltR0 $r1, $r3 # if x < best_matching_score then R0 == 0
        beqR0 6 #jumps back 2^6 to a_loop;
        seqR0 $r1, $r3 # if x = best_matching_score then R0 == -1
        #if R0 is -1 at this point, increment the best_matching_count
        init $r0, 0 # set r0 = 0
        lw $r3, $r0 # load val '0' into r3
        addi $r3, 3 # r3 = 0 + 3 = 3
        addi $r3, 1 # r3 = 1 + 3 = 4
        lw $r3, $r3 # load best_matching_count into r3 from mem loc 4
        addi $r3, 1 # r3 = best_matching_count + 1
        init $r0, 1 # set r0 = 1
        add $r0, $r0 # r0 = 1 + 1 = 2
        add $r0, $r0 # r0 = 2 + 2 = 4
        sw $r3, $r0 # store incremented best_matching_count back to mem loc 4
        beqR0 6 #jumps back 2^6 to a_loop;
        #if the first two cases fail, then it does not branch
        init $r0, 1 # set r0 = 1
        lw $r3, $r0 # load val '0' into r3
        addi $r3, 2 # r3 = 1 + 2 = 3
        lw $r0, $r3 # reset best_matching_count to 1 at mem loc 3; # x is in mem loc 32
        init $r0, 0 # set r0 = 0
        lw $r3, $r0 # load val '0' into r3
        addi $r3, 3 # r3 = 0 + 3 = 3
        add $r0, $r3 # r3 = 0 + 3 = 3
        subi $r3, 2 # r3 = 3 - 2 = 1
        sll $r3, 3 # r3 = 1 << 3 = 8
        sll $r3, 2 # r3 = 8 << 2 = 32
        lw $r3, $r3 # x stored in r3 from mem loc 32
        lw $r3, $r0 # x replaces old best_matching_count with new one at mem loc 3
        j 7 # jump back by 2^7 to a_loop
out:
        Halt
```

2. Machine Code for Program 1:

```
01001001 #achieve the number 6
01001001
01001001
01001001
01001001
01001001
```

```
01010000 #set $r0 to memory location 0x0000
00011000 #6 is now in memory location 0x0000
01001001 #achieve the number 17
01001001
01001001
01001001
01001001
01001001
01001001
01001001
01001001
01001001
01001001
01010001 #set $r0 to memory location 0x0001
00011000 #17 is now in memory location 0x0001
01010000 #set $r0 to memory location 0x0000
01001101
01001101
01000011 #set $r0 to memory location 0x0002
00101101 #r3 is now set to 1
00101101 #r3 is reset to 0
00101001
00101001
00101001
00101001
00101001
00101001
00101001
00011000 #10 is now in memory location 0x0002
01010000 #set $r0 to memory location 0x0000
00001000 #pull 6 back out from memory
01000110
01001101
00100011
01111010 #decrement the counter by 1 then 2 then 3. . .
01110011 #if counter is 0 leave the loop
00101101
00101101
00101101
00101101
00101101
00101101 #reset r3 back to 0
01010000
01001101
01001101
01001101
01000011
00010100 #store product of 6 times larger numbrer in 0x0003
```

```
01010000
00101101
01000011 #set r0 to memory location 0x0002
00000100 #$r1 is now equal to 10
00101101
00100111
00010100 #decrement the counter and put it back in memory location 0x0002
00110111 #check to see if counter is less than 1
00101101
01010000
01001101
01001101
01001101
01000011
00001000 #load product of 6 times larger number in 0x0003
01011000
00101101
00101101
00101101 #reset r3
01001101
01001101
01001101
01001101
01000011 #memory 0x0004 has data value 0
00101101
00101101
00101101
00101101
00000100
01010000
01111010
01110110
01010000
01001101
01001101
01001101
01000011
00101101
00101101
00101101
00000100 #load in the big number from 0x0003
01010100
01000000
01000011
00101101
00001000 #load in 17 from 0x0001
 #mod 17
00100110
```

```
00110110
01111010
01110010
00010100 #answer is in memory location 0x0001
```

Machine Code for Program 2:

```
01010001 # set r0 = 1 temporarily
01010101 # set r1 = 1
01000101 # r1 = 1 + 1 = 2
01000100 # r1 = 2 + 1 = 3
01010011 # set r0 = -1
00010001 # set $r0 val of -1 into mem loc 3 which is 0x0003
01010001 # set r0 = 1 temporarily
01000100 # r1 = 3 + 1 = 4
01010011 # set r0 back to -1
00010001 # set $r0 val of -1 into mem loc 4 which is 0x0004
01010000 # reset r0 back to 0
01010100 # reset r1 back to 0
00010000 # r0 is already = 0, set $r0 val of 0 into mem loc 0 which is 0x0000
01010101 # set r1 to 1
00010101 # set $r1 val of 1 into mem loc 1 which is 0x0001
01010100 # reset r1 back to 0, r0 is still 0 at this point
00001000 # r0 is 0 at this point, r2 now = 0
00001100 # r0 is 0 at this point, r3 now = 0
01001011 # $r2 = 0 + 3 = 3
01011010 # $r2 = 3 * 2^2 = 3 * 4 = 12
01001101 # $r3 = 0 + 2 = 2
00011011 # store r2 val of 12 into mem loc 2 or 0x0002
01001111 # r3 = 2 + 3 = 5
01010000 # set r0 = 0
00001000 # load 0 into r2 from mem loc 0 or 0x0000
00011011 # store 0 into mem loc 5
01001001 # r2 = 0 + 1 = 1
01001101 # r3 = 5 + 1 = 6
00011011 # store 1 into mem loc 6
01001001 # r2 = 1 + 1 = 2
01001101 # r3 = 6 + 1 = 7
00011011 # store 2 into mem loc 7
01001001 # r2 = 2 + 1 = 3
01001101 # r3 = 7 + 1 = 8
00011011 # store 3 into mem loc 8
01001001 # r2 = 3 + 1 = 4
01001101 # r3 = 8 + 1 = 9
00011011 # store 4 into mem loc 9
01001001 # r2 = 4 + 1 = 5
01001101 # r3 = 9 + 1 = 10
00011011 # store 5 into mem loc 10
```

```
01000101 # r2 = 5 + 1 = 6
01001101 # r3 = 10 + 1 = 11
00011011 # store 6 into mem loc 11
01001001 # r2 = 6 + 1 = 7
01001101 # r3 = 11 + 1 = 12
00011011 # store 7 into mem loc 12
01001001 # r2 = 7 + 1 = 8
01001101 # r3 = 12 + 1 = 13
00011011 # store 8 into mem loc 13
01001001 # r2 = 8 + 1 = 9
01001101 # r3 = 13 + 1 = 14
00011011 # store 9 into mem loc 14
01001001 # r2 = 9 + 1 = 10
01001101 # r3 = 14 + 1 = 15
00011011 # store 10 into mem loc 15
01001001 # r2 = 10 + 1 = 11
01001101 # r3 = 15 + 1 = 16
00011011 # store 11 into mem loc 16
01001001 # r2 = 11 + 1 = 12
01001101 # r3 = 16 + 1 = 17
00011011 # store 13 into mem loc 17
01001001 # r2 = 12 + 1 = 13
01001101 # r3 = 17 + 1 = 18
00011011 # store 14 into mem loc 18
01001001 # r2 = 13 + 1 = 14
01001101 # r3 = 18 + 1 = 19
00011011 # store 15 into mem loc 19
01001001 # r2 = 14 + 1 = 15
01001101 # r3 = 19 + 1 = 20
00011011 # store 16 into mem loc 20
01001001 # r2 = 15 + 1 = 16
01001101 # r3 = 20 + 1 = 21
00011011 # store 17 into mem loc 21
01001001 # r2 = 16 + 1 = 17
01001101 # r3 = 21 + 1 = 22
00011011 # store 18 into mem loc 22
01001001 # r2 = 17 + 1 = 18
01001101 # r3 = 22 + 1 = 23
00011011 # store 19 into mem loc 23
01001001 # r2 = 18 + 1 = 19
01001101 # r3 = 23 + 1 = 24
00011011 # store 20 into mem loc 24
01010000 # reset r0 = 0
01010100 # reset r1 = 0
00001000 # reset r2 = 0
00001100 # reset r3 = 0
01001001 # registers are initially at 0, so r2 = 0 + 1 = 1
01011011 # r2 = r2 << 3 = 8
```

```
01011001 # r2 = r2 << 1 = 16
01001011 # r2 = 16 + 3 = 19
01001001 # r2 = 19 + 1 = 20
01001111 # r3 = 0 + 3 = 3
01001101 # r3 = 3 + 1 = 4
01001110 # r3 = 4 + 2 = 6
01000010 # r0 = 0 + 20 = 20
01000011 # r0 = 20 + 6 = 26, this is for the mem loc
00011000 # store $r2 val of 20 into mem loc 26 which is 0x001A
01001011 # r2 = 20 + 3 = 23
01001011 # r2 = 23 + 3 = 26
01001001 # r2 = 26 + 1 = 27
01000011 # r0 = 26 + 6 = 32
00010010 # store $r0 val of 32 into mem loc 27 which is 0x001B
00101011 # r2 = 27 - 3 = 24
00101011 # r2 = 24 - 3 = 21
00101001 # r2 = 21 - 1 = 20
01010100 # set r1 = 0
00001101 # load val 0 from mem location 0 or 0x0000, r3 = 0
00101101 # r3 = 0 - 1 = -1
01010100 # reset r1 to 0
01010000 # set r0 to 0
01010000 # always sets r0 to 0 first
01000010 # since r0 is always 0 beforehand, r0 = r2 everytime
01111110 # jump forward 2^6 to leave a_loop
00101001 # decrement r2
01001101 # increment r3
00000111 # load array value at mem loc 'r3' into r1; #load T val in 0x0002 into r0
01010001 # set r0 = 1
01000000 # r0 = 1 + 1 = 2
00000000 # load T from mem loc 0x0002 into r0, r0 now = T
01100001 # XOR of r0 and r1, store result into r0
01010000 # set r0 = 0
00010100 # store XOR result into mem loc 0x0000 temporarily, overwriting '0'
01010001 # set r0 = 1
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
01000000 # r0 = 4 + 4 = 8
01000000 # r0 = 8 + 8 = 16
01000000 # r0 = 16 + 16 = 32
01010101 # set r1 = 1, used to decrement r0 by 1
00100001 # r0 = 32 - 1 = 31
01000101 # r1 = 1 + 1 = 2
01000101 # r1 = 2 + 2 = 4
01000101 # r1 = 4 + 4 = 8
01000101 # r1 = 8 + 8 = 16
01000101 # r1 = 16 + 16 = 32
01000101 # r1 = 32 + 32 = 64
```

```
00010001 # store r0 val of 31 into mem loc 64; # store array element # in r3 into mem loc 28 or 0x001C
01000101 # r1 = 1 + 1 = 2
01000101 # r1 = 2 + 2 = 4
01010001 # set r0 = 1
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
01000000 # r0 = 4 + 4 = 8
01000000 # r0 = 8 + 8 = 16
01000000 # r0 = 16 + 16 = 32
00100001 # r0 = 32 - 4 = 28
00011100 # store r3 = array element # in mem loc 28
00001101 # load val 1 into r3 from mem location 0x0001
01010000 # set r0 = 0 for HD
01010101 # set r1 = 1
01000101 # r1 = 1 + 1 = 2
01000101 # r1 = 2 + 2 = 4
01000101 # r1 = 4 + 4 = 8
01000101 # r1 = 8 + 8 = 16
01000101 # r1 = 16 + 16 = 32
00010001 # store HD into mem loc 32
01010101 # set r1 = 1 for masking
01010001 # set r0 = 1
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
01000000 # r0 = 4 + 4 = 8
01000000 # r0 = 8 + 8 = 16
01000000 # r0 = 16 + 16 = 32
01000000 # r0 = 32 + 32 = 64
00000000 # load r0 val of 31 init stored in mem loc 64 back into r0
01111101 # jump forward 2^5 to leave m_loop
01010000 # set r0 = 0 so r3 can access mem loc 0x0000/XOR result
00001100 # r3 = XOR result which is stored in mem loc 0x0000
01101101 # AND the XOR result (r3) with masking value (r1)
01000101 # shift by 1 for mask value
01010000 # set r0 = 0 to compare with AND result
00111100 # check if $r0 = $r3, if so then r0 = -1
01111100 # jump back -2^4 to m_loop
01010001 # set r0 = 1
00001100 # load val 1 into r3 from mem location 0x0001
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
01000000 # r0 = 4 + 4 = 8
01000000 # r0 = 8 + 8 = 16
01000000 # r0 = 16 + 16 = 32
00000000 # load HD into r0 from mem loc 32
01000011 # HD + 1 (r3)
01011111 # r3 = 1 << 3 = 8
01011110 # r3 = 8 << 2 = 32
```

```
00010011 # store incremented HD back to mem loc 32
01110111 # jumps back 2^7 to reach a_loop
01010000 # set r0 = 0
00010000 # set mem loc 0x0000 back to '0' overwriting XOR result
01010001 # set r0 = 1
01001111 # r3 = 1 + 3 = 4
01001101 # r3 = 4 + 1 = 5
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
01000000 # r0 = 4 + 4 = 8
01000000 # r0 = 8 + 8 = 16
01000000 # r0 = 16 + 16 = 32
01010100 # set r1 = 0
01000100 # r1 = 0 + 32 = 32
00000000 # load HD into r0 from mem loc 32
00100100 # r1 = 32 - HD
00010100 # store x (which is 32 - HD) into mem loc 32, overwriting HD
01010000 # set r0 = 0
00001100 # load val '0' into r3
01001111 # r3 = 0 + 3 = 3
01010001 # set r0 = 1 to avoid next cases initially
00001111 # load best_matching_score val stored in mem loc 3 into r3
00110111 # if x < best_matching_score then R0 == 0
01111110 #jumps back 2^6 to a_loop;
00110111 # if x = best_matching_score then R0 == -1
01010000 # set r0 = 0
00001100 # load val '0' into r3
01001111 # r3 = 0 + 3 = 3
01001101 # r3 = 1 + 3 = 4
00001111 # load best_matching_count into r3 from mem loc 4
01001101 # r3 = best_matching_count + 1
01010001 # set r0 = 1
01000000 # r0 = 1 + 1 = 2
01000000 # r0 = 2 + 2 = 4
00011100 # store incremented best_matching_count back to mem loc 4
01111110 #jumps back 2^6 to a_loop;
01010001 # set r0 = 1
00001100 # load val '0' into r3
01001101 # r3 = 1 + 2 = 3
00000011 # reset best_matching_count to 1 at mem loc 3; # x is in mem loc 32
01010000 # set r0 = 0
00001100 # load val '0' into r3
01001111 # r3 = 0 + 3 = 3
01000011 # r3 = 0 + 3 = 3
00101110 # r3 = 3 - 2 = 1
01011111 # r3 = 1 << 3 = 8
01011110 # r3 = 8 << 2 = 32
00001111 # x stored in r3 from mem loc 32
```

00001100 # x replaces old best_matching_count with new one at mem loc 3
01110111 # jump back by 2^7 to a_loop
01111111 # (HALT)

3. Output of Python Disassembler for Program 1:

```
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
init $0, 0
sw $2, ($0)
```

```
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
addi $2, 1
init $0, 1
sw $2, ($0)
```

```
init $0, 0
addi $3, 1
addi $3, 1
add $0, $3
subi $3, 1
subi $3, 1
subi $2, 1
subi $2, 1
subi $2, 1
subi $2, 1
subi $2, 1
subi $2, 1
subi $2, 1
sw $2, ($0)
```

```
init $0, 0
lw $2, ($0)
```

```
add $1, $2
addi $3, 1
sub $0, $3
beqR0 2
j 3
```

```
subi $3, 1
subi $3, 1
subi $3, 1
subi $3, 1
subi $3, 1
subi $3, 1
init $0, 0
addi $3, 1
addi $3, 1
addi $3, 1
add $0, $3
sw $1, ($0)
init $0, 0
subi $3, 1
add $0, $3
lw $1, ($0)
subi $3, 1
sub $1, $3
sw $1, ($0)
sltR0 $1, $3
subi $3, 1
init $0, 0
addi $3, 1
addi $3, 1
addi $3, 1
add $0, $3
lw $2, ($0)
sll $2, 0
subi $3, 1
subi $3, 1
subi $3, 1
addi $3, 1
addi $3, 1
addi $3, 1
addi $3, 1
add $0, $3
subi $3, 1
subi $3, 1
subi $3, 1
subi $3, 1
lw $1, ($0)
init $0, 0
beqR0 2
j 6
```

```
init $0, 0
addi $3, 1
addi $3, 1
addi $3, 1
add $0, $3
subi $3, 1
subi $3, 1
subi $3, 1
lw $1, ($0)
```

```
init $1, 0
add $0, $0
add $0, $3
subi $3, 1
lw $2, ($0)


sub $1, $2
sltR0 $1, $2
beqR0 2
j 2

sw $1, ($0)
```

Output of Python Disassembler for Program 2:

```
init $0, 1#setr0=1temporarily
init $1, 1#setr1=1
add $1, $1#r1=1+1=2
add $1, $0#r1=2+1=3
init $0, -1#setr0=-1
sw $0, ($1)#set$r0valof-1intomemloc3whichis0x0003
init $0, 1#setr0=1temporarily
add $1, $0#r1=3+1=4
init $0, -1#setr0backto-1
sw $0, ($1)#set$r0valof-1intomemloc4whichis0x0004
init $0, 0#resetr0backto0
init $1, 0#resetr1backto0

sw $0, ($0)#r0isalready=0,set$r0valof0intomemloc0whichis0x0000
init $1, 1#setr1to1
sw $1, ($1)#set$r1valof1intomemloc1whichis0x0001
init $1, 0#resetr1backto0,r0isstill0atthispoint

lw $2, ($0)#r0is0atthispoint,r2now=0
lw $3, ($0)#r0is0atthispoint,r3now=0

addi $2, 3#$r2=0+3=3
sll $2, 2#$r2=3*2^2=3*4=12
addi $3, 1#$r3=0+2=2
sw $2, ($3)#storer2valof12intomemloc2or0x0002
addi $3, 3#r3=2+3=5
init $0, 0#setr0=0
lw $2, ($0)#load0intor2frommemloc0or0x0000
sw $2, ($3)#store0intomemloc5
addi $2, 1#r2=0+1=1
addi $3, 1#r3=5+1=6
sw $2, ($3)#store1intomemloc6
addi $2, 1#r2=1+1=2
addi $3, 1#r3=6+1=7
sw $2, ($3)#store2intomemloc7
addi $2, 1#r2=2+1=3
addi $3, 1#r3=7+1=8
sw $2, ($3)#store3intomemloc8
addi $2, 1#r2=3+1=4
addi $3, 1#r3=8+1=9
sw $2, ($3)#store4intomemloc9
addi $2, 1#r2=4+1=5
addi $3, 1#r3=9+1=10
sw $2, ($3)#store5intomemloc10
```

```
add $1, $1#r2=5+1=6
addi $3, 1#r3=10+1=11
sw $2, ($3)#store6intomemloc11
addi $2, 1#r2=6+1=7
addi $3, 1#r3=11+1=12
sw $2, ($3)#store7intomemloc12
addi $2, 1#r2=7+1=8
addi $3, 1#r3=12+1=13
sw $2, ($3)#store8intomemloc13
addi $2, 1#r2=8+1=9
addi $3, 1#r3=13+1=14
sw $2, ($3)#store9intomemloc14
addi $2, 1#r2=9+1=10
addi $3, 1#r3=14+1=15
sw $2, ($3)#store10intomemloc15
addi $2, 1#r2=10+1=11
addi $3, 1#r3=15+1=16
sw $2, ($3)#store11intomemloc16
addi $2, 1#r2=11+1=12
addi $3, 1#r3=16+1=17
sw $2, ($3)#store13intomemloc17
addi $2, 1#r2=12+1=13
addi $3, 1#r3=17+1=18
sw $2, ($3)#store14intomemloc18
addi $2, 1#r2=13+1=14
addi $3, 1#r3=18+1=19
sw $2, ($3)#store15intomemloc19
addi $2, 1#r2=14+1=15
addi $3, 1#r3=19+1=20
sw $2, ($3)#store16intomemloc20
addi $2, 1#r2=15+1=16
addi $3, 1#r3=20+1=21
sw $2, ($3)#store17intomemloc21
addi $2, 1#r2=16+1=17
addi $3, 1#r3=21+1=22
sw $2, ($3)#store18intomemloc22
addi $2, 1#r2=17+1=18
addi $3, 1#r3=22+1=23
sw $2, ($3)#store19intomemloc23
addi $2, 1#r2=18+1=19
addi $3, 1#r3=23+1=24
sw $2, ($3)#store20intomemloc24

init $0, 0#resetr0=0
init $1, 0#resetr1=0
lw $2, ($0)#resetr2=0
lw $3, ($0)#resetr3=0

addi $2, 1#registersareinitiallyat0,sor2=0+1=1
sll $2, 3#r2=r2<<3=8
sll $2, 1#r2=r2<<1=16
addi $2, 3#r2=16+3=19
addi $2, 1#r2=19+1=20
addi $3, 3#r3=0+3=3
addi $3, 1#r3=3+1=4
addi $3, 2#r3=4+2=6
add $0, $2#r0=0+20=20
add $0, $3#r0=20+6=26,thisisforthememloc
sw $2, ($0)#store$r2valof20intomemloc26whichis0x001A

addi $2, 3#r2=20+3=23
addi $2, 3#r2=23+3=26
addi $2, 1#r2=26+1=27
add $0, $3#r0=26+6=32
sw $0, ($2)#store$r0valof32intomemloc27whichis0x001B

subi $2, 3#r2=27-3=24
subi $2, 3#r2=24-3=21
subi $2, 1#r2=21-1=20

init $1, 0#setr1=0
lw $3, ($1)#loadval0frommemlocation0or0x0000,r3=0
subi $3, 1#r3=0-1=-1
init $1, 0#resetr1to0
init $0, 0#setr0to0


init $0, 0#alwayssetsr0to0first
add $0, $2#sincer0isalways0beforehand,r0=r2everytime
beqR0 6#jumpforward2^6toleavea_loop
subi $2, 1#decrementr2
addi $3, 1#incrementr3
lw $1, ($3)#loadarrayvalueatmemloc'r3'intor1;#loadTvalin0x0002intor0

init $0, 1#setr0=1
add $0, $0#r0=1+1=2
lw $0, ($0)#loadTfrommemloc0x0002intor0,r0now=T
xor $0, $1#XORofr0andr1,storeresultintor0
init $0, 0#setr0=0
sw $1, ($0)#storeXORresultintomemloc0x0000temporarily,overwriting'0'
init $0, 1#setr0=1
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
add $0, $0#r0=4+4=8
add $0, $0#r0=8+8=16
add $0, $0#r0=16+16=32
init $1, 1#setr1=1,usedtodecrementr0by1
sub $0, $1#r0=32-1=31
add $1, $1#r1=1+1=2
add $1, $1#r1=2+2=4
add $1, $1#r1=4+4=8
add $1, $1#r1=8+8=16
add $1, $1#r1=16+16=32
add $1, $1#r1=32+32=64
sw $0, ($1)#storer0valof31intomemloc64;#storearrayelement#inr3intomemloc28or0x001C

add $1, $1#r1=1+1=2
add $1, $1#r1=2+2=4
init $0, 1#setr0=1
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
add $0, $0#r0=4+4=8
add $0, $0#r0=8+8=16
add $0, $0#r0=16+16=32
sub $0, $1#r0=32-4=28
sw $3, ($0)#storer3=arrayelement#inmemloc28
lw $3, ($1)#loadval1intor3frommemlocation0x0001
init $0, 0#setr0=0forHD
init $1, 1#setr1=1
add $1, $1#r1=1+1=2
add $1, $1#r1=2+2=4
add $1, $1#r1=4+4=8
add $1, $1#r1=8+8=16
add $1, $1#r1=16+16=32
sw $0, ($1)#storeHDintomemloc32
init $1, 1#setr1=1formasking

init $0, 1#setr0=1
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
add $0, $0#r0=4+4=8
add $0, $0#r0=8+8=16
add $0, $0#r0=16+16=32
add $0, $0#r0=32+32=64
lw $0, ($0)#loadr0valof31initstoredinmemloc64backintor0
beqR0 5#jumpforward2^5toleavem_loop
init $0, 0#setr0=0sor3canaccessmemloc0x00/XORresult
lw $3, ($0)#r3=XORresultwhichisstoredinmemloc0x0000
and $3, $1#ANDtheXORresult(r3)withmaskingvalue(r1)
add $1, $1#shiftby1formaskvalue
init $0, 0#setr0=0tocomparewithANDresult
seqR0 $3, $0#checkif$r0=$r3,ifsothenr0=-1
beqR0 4#jumpback-2^4tom_loop
init $0, 1#setr0=1
lw $3, ($0)#loadval1intor3frommemlocation0x0001
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
add $0, $0#r0=4+4=8
add $0, $0#r0=8+8=16
add $0, $0#r0=16+16=32
lw $0, ($0)#loadHDintor0frommemloc32
add $0, $3#HD+1(r3)
sll $3, 3#r3=1<<3=8
sll $3, 2#r3=8<<2=32
sw $0, ($3)#storeincrementedHDbacktomemloc32
j 7#jumpsback2^7toreacha_loop

init $0, 0#setr0=0
sw $0, ($0)#setmemloc0x0000backto'0'overwritingXORresult

init $0, 1#setr0=1
addi $3, 3#r3=1+3=4
addi $3, 1#r3=4+1=5
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
add $0, $0#r0=4+4=8
add $0, $0#r0=8+8=16
add $0, $0#r0=16+16=32
init $1, 0#setr1=0
add $1, $0#r1=0+32=32
lw $0, ($0)#loadHDintor0frommemloc32
sub $1, $0#r1=32-HD
sw $1, ($0)#storex(whichis32-HD)intomemloc32,overwritingHD
init $0, 0#setr0=0
lw $3, ($0)#loadval'0'intor3
addi $3, 3#r3=0+3=3
init $0, 1#setr0=1toavoidnextcasesinitially
lw $3, ($3)#loadbest_matching_scorevalstoredinmemloc3intor3
sltR0 $1, $3#ifx<best_matching_scorethenR0==0
beqR0 6#jumpsback2^6toa_loop;
sltR0 $1, $3#ifx=best_matching_scorethenR0==-1
```

```
init $0, 0#setr0=0
lw $3, ($0)#loadval'0'intor3
addi $3, 3#r3=0+3=3
addi $3, 1#r3=1+3=4
lw $3, ($3)#loadbest_matching_countintor3frommemloc4
addi $3, 1#r3=best_matching_count+1
init $0, 1#setr0=1
add $0, $0#r0=1+1=2
add $0, $0#r0=2+2=4
sw $3, ($0)#storeincrementedbest_matching_countbacktomemloc4
beqR0 6#jumpsback2^6toa_loop;
```

```
init $0, 1#setr0=1
lw $3, ($0)#loadval'0'intor3
addi $3, 1#r3=1+2=3
lw $0, ($3)#resetbest_matching_countto1atmemloc3;#xisinmemloc32
```

```
init $0, 0#setr0=0
lw $3, ($0)#loadval'0'intor3
addi $3, 3#r3=0+3=3
add $0, $3#r3=0+3=3
subi $3, 2#r3=3-2=1
sll $3, 3#r3=1<<3=8
sll $3, 2#r3=8<<2=32
lw $3, ($3)#xstoredinr3frommemloc32
lw $3, ($0)#xreplacesoldbest_matching_countwithnewoneatmemloc3
j 7#jumpbackby2^7toa_loop
```

```
Halt#(HALT)
```

4. Python Code for ISA Disassembler:

```
# Authors: Henry Sampson, Karim, Eric
# SIC instruction encoding format:
# ~~: R0/R1
# --: R2/R3
# ii: immediate
# lw    p 000 xx yy
# sw    p 001 xx yy
# add   p 100 ~~ --
# addi  p 100 -- ii
# sub   p 010 ~~ --
# subi  p 010 -- ii
# sltR0 p 011 ~~ --
# seqR0 p 011 -- ~~
# xor   p 110 ~~ --
# and   p 110 -- ~~
# init  p 101 ~~ ii
# sll   p 101 -- ii
# j     p 111 0i ii
# beqR0 p 111 1i ii
# Halt  p 111 11 11
# ----------------------------------------------------------
```

```
print("ECE366 Fall 2018 mini SIC disassembler")
input_file = open("MIPS_machine_code", "r")
output_file = open("MIPS_asm.txt", "w")

for line in input_file:
    line = line.replace("\n", "")  # remove 'endline' character
    if(line[0:1] == '0'):
        line = line.replace("0", "", 1)  # remove parity bit
    else:
        line = line.replace("1", "", 1)  # remove parity bit
    line = line.replace(" ", "")  # remove spaces anywhere in line

    if (line[0:3] == '000'):  # lw
        line = line.replace("000", "lw ", 1)  # remove 000 and use lw
        if(line[3:5] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif(line[3:5] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif(line[3:5] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[7:9] == '00'):
            line = line.replace('00', '($0)', 1)
        elif (line[7:9] == '01'):
            line = line.replace('01', '($1)', 1)
        elif (line[7:9] == '10'):
            line = line.replace('10', '($2)', 1)
        else:
            line = line.replace('11', '($3)', 1)

    elif (line[0:3] == '001'):  # sw
        line = line.replace("001", "sw ", 1)  # remove 000 and use sw
        if(line[3:5] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif(line[3:5] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif(line[3:5] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[7:9] == '00'):
            line = line.replace('00', '($0)', 1)
        elif (line[7:9] == '01'):
            line = line.replace('01', '($1)', 1)
        elif (line[7:9] == '10'):
            line = line.replace('10', '($2)', 1)
        else:
            line = line.replace('11', '($3)', 1)



    elif (line[0:3] == '100'):  # add/addi
        if(line[3:4] == '0'):
            line = line.replace('100', 'add ', 1)
            if (line[4:6] == '00'):
                line = line.replace('00', '$0, ', 1)
            elif (line[4:6] == '01'):
                line = line.replace('01', '$1, ', 1)
            elif (line[4:6] == '10'):
                line = line.replace('10', '$2, ', 1)
            else:
                line = line.replace('11', '$3, ', 1)

            if (line[8:10] == '00'):
                line = line.replace('00', '$0', 1)
            elif (line[8:10] == '01'):
                line = line.replace('01', '$1', 1)
            elif (line[8:10] == '10'):
                line = line.replace('10', '$2', 1)
            else:
                line = line.replace('11', '$3', 1)
        else:
            line = line.replace('100', 'addi ', 1)
            if (line[5:7] == '00'):
                line = line.replace('00', '$0, ', 1)
            elif (line[5:7] == '01'):
                line = line.replace('01', '$1, ', 1)
            elif (line[5:7] == '10'):
                line = line.replace('10', '$2, ', 1)
            else:
                line = line.replace('11', '$3, ', 1)

            if (line[9:11] == '00'):
                line = line.replace('00', '0')
            elif (line[9:11] == '01'):
                line = line.replace('01', '1')
            elif (line[9:11] == '10'):
                line = line.replace('10', '2')
            elif (line[9:11] == '11'):
                line = line.replace('11', '3')
```

```
elif (line[0:3] == '010'):  # sub/subi
    if(line[3:4] == '0'):
        line = line.replace('010', 'sub ', 1)
        if (line[4:6] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif (line[4:6] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif (line[4:6] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[8:10] == '00'):
            line = line.replace('00', '$0', 1)
        elif (line[8:10] == '01'):
            line = line.replace('01', '$1', 1)
        elif (line[8:10] == '10'):
            line = line.replace('10', '$2', 1)
        else:
            line = line.replace('11', '$3', 1)
    else:
        line = line.replace('010', 'subi ', 1)
        if (line[5:7] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif (line[5:7] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif (line[5:7] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[9:11] == '00'):
            line = line.replace('00', '0')
        elif (line[9:11] == '01'):
            line = line.replace('01', '1')
        elif (line[9:11] == '10'):
            line = line.replace('10', '2')
        elif (line[9:11] == '11'):
            line = line.replace('11', '3')

elif (line[0:3] == '011'):  # sltR0/seqR0
    if(line[3:4] == '0'):
        line = line.replace('011', 'sltR0 ', 1)
        if (line[6:8] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif (line[6:8] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif (line[6:8] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[10:12] == '00'):
            line = line.replace('00', '$0', 1)
        elif (line[10:12] == '01'):
            line = line.replace('01', '$1', 1)
        elif (line[10:12] == '10'):
            line = line.replace('10', '$2', 1)
        else:
            line = line.replace('11', '$3', 1)
    else:
        line = line.replace('011', 'seqR0 ', 1)
        if (line[6:8] == '00'):
            line = line.replace('00', '$0, ', 1)
        elif (line[6:8] == '01'):
            line = line.replace('01', '$1, ', 1)
        elif (line[6:8] == '10'):
            line = line.replace('10', '$2, ', 1)
        else:
            line = line.replace('11', '$3, ', 1)

        if (line[10:12] == '00'):
            line = line.replace('00', '$0', 1)
        elif (line[10:12] == '01'):
            line = line.replace('01', '$1', 1)
        elif (line[10:12] == '10'):
            line = line.replace('10', '$2', 1)
        else:
            line = line.replace('11', '$3', 1)

elif (line[0:3] == '110'):  # xor/and
    if (line[3:4] == '0'):
        line = line.replace('110', 'xor ', 1)
    else:
        line = line.replace('110', 'and ', 1)

    if (line[4:6] == '00'):
        line = line.replace('00', '$0, ', 1)
    elif (line[4:6] == '01'):
        line = line.replace('01', '$1, ', 1)
    elif (line[4:6] == '10'):
        line = line.replace('10', '$2, ', 1)
    else:
        line = line.replace('11', '$3, ', 1)
if (line[8:10] == '00'):
    line = line.replace('00', '$0', 1)
elif (line[8:10] == '01'):
    line = line.replace('01', '$1', 1)
elif (line[8:10] == '10'):
    line = line.replace('10', '$2', 1)
else:
    line = line.replace('11', '$3', 1)

elif (line[0:3] == '101'):  # init/sll
if(line[3:4] == '1'):
    line = line.replace('101', 'sll ', 1)
    if (line[4:6] == '00'):
        line = line.replace('00', '$0, ', 1)
    elif (line[4:6] == '01'):
        line = line.replace('01', '$1, ', 1)
    elif (line[4:6] == '10'):
        line = line.replace('10', '$2, ', 1)
    else:
        line = line.replace('11', '$3, ', 1)

    if (line[8:10] == '00'):
        line = line.replace('00', '0')
    elif (line[8:10] == '01'):
        line = line.replace('01', '1')
    elif (line[8:10] == '10'):
        line = line.replace('10', '2')
    elif (line[8:10] == '11'):
        line = line.replace('11', '3')
else:
    line = line.replace('101', 'init ', 1)
    if (line[5:7] == '00'):
        line = line.replace('00', '$0, ', 1)
    elif (line[5:7] == '01'):
        line = line.replace('01', '$1, ', 1)
    elif (line[5:7] == '10'):
        line = line.replace('10', '$2, ', 1)
    else:
        line = line.replace('11', '$3, ', 1)

    if (line[9:11] == '00'):
        line = line.replace('00', '0')
    elif (line[9:11] == '01'):
        line = line.replace('01', '1')
    elif (line[9:11] == '10'):
        line = line.replace('10', '-2')
    elif (line[9:11] == '11'):
        line = line.replace('11', '-1')
elif (line[0:3] == '111'):  # j/beqR0
    if (line[3:4] == '0'):
        line = line.replace('1110', 'j ', 1)
        if (line[2:5] == '000'):
            line = line.replace('000', '0')
        elif (line[2:5] == '001'):
            line = line.replace('001', '1')
        elif (line[2:5] == '010'):
            line = line.replace('010', '2')
        elif (line[2:5] == '011'):
            line = line.replace('011', '3')
        elif (line[2:5] == '100'):
            line = line.replace('100', '4')
        elif (line[2:5] == '101'):
            line = line.replace('101', '5')
        elif (line[2:5] == '110'):
            line = line.replace('110', '6')
        else:
            line = line.replace('111', '7')
    else:
        line = line.replace('1111', 'beqR0 ', 1)
        if (line[6:9] == '000'):
            line = line.replace('000', '0')
        elif (line[6:9] == '001'):
            line = line.replace('001', '1')
        elif (line[6:9] == '010'):
            line = line.replace('010', '2')
        elif (line[6:9] == '011'):
            line = line.replace('011', '3')
        elif (line[6:9] == '100'):
            line = line.replace('100', '4')
        elif (line[6:9] == '101'):
            line = line.replace('101', '5')
        elif (line[6:9] == '110'):
            line = line.replace('110', '6')
        else:
            line = line.replace('beqR0 111', 'Halt')

else:
    print("Unknown instruction:" + line)

output_file.write(line + "\n")

input_file.close()
output_file.close()
```

Part D)

1. ALU Schematic

2. CPU Datapath Design

3. Control Logic Design