INTRODUCTION:

This activity is used to design and develop two assembly codes in MARS and gain some familiarity of ISA control structures along with the usage of $lo, $hi registers by logging few selected registers' values and memory content of selected addresses in a table for each task.

MY GROUP:

Name: Student_Team 6
Members: Tirumala Saiteja Goruganthu (016707210), Harish Marepalli (016707314)

SOURCE CODE (Task-1):

```
ori $a0, $0, 0x8000           #store value into variable a
ori $a1, $0, 0x00A9           #store value into variable b
ori $s0, $0, 1974             #store value into variable c
multu $a0, $a0                #multiply a with itself
mflo $s1                      #store in s1 register (no overflow condition)
addi $t0, $0, 0x20            #store base address in temporary register t0
sw $s1, 0($t0)                #store x to location 0x20
multu $s1, $a1                #multiply x with b
mfhi $s2                      #move hi content to y
sw $s1, 4($t0)                #store lo content to [y](ask doubt on what to store in memory location for y i.e, y.hi or y.lo)
sw $s2, 8($t0)                #store hi content to [y+4]
srl $s2, $s1, 16              #right shift y.lo value
jal compute                   #jump to compute label to calculate the given formula
sw $s0, 12($t0)               #store c value to 0x2c location
addi $t3, $0, 1               #store value '1' in t2 register for future beq comparison
while:   slti $t1, $s0, 1665  #check if c<1665 => t1 =1 else t1=0
beq $t1, $t3, done            #branch if t1==t2 to done since we should come out of the while loop
jal compute                   #jump to compute label to calculate the given formula
j while                       #loop back to while
compute:  divu $s2, $s0       #divide y with c and store in temporary t1
mflo $t1                      #move the quotient of previous result to t1 register
add $t1, $s0, $t1             #add c to previous result and store in temporary t1
srl $s0, $t1, 1               #right shift by amount '1' to essentially divide by 2
jr $ra                        #jump to return address given by ra register
done:sll $s0, $s0, 8          #logic left shift c value by 8 and store it back in c (s0)
sw $s0, 16($t0)               #store c value to 0x30 location
```

TEST LOG (Task-1):

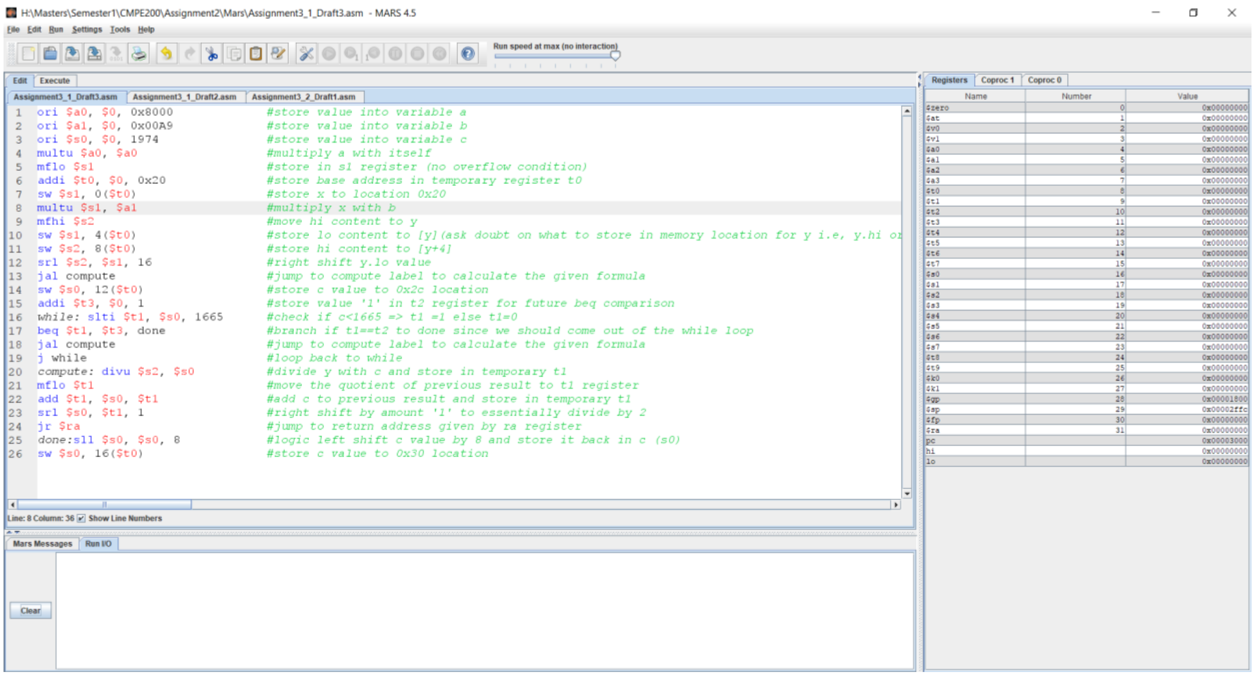
Programmer's Name: Tirumala Saiteja Goruganthu
Date: 09-24-2022

Adr	MIPS Instruction	Machine Code	Registers				
			$a0	$a1	$s0	$s1	$s2
3000	ori $a0, $0, 0x8000	0x34048000	0x00008000	0x00000000	0x00000000	0x00000000	0x00000000
3004	ori $a1, $0, 0x00A9	0x340500a9	0x00008000	0x000000a9	0x00000000	0x00000000	0x00000000
3008	ori $s0, $0, 1974	0x341007b6	0x00008000	0x000000a9	0x000007b6	0x00000000	0x00000000
300c	multu $a0, $a0	0x00840019	0x00008000	0x000000a9	0x000007b6	0x00000000	0x00000000
3010	mflo $s1	0x00008812	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00000000
3014	addi $t0, $0, 0x20	0x20080020	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00000000
3018	sw $s1, 0($t0)	0xad110000	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00000000
301c	multu $s1, $a1	0x02250019	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00000000
3020	mfhi $s2	0x00009010	0x00008000	0x000000a9	0x000007b6	0x40000000	0x0000002a
3024	sw $s1, 4($t0)	0xad110004	0x00008000	0x000000a9	0x000007b6	0x40000000	0x0000002a
3028	sw $s2, 8($t0)	0xad120008	0x00008000	0x000000a9	0x000007b6	0x40000000	0x0000002a
302c	srl $s2, $s1, 16	0x00119402	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00004000
3030	jal compute	0x0c000c13	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00004000
3034	sw $s0, 12($t0)	0xad10000c	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
3038	addi $t3, $0, 1	0x200b0001	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
303c	while: slti $t1, $s0, 1665	0x2a090681	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
3040	beq $t1, $t3, done	0x112b0007	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
3044	jal compute	0x0c000c13					
3048	j while	0x08000c0f					
304c	compute: divu $s2, $s0	0x0250001b	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00004000
3050	mflo $t1	0x00004812	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00004000
3054	add $t1, $s0, $t1	0x02094820	0x00008000	0x000000a9	0x000007b6	0x40000000	0x00004000
3058	srl $s0, $t1, 1	0x00098042	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
305C	jr $ra	0x03e00008	0x00008000	0x000000a9	0x000003df	0x40000000	0x00004000
3060	done: sll $s0, $s0, 8	0x00108200	0x00008000	0x000000a9	0x0003df00	0x40000000	0x00004000
3064	sw $s0, 16($t0)	0xad100010	0x00008000	0x000000a9	0x0003df00	0x40000000	0x00004000
3068							
306C							

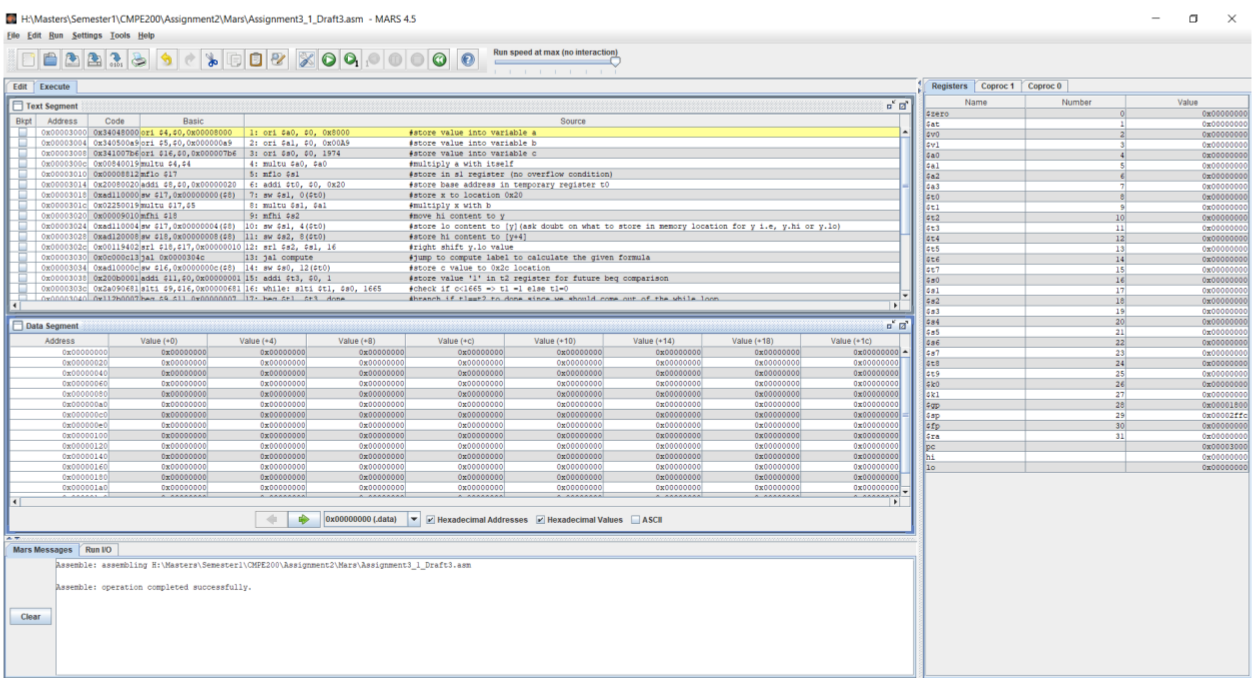
Memory contents				
Word @ 0x20	Word @ 0x24	Word @ 0x28	Word @ 0x2C	Word @ 0x30
0x40000000	0x40000000	0x0000002a	0x000003df	0x0003df00

SCREEN CAPTURES (Task-1):

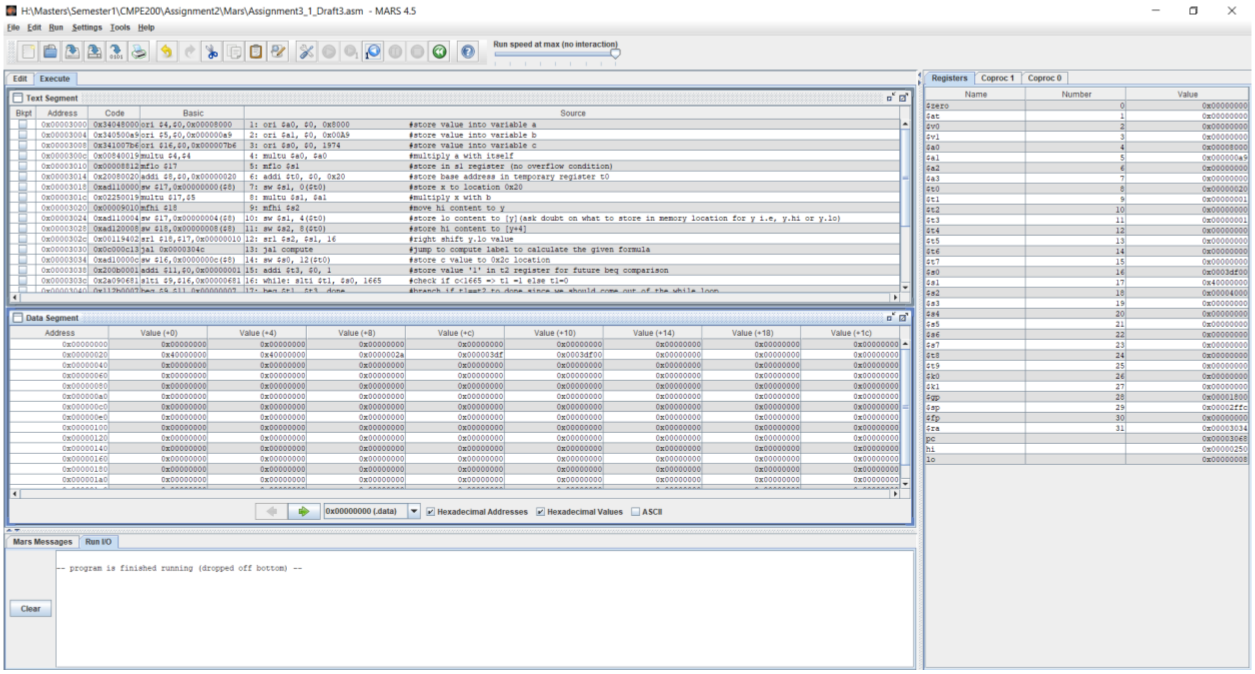
1. Snippet of the editor with task-1 assembly code.



2. Snippet of the assembled code for task 1 (before execution)



3. Snippet of the final execution of task-1 code in MARS



SOURCE CODE (Task-2):

1. *ori $a0, $0, 5* — *#initialize a0 with value 5*
2. *sw $a0, 0($t0)* — *#store the content in a0 to t0 (0x00 memory location)*
3. *ori $t1, $0, 1* — *#initialize t1 with value 1*
4. *while: beq $a0, $0, done* — *#branch if a0=0 to done else run below code*
5. *mult $t1, $a0* — *#multiply t1 with a0 => f*n*
6. *mflo $t1* — *#assign lo content into t1 again => f=f*n*
7. *addi $a0, $a0, -1* — *#decrement the content of a0*
8. *j while* — *#loop back to while label*
9. *done: sw $t1, 16($t0)* — *#store the content of t1 (final answer) to 0x10 location*
10. *lw $s0, 16($t0)* — *#load the content of 0x10 location to register s0*

TEST LOG (Task-2):

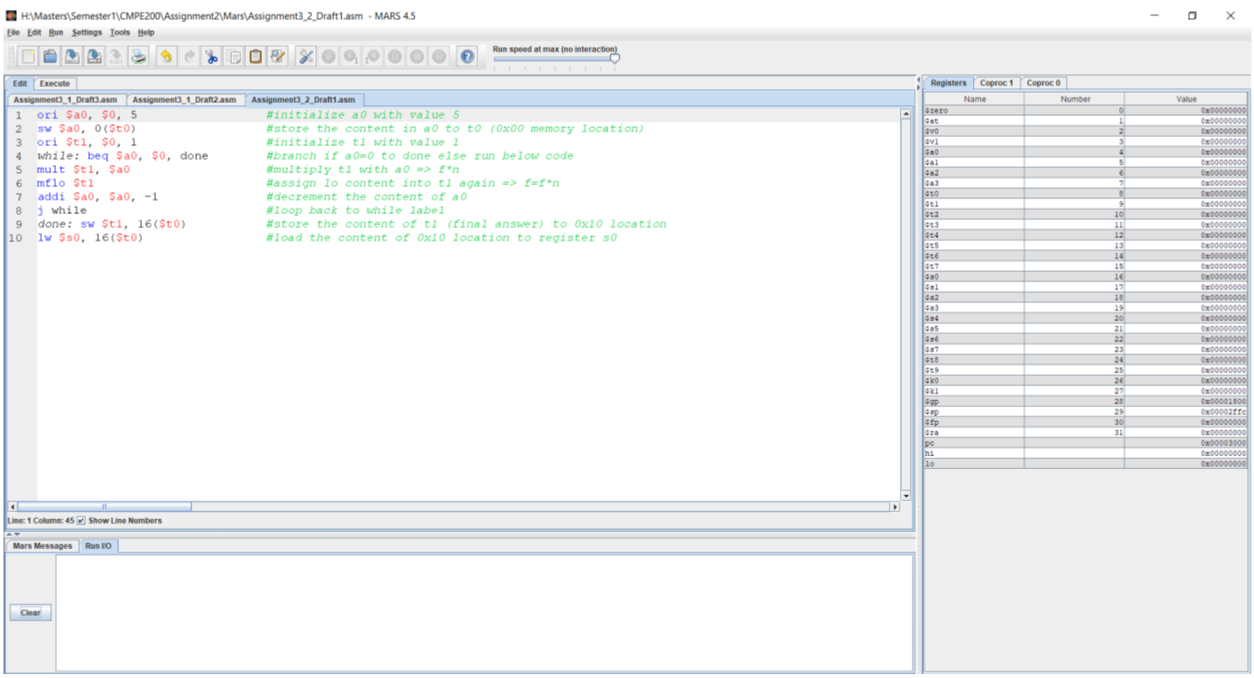
Programmer's Name: Tirumala Saiteja Goruganthu
Date: 09-24-2022

Adr	MIPS Instruction	Machine Code	Registers				Memory Content	
			$a0	**$s0**	**$t0**	**$t1**	Word @ 0x00	Word @ 0x10
3000	ori $a0, $0, 5	0x34040005	0x00000005	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
3004	sw $a0, 0($t0)	0xad040000	0x00000005	0x00000000	0x00000000	0x00000000	0x00000005	0x00000000
3008	ori $t1, $0, 1	0x34090001	0x00000005	0x00000000	0x00000000	0x00000001	0x00000005	0x00000000

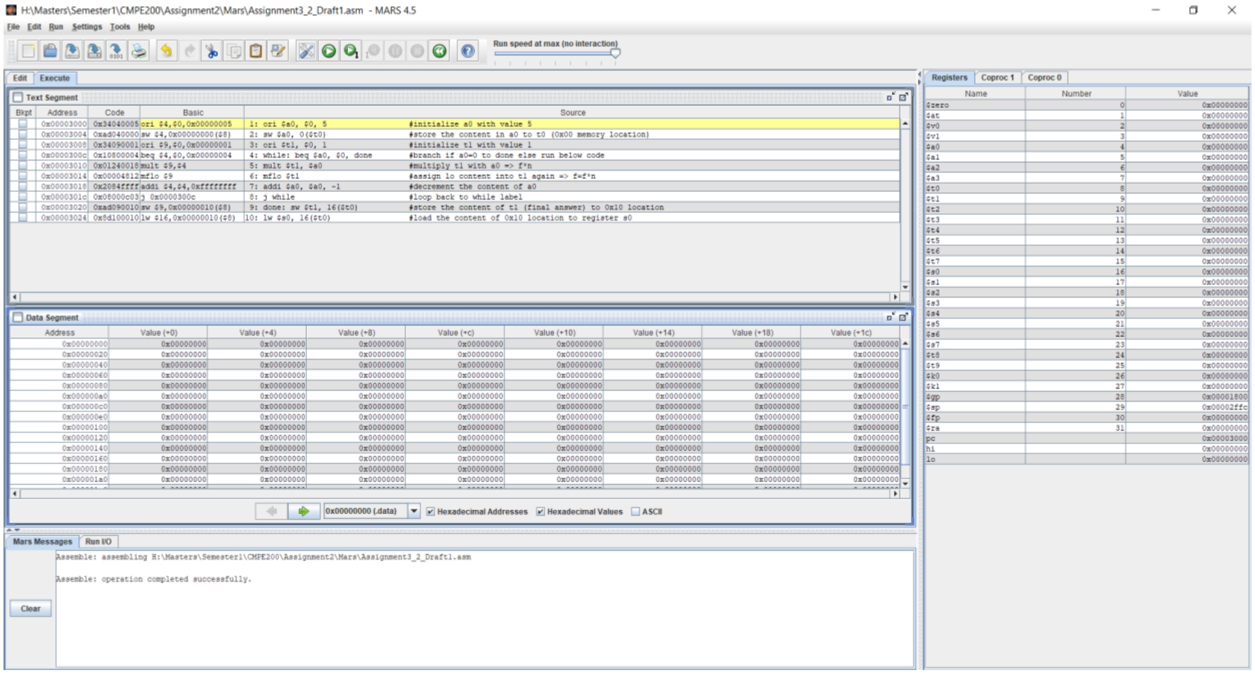
300c	while: beq $a0, $0, done	0x10800004	0x00000000	0x00000000	0x00000000	0x00000078	0x00000005	0x00000000
3010	mult $t1, $a0	0x01240018	0x00000001	0x00000000	0x00000000	0x00000078	0x00000005	0x00000000
3014	mflo $t1	0x00004812	0x00000001	0x00000000	0x00000000	0x00000078	0x00000005	0x00000000
3018	addi $a0, $a0, -1	0x2084ffff	0x00000000	0x00000000	0x00000000	0x00000078	0x00000005	0x00000000
301c	j while	0x08000c03	0x00000000	0x00000000	0x00000000	0x00000078	0x00000005	0x00000000
3020	done: sw $t1, 16($t0)	0xad090010	0x00000000	0x00000000	0x00000000	0x00000078	0x00000005	0x00000078
3024	lw $s0, 16($t0)	0x8d100010	0x00000000	0x00000078	0x00000000	0x00000078	0x00000005	0x00000078
3028								

SCREEN CAPTURES (Task-2):

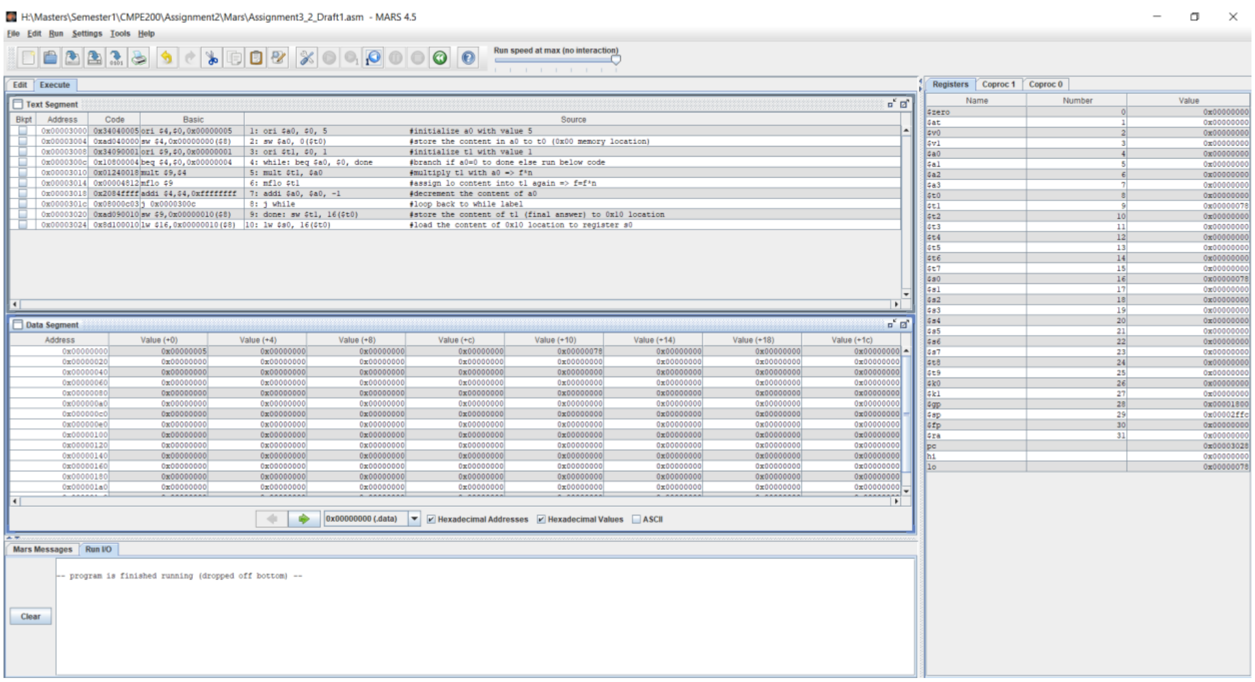
1. Snippet of the editor with task-2 assembly code



2. Snippet of the assembled code for task-2 (before execution)



3. Snippet of the final execution of task-2 code in MARS



DISCUSSION SECTION:

The explanation of unique instructions in the sample code with the help of the MIPS reference data card.

1. multu rs, rt => {hi, lo} = rs * rt
   Performs unsigned multiplication of rs and rt and stores the result in lo and hi registers. Note that hi register is filled only when the result is more than 32 bits.
2. divu rs, rt => hi = rs % rt; lo = rs / rt

Perform unsigned division of rs and rt and stores the result in lo and hi registers. Note that hi register contains the remainder of the result and lo registers contains the quotient of the result.

3. mflo rd => rd = lo
   This instruction will move the content from lo register to the rd register.

4. mfhi rd => rd = hi
   This instruction will move the content from hi register to the rd register.

5. sll rd, rt, sh => rd = rt << sh
   This instruction will perform the bitwise left shift operation on the content of rt register by sh amount and stores back the result to rd.

6. srl rd, rt, sh => rd = rt >> sh
   This instruction will perform the bitwise right shift operation on the content of rt register by sh amount and stores back the result to rd.

7. j addr => PC = addr
   Jumps to the given address by putting its value in the program counter (PC).

8. jal addr => $ra = next PC value; PC = addr
   Jump and Link to the given address by putting its value in the program counter (PC) as well as storing the next PC in $ra register.

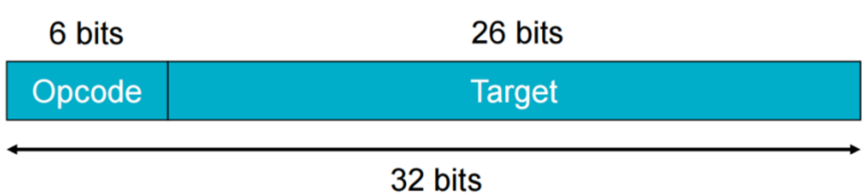
9. jr $ra => PC = $ra
   After a subroutine is completed, the control (PC) should go back to the next instruction after the caller method. We use jr instruction to do the same for us.

Following are the observations with respect to the machine codes:

1. Let us take an example of the task-1 source code where we use J-type instruction.

   0x0000303c   while: slti $t1, $s0, 1665
   ….
   ….
   ….
   ….
   0x00003048   j while => j 0x0000303c

   So, let us find the machine code of ‘j 0x0000303c instruction.
   The Machine code of J-Type instructions can be seen as follows:



   Opcode for j instruction is 000010.
   Problem here is that the address is 32-bit but only 26 bits are available. To overcome this issue, we follow something called ‘Jump Target Addressing.’

   Target address in hex => 0x0000303c

Target address in binary => 0000 0000 0000 0000 0011 0000 0011 1100

Remove the first 4 bits and last 2 bits that we add from next PC and shift left operation respectively to get the required 26-bit immediate.

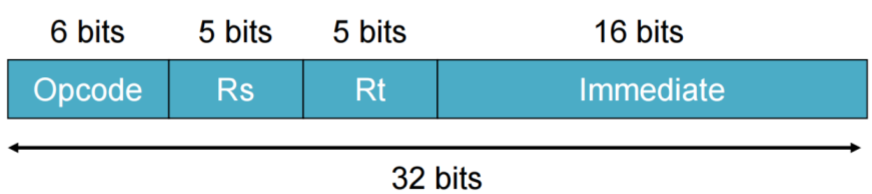
Immediate => 0000 0000 0000 0011 0000 0011 11

Finally, the machine code is opcode + immediate
=> 000010 0000 0000 0000 0011 0000 0011 11
=> 0000 1000 0000 0000 0000 1100 0000 1111
In hexadecimal, the machine code is => 0x08000c0f

2. Let us take an example of the task-1 source code where we use I-type instruction (beq)

0x00003040 beq $t1, $t3, done
….
….
….
….
0x00003060 done: …

The machine code format of I-type instruction is as follows



To find the machine code of the above beq instruction, we need the 16-bit immediate which will be positive value since the branching is happening downwards direction.

We know that the branch target addressing is done using below equation

$PC_{Target} = (PC_{beq} + 4) + 4N$ where N=immediate

Here, PC_{Target} = 0x00003060
PC_{beq} = 0x00003040

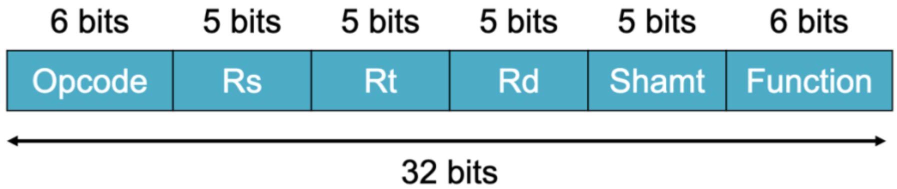
=> 0x00003060 = (0x00003040 + 4) + 4N
=> 4N = 0x0000001c
=> N = 0x00000007 => take only 16bits => 0007

Therefore, the machine code is

=> 000100 01001 01011 0000 0000 0000 0111
=> 0001 0001 0010 1011 0000 0000 0000 0111
In hexadecimal, the machine code is => 0x112b0007

3. Let us take an example of the multu instruction in the task-1 source code.

0x0000300c multu $a0, $a0
The machine code format for the multu (R-Type) instruction is as follows



The opcode-function value for multu instruction is 000000-011001
Rs = $a0 = $4 = 00100
Rt = $a0 = $4 = 00100
Rd = 00000
Shamt = 00000

So, the final machine code is 000000 00100 00100 00000 00000 011001
=> 0000 0000 1000 0100 0000 0000 0001 1001
=> In hexadecimal format we write, 0x00840019

COLLABORATION SECTION:

1. For the given C++ pseudo codes, developed corresponding assembly codes in the MARS software by collaborating with each other.
2. Worked together to execute the code and to observe all the operations and values.
3. Collaborated in understanding the concepts such as $hi, $lo register operations, looping control structures and discussed and realized few basics of preparing a J-type instruction machine code using Jump Target Addressing.
4. By working together, we debugged each line to know the contents of the relevant registers and recorded the memory values at certain addresses.

CONCLUSION:

In conclusion, we end the discussion by understanding ISA control structures and the $lo, $hi registers. We also gained insight into the concept of jump target addressing and branch target addressing.