

San Jose State University
Department of Computer Engineering

CMPE 200 Report

Assignment 2 Report

Title MIPS Instruction Set Architecture & Programming (2)

Semester: Fall 2023 **Date:** 09/16/2023

by

Name: Sai Kashyap Kurella **SID:** 016018925

I. SOURCE CODE

mipstest.smd

Test the following MIPS instructions.

add, sub, and, or, slt, addi, lw, sw, beq, j

#	Assembly	Description	Address	Machine
main:	addi \$2, \$0, 5	# initialize \$2 = 5	0	20020005
	addi \$3, \$0, 12	# initialize \$3 = 12	4	2003000c
	addi \$7, \$3, -9	# initialize \$7 = 3	8	2067fff7
	or \$4, \$7, \$2	# \$4 <= 3 or 5 = 7	c	00e22025
	and \$5, \$3, \$4	# \$5 <= 12 and 7 = 4	10	00642824
	add \$5, \$5, \$4	# \$5 = 4 + 7 = 11	14	00a42820
	beq \$5, \$7, end	# shouldn't be taken	18	10a7000a
	slt \$4, \$3, \$4	# \$4 = 12 < 7 = 0	1c	0064202a
	beq \$4, \$0, around	# should be taken	20	10800001
	addi \$5, \$0, 0	# shouldn't execute	24	20050000
around:	slt \$4, \$7, \$2	# \$4 = 3 < 5 = 1	28	00e2202a
	add \$7, \$4, \$5	# \$7 = 1 + 11 = 12	2c	00853820
	sub \$7, \$7, \$2	# \$7 = 12 - 5 = 7	30	00e23822
	sw \$7, 68(\$3)	# [80] = 7	34	ac670044
	lw \$2, 80(\$0)	# \$2 = [80] = 7	38	8c020050
	j end	# should be taken	3c	08000011
	addi \$2, \$0, 1	# shouldn't execute	40	20020001
end:	sw \$2, 84(\$0)	# write adr 84 = 7	44	ac020054
j main		# go back to beginning	48	08000c00

II. TEST LOG

Adr	Machine Code for MARS	Machine Code for MIPSASM	PC	Registers					Memory Content	
				\$v0	\$v1	\$a0	\$a1	\$a3	[80]	[84]
3000	0x20020005	0x20020005	0x00003000	0x00000007	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000000	0x00000000
3004	0x2003000c	0x2003000c	0x00003004	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000000	0x00000000
3008	0x2067fff7	0x2067fff7	0x00003008	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000000	0x00000000
300c	0x00e22025	0x00e22025	0x0000300c	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000003	0x00000000	0x00000000
3010	0x00642824	0x00642824	0x00003010	0x00000005	0x0000000c	0x00000007	0x0000000b	0x00000003	0x00000000	0x00000000
3014	0x00a42820	0x00a42820	0x00003014	0x00000005	0x0000000c	0x00000007	0x00000004	0x00000003	0x00000000	0x00000000
3018	0x10a7000a	0x10e5000a	0x00003018	0x00000005	0x0000000c	0x00000007	0x0000000b	0x00000003	0x00000000	0x00000000
301c	0x0064202a	0x0064202a	0x0000301c	0x00000005	0x0000000c	0x00000007	0x0000000b	0x00000003	0x00000000	0x00000000
3020	0x10800001	0x10040001	0x00003020	0x00000005	0x0000000c	0x00000000	0x0000000b	0x00000003	0x00000000	0x00000000
3024	0x20050000	0x20050000	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken
3028	0x00e2202a	0x00e2202a	0x00003028	0x00000005	0x0000000c	0x00000000	0x0000000b	0x00000003	0x00000000	0x00000000
302c	0x00853820	0x00853820	0x0000302c	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000003	0x00000000	0x00000000
3030	0x00e23822	0x00e23822	0x00003030	0x00000005	0x0000000c	0x00000001	0x0000000b	0x0000000c	0x00000000	0x00000000
3034	0xac670044	0xac670044	0x00003034	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000007	0x00000000
3038	0x8c020050	0x8c020050	0x00003038	0x00000005	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000007	0x00000000
303c	0x08000c11	0x08000011	0x0000303c	0x00000007	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000007	0x00000000
3040	0x20020001	0x20020001	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken	Not Taken
3044	0xac020054	0xac020054	0x00003044	0x00000007	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000007	0x00000000
3048	0x08000c00	0x08000000	0x00003048	0x00000007	0x0000000c	0x00000001	0x0000000b	0x00000007	0x00000007	0x00000007

The screenshot displays the Mars 4.5 debugger interface. The main window is titled "private/var/folders/b5/q4/q0j5f5q9v821_7x1c4w0000gn/T/hspcrdata_kashyapkurella/mpis1.asm - MARS 4.5". The interface is divided into several panes:

- File Edit Run Settings Tools Help**: The top menu bar.
- Run speed at max (no interaction)**: A status bar at the top right.
- Text Segment**: The central pane showing assembly code. It includes columns for Bkpt, Address, Code, Basic, and Source. The code is for a MIPS assembly program that initializes registers \$2, \$3, and \$7, and then performs a loop. The source code is as follows:


```

      6: main:
      7:      addi $2, $0, 5      # initialize $2 = 5
      8:      addi $3, $0, 12     # initialize $3 = 12
      9:      addi $7, $3, -9    # initialize $7 = 3
      10:     or $4, $2, $7      # $4 = 3 or 5 or 7
      11:     and $5, $3, $4      # $5 = 12 and 7 or 4
      12:     beq $5, $5, $4     # shouldn't be taken
      13:     slt $4, $3, $4      # $4 = 12 < 7 or 0
      14:     beq $4, $0, around   # should be taken
      15:     addi $5, $0, 0      # shouldn't execute
      17: around:
      18:     slt $4, $7, $2      # $4 = 3 < 5 = 1
      19:     add $7, $4, $5      # $7 = 1 + 11 = 12
      
```
- Data Segment**: The bottom pane showing memory addresses and their values. It includes columns for Address, Value (+0), Value (+4), Value (+8), Value (+C), Value (+10), Value (+14), Value (+18), and Value (+1c). The values are mostly 0x00000000, with some non-zero values at addresses 0x00000007 and 0x00000008.
- Registers**: The rightmost pane showing the state of registers. It includes columns for Name, Num., and Value. The registers are \$2 through \$31, with values ranging from 0x00000000 to 0x00000007.
- Buttons**: At the bottom, there are buttons for "Mars Messages", "Run I/O", and "Clear".

The screenshot shows the Mars MIPS simulator interface. The main window displays assembly code for a MIPS program. The interface includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar with various icons, and a status bar at the bottom showing "Mars Messages" and "Run I/O".

The assembly code is organized into sections:

- Text Segment:** Contains instructions for initializing variables and performing arithmetic operations.

Bkpt	Address	Code	Basic	Source
0x00003000	0x20020005	addi \$2, \$0, 0x00000005	6: main:	addi \$2, \$0, 5 # initialize \$2 = 5
0x00003004	0x2003000c	addi \$3, \$0, 0x0000000c	7:	addi \$3, \$0, 12 # initialize \$3 = 12
0x00003008	0x2067ffff	addi \$7, \$3, 0xffffffffff	8:	addi \$7, \$3, -9 # initialize \$7 = 3
0x0000300c	0x00e22025	or \$4, \$7, \$2	9:	or \$4, \$7, \$2 # \$4 <= 3 or 5 = 7
0x00003010	0x00e42824	and \$5, \$3, \$4	10:	and \$5, \$3, \$4 # \$5 <= 12 and 7 = 4
0x00003014	0x00e42820	add \$5, \$5, \$4	11:	add \$5, \$5, \$4 # \$5 = 4 + 7 = 11
0x00003018	0x10a7000a	beq \$5, \$7, end	12:	beq \$5, \$7, end # shouldn't be taken
0x0000301c	0x00e4282a	slt \$4, \$3, \$4	13:	slt \$4, \$3, \$4 # \$4 = 12 < 7 = 0
0x00003020	0x10800001	beq \$4, \$0, around	14:	beq \$4, \$0, around # should be taken
0x00003024	0x20050000	addi \$5, \$0, 0x00000000	15:	addi \$5, \$0, 0 # shouldn't execute
0x00003028	0x00e2202a	slt \$4, \$7, \$2	17:	around: slt \$4, \$7, \$2 # \$4 = 3 < 5 = 1
0x0000302c	0x00053820	add \$7, \$4, \$5	18:	add \$7, \$4, \$5 # \$7 = 1 + 11 = 12
- Data Segment:** Contains memory addresses and their corresponding values.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
- Registers:** Shows the state of various registers.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000007
\$v1	3	0x0000000c
\$a0	4	0x00000001
\$a1	5	0x0000000b
\$a2	6	0x00000000
\$a3	7	0x00000007
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001000
\$sp	29	0x000021fc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00003000
hi		0x00000000
lo		0x00000000

The status bar at the bottom shows "Mars Messages" and "Run I/O".

MIPS Assembler and Simulator - [D:\Cmpe-200\MIPSASM\Mips_test.smd]

Clipboard: Paste, Copy path, Move to, Copy to, Delete, Rename, Organize

Registers / Memory

Register	Value	Register	Value	Register	Value
\$5	00000000	\$9	00000000		
\$6	00000000	\$t0	00000000		
\$7	00000000	\$t1	00000000		
\$t0	00000001	\$gp	00000000		
\$t1	0000000B	\$sp	00000200		
\$t2	00000000	\$t2	00000000		
\$t3	00000007	\$t3	00000000		
\$t4	00000000				
\$t5	00000000	\$lo	00000000		
\$t6	00000000	\$hi	00000000		
\$t7	00000000				
\$t8	00000000				
\$t9	00000000				
\$t10	00000000				
\$t11	00000000				
\$t12	00000000				
\$t13	00000000				
\$t14	00000000				
\$t15	00000000				

Code

```
0x044 00000000  
0x048 00000000  
0x04C 00000000  
0x050 00000000  
0x054 00000000  
0x058 00000000  
0x05C 00000000  
0x060 00000000  
0x064 00000000  
0x068 00000000  
0x06C 00000000  
0x070 00000000  
0x074 00000000  
0x078 00000000  
0x07C 00000000  
0x080 00000000  
0x084 00000000  
0x088 00000000  
0x08C 00000000  
0x090 00000000  
0x094 00000000
```

Machine Code

Address	Code	Instruction	Comment
0x00000000	0x20020005	addi \$v0, \$zero, 5	\$v0 = 5
0x00000004	0x2003000C	addi \$v1, \$zero, 12	\$v1 = 12
0x00000008	0x2067FFFF	addi \$a3, \$v1, -9	\$a3 = \$v1 + -9
0x0000000C	0x00E22025	or \$a0, \$a3, \$v0	\$a0 = \$a3 \$v0
0x00000010	0x00642824	and \$a1, \$v1, \$a0	\$a1 = \$v1 & \$a0
0x00000014	0x00A42820	add \$a1, \$a1, \$a0	\$a1 = \$a1 + \$a0
0x00000018	0x10E5000A	beq \$a3, \$a1, 10	if (\$a3 == \$a1) goto
0x0000001C	0x0064202A	slt \$a0, \$v1, \$a0	if (\$v1 < \$a0) \$a0 =
0x00000020	0x10040001	beq \$zero, \$a0, 1	if (\$zero == \$a0) got
0x00000024	0x20050000	addi \$a1, \$zero, 0	\$a1 = 0
0x00000028	0x00E2202A	slt \$a0, \$a3, \$v0	if (\$a3 < \$v0) \$a0 =
0x0000002C	0x00853820	add \$a3, \$a0, \$a1	\$a3 = \$a0 + \$a1
0x00000030	0x00E23822	sub \$a3, \$a3, \$v0	\$a3 = \$a3 - \$v0
0x00000034	0xAC670044	sw \$a3, 68(\$v1)	mem[\$v1 + 68] = \$a3
0x00000038	0x8C020050	lw \$v0, 80(\$zero)	\$v0 = mem[\$zero + 80]
0x0000003C	0x00000011	j 0x00000011	jump to addr 0x00000011

ST6UNST
templates
templedit

18-09-20
07-03-20
20-02-20

```
end: sw $2, 84($0) # write adr 84 = 7 44 ac020054  
j main # go back to beginning 48 08000c00
```

IV. DISCUSSIONS

1. *Initialization:*

At the beginning of the **main** section, there is variable initialization using **addi** instructions. Registers **\$2**, **\$3**, and **\$7** are initialized with values 5, 12, and 3, respectively.

2. *Logical and Arithmetic Operations:*

or: The **or** instruction performs a bitwise OR operation between registers **\$7** and **\$2**, storing the result in **\$4**. In this case, it calculates 3 OR 5, resulting in 7.

and: The **and** instruction performs a bitwise AND operation between registers **\$3** and **\$4**, storing the result in **\$5**. It calculates 12 AND 7, resulting in 4.

add: The **add** instruction adds registers **\$5** and **\$4**, storing the result back in **\$5**. It calculates 4 + 7, resulting in 11.

slt: The **slt** instruction sets **\$4** to 1 if **\$3** is less than **\$4** ($12 < 7$). In this case, it sets **\$4** to 0.

3. *Branching:*

beq: There are two **beq** (branch on equal) instructions. The first one checks if **\$5** is equal to **\$7** and branches to the **end** label if true. The second one checks if **\$4** is equal to 0 and branches to the **around** label if true.

4. *Label Usage:*

Labels like **around**, **end**, and **main** are used as targets for branching instructions, providing control flow within the program.

5. *Memory Operations:*

sw: The **sw** (store word) instruction stores the value of register **\$7** into memory at address offset 68 from the address in register **\$3**.

lw: The **lw** (load word) instruction loads a word from memory at address offset 80 from the address in register **\$0** (which is typically used as a null register) into register **\$2**.

6. *Jump Instructions:*

j: The **j** (jump) instruction is used to unconditionally jump to the specified label. It is used to control the program flow, both at the **end** and **main** labels.

7. *Commented Instruction Addresses:*

The comments indicate the memory addresses for each instruction in hexadecimal format.

8. *Execution Flow:*

The code has conditional branches (**beq**) and unconditional jumps (**j**), which determine the execution flow of the program. The program starts at the **main** label and loops between the **around** and **end** labels.

V. CONCLUSION

To sum up, this MIPS assembly code performs basic arithmetic and logic operations, utilizes branching for conditional execution, and interacts with memory through load and store operations. The specific behaviour and results of the code execution would depend on the initial values of registers and memory locations, as well as the branching conditions. The machine codes of the corresponding assembly codes and register values were observed on the MARS & MIPSASM assemblers which helped me in realizing how an assembly code is interpreted by the underlying CPU.