

2021-Eg 211-Computer Architecture

Assignment 2

Submitted By Vihan Vashishth(Imt2021065) And Vineet Priyadarshi(Imt2021018)

Instructions to run the MIPS program files

- Make sure you are in the directory which contains the file

```
IMT2021018 IMT2021065_Q1A/B.asm
```

- Execute the following command in terminal:

```
java -jar Mars4_5.jar nc <file_name>.asm
```

OR

- In MARS-MIPS, use the F3 to assemble the code and F5 to run it.

Bubble Sort

C equivalent algorithm:

```

for (i = 0; i < n - 1; i++){
    for (j = 0; j < n - i - 1; j++)
    {
        if (arr[j] > arr[j + 1])
        {
            swap(arr[j], arr[j + 1]);
        }
    }
}

```

The MIPS code:

- The program first takes 'n' as the input where n is the number of integers in our array. This 'n' is stored in \$t1.
- Then input the address of the base of our input array. It'll be stored in \$t2.

- While $i < n-1$, the program continues inside the outerloop, incrementing the value of 'i' by 1. Else it branches to outerlooppend and exits the loop.
- While $j < n-i-1$, the program continues inside the innerloop, incrementing the value of 'j' by 1. Else it branches to the innerloopend and exits the inner loop until the 'i' is incremented.
- The swap function is called if $arr[j] > arr[j+1]$. It swaps the numbers stored int j and j+1 location in the input array.
- The numbers are sorted in the same input array and our original array is printed out as a result.

Storing the inputs(12, 23, 4, 8, 44) in the memory:

The screenshot displays the Mars MIPS simulator interface. The 'Text Segment' window shows assembly code for a bubble sort algorithm. The 'Data Segment' window shows the memory layout where the input array is stored. The 'Registers' window shows the current state of the MIPS registers.

Text Segment:

Addr	Code	Basic	Source
4194336	0xad4c0000	sw \$12,0(\$10)	25: sw \$12,0(\$10)
4194340	0x214a0004	addi \$10,\$10,4	26: addi \$10,\$10,4
4194344	0x22f70001	addi \$23,\$23,1	27: addi \$23,\$23,1
4194348	0x00100006	j 4194320	28: j loop1
4194352	0x00185021	addu \$10,\$0,\$24	29: loopend: move \$12,\$18
4194356	0x0000b021	addu \$23,\$0,\$0	32: move \$s7,\$zero
4194360	0x0000b021	addu \$22,\$0,\$0	33: move \$s6,\$zero
4194364	0x20010001	addi \$1,\$0,1	34: sub1 \$t1,\$t1,1
4194368	0x01214822	sub \$9,\$9,\$1	

Data Segment:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	12	23	4	8	44	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0

Registers:

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	44
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	268501136
\$t3	11	0
\$t4	12	44
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$t8	16	0
\$t9	17	0
\$s0	18	0
\$s1	19	0
\$s2	20	0
\$s3	21	0
\$s4	22	0
\$s5	23	0
\$s6	24	0
\$s7	25	4
\$s8	26	268501120
\$s9	27	0
\$s10	28	268468224
\$s11	29	2147479548
\$s12	30	0
\$s13	31	4194336
\$pc		4194340
\$hi		0
\$lo		0

Mars Messages:

```

6
268501120
12
23
4
8
44
  
```

The output:

The screenshot shows the Mars MIPS simulator interface. The Text Segment window displays assembly code with addresses from 4194304 to 4194336. The Data Segment window shows memory values at addresses 268500992 to 268501184. The Registers window on the right shows the state of MIPS registers, with \$t2 containing 268501144. The Mars Messages window at the bottom shows the output of the program, which is 'program is finished running'.

Input With Negative Numbers (-2, -18, 5, 0, 11):

The screenshot shows the Mars MIPS simulator interface. The Text Segment window displays assembly code with addresses from 4194304 to 4194336. The Data Segment window shows memory values at addresses 268501024 to 268501216. The Registers window on the right shows the state of MIPS registers, with \$t2 containing 268501140. The Mars Messages window at the bottom shows the output of the program, which is 'program is finished running'.

Input With Duplicate Numbers (-7, 11, 5, 5, 11):

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100033	jal	4194508
	4194308	0x00c4821	addu	\$9,\$0,\$12
	4194312	0x0c100033	jal	4194508
	4194316	0x00c5021	addu	\$10,\$0,\$12
	4194320	0x000ac021	addu	\$24,\$0,\$10
	4194324	0x0000b21	addu	\$23,\$0,\$0
	4194328	0x12e9005	beq	\$23,\$9,\$5
	4194332	0x0c100033	jal	4194508
	4194336	0xad4c0000	sw	\$12,0(\$10)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	10	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	-7	5	5	11	11	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268500992
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	5
\$t2	10	268501140
\$t3	11	0
\$t4	12	11
\$t5	13	0
\$t6	14	268501124
\$t7	15	0
\$s0	16	0
\$s1	17	5
\$s2	18	-7
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	5
\$t8	24	268501120
\$t9	25	1
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194488
pc		4194508
hi		0
lo		0

Mars Messages

```

5
268501120
-7
11
5
5
11
-7
5
5
11
-- program is finished running --

```

Input With Large Numbers (192837465, 918273645, 555555555):

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100033	jal	4194508
	4194308	0x00c4821	addu	\$9,\$0,\$12
	4194312	0x0c100033	jal	4194508
	4194316	0x00c5021	addu	\$10,\$0,\$12
	4194320	0x000ac021	addu	\$24,\$0,\$10
	4194324	0x0000b21	addu	\$23,\$0,\$0
	4194328	0x12e9005	beq	\$23,\$9,\$5
	4194332	0x0c100033	jal	4194508
	4194336	0xad4c0000	sw	\$12,0(\$10)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	10	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	192837465	555555555	918273645	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268500992
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	3
\$t2	10	268501132
\$t3	11	0
\$t4	12	918273645
\$t5	13	0
\$t6	14	268501124
\$t7	15	0
\$s0	16	0
\$s1	17	555555555
\$s2	18	192837465
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	3
\$t8	24	268501120
\$t9	25	1
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194488
pc		4194508
hi		0
lo		0

Mars Messages

```

3
268501120
192837465
918273645
555555555
555555555
555555555
918273645
-- program is finished running --

```

Sum Of Three Consecutive Elements In An Array

C equivalent algorithm:

```
int A[3*N] = {a list of 3*N values}; int B[N];
int i, j=0;
void main (void)
{
for (i=0; i<N; i++){ B[i] = res_triplet(A,j); j=j+3;
}
}
```

```
int res_triplet(int V[ ], int pos) {
int i, sum=0;
for (i=0; i<3; i++)

sum = sum + V[pos+i]; sum=abs(sum);
return sum;
}
```

- Function `abs(int x)` returns the absolute value of its input argument.

The MIPS code:

- The program first takes 'n' as the input where n is the number of integers in our array. This 'n' is stored in \$t1.
- Then input the address of the base of our input array. It'll be stored in \$t2.
- Then input the address of the base of our output array. It'll be stored in \$t3.
- After taking in the input final_arr_loop is executed till 'i' is less than 'n'.
- res_triplet function is called in the final_arr_loop in each iteration. It sums three consecutive elements and returns the sum's absolute value.
- The absolute values are stored in \$t3's value's memory location.
- After the completion of the final_arr_loop, the n numbers who's starting address is given by \$t3 are printed.

Storing the inputs (2, 3, 4, 1, 8, 9) in the memory:

Text Segment

Bkpt	Address	Code	Basic	Source
	4194344	0x0ad4c0000	sw \$t2,0(\$t0)	25: sw \$t4,0(\$t2)
	4194348	0x214a0004	addi \$t0,\$t0,4	26: addi \$t2,\$t2,4
	4194352	0x22f70001	addi \$t3,\$t3,1	27: addi \$s7,\$s7,1
	4194356	0x00100008	j 4194336	28: j loop1
	4194360	0x00185021	addu \$t0,\$t0,\$t4	29: loop1end: move \$t2,\$t8
	4194364	0x0000b021	addu \$t3,\$t0,\$t0	31: move \$s7, \$zero
	4194368	0x0000b021	addu \$t4,\$t0,\$t1	32: move \$t8,\$t3
	4194372	0x12e90006	beq \$t3,\$t9,6	33: final_arr_loop: beq \$s7,\$t1,loop1end2
	4194376	0x0c100030	jal 4194496	34: jal res_triplet

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	2	3	4	1	8	9	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	9
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	6
\$t2	10	268501140
\$t3	11	268501184
\$t4	12	9
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	5
\$t8	24	268501120
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194344
pc		4194348
hi		0
lo		0

Mars Messages

```

Reset: reset completed.
6
268501120
268501184
2
3
4
1
8
9

```

The output:

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100023	jal 4194444	6: jal input_int
	4194308	0x000c4821	addu \$9,\$t0,\$t2	7: move \$t1,\$t4
	4194312	0x0c100023	jal 4194444	11: jal input_int
	4194316	0x000c5021	addu \$t0,\$t0,\$t2	12: move \$t2,\$t4
	4194320	0x0c100023	jal 4194444	16: jal input_int
	4194324	0x000c5021	addu \$t1,\$t0,\$t2	17: move \$t3,\$t4
	4194328	0x000ac021	addu \$t4,\$t0,\$t0	21: move \$t8,\$t2
	4194332	0x0000b021	addu \$t3,\$t0,\$t0	22: move \$s7,\$zero #1 = 0
	4194336	0x12e90005	beq \$t3,\$t9,5	23: loop1: beq \$s7,\$t1,loop1end

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	10	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	2	3	4	1	8	9	0	0
268501152	0	0	0	0	0	0	0	0
268501184	9	18	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268500992
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	6
\$t2	10	268501144
\$t3	11	268501192
\$t4	12	18
\$t5	13	8
\$t6	14	17
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	6
\$t8	24	268501184
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194424
pc		4194444
hi		0
lo		0

Mars Messages

```

6
268501120
268501184
2
3
4
1
8
9
18
-- program is finished running --

```


Input With Negative Numbers (-2, -18, 5, 0, 11):

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100023	jal	4194444
	4194308	0x000c4821	addu	\$9,\$0,\$12
	4194312	0x0c100023	jal	4194444
	4194316	0x000c5021	addu	\$10,\$0,\$12
	4194320	0x0c100023	jal	4194444
	4194324	0x000c5821	addu	\$11,\$0,\$12
	4194328	0x000ac021	addu	\$24,\$0,\$10
	4194332	0x0000b021	addu	\$23,\$0,\$0
	4194336	0x12e90005	beq	\$23,\$9,5
				23: loop1: beq \$57,\$t1, loop1end

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	10	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	-7	-2	2	4	2	-1	0	0
268501152	0	0	0	0	0	0	0	0
268501184	7	5	0	0	0	0	0	0

Ox10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII

Mars MessagesRun I/O

```

6
268501120
268501184
-7
-2
2
4
2
-1
7
5
-- program is finished running --

```

Clear

RegistersCoproc 1Coproc 0

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268500992
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	6
\$t2	10	268501144
\$t3	11	268501192
\$t4	12	5
\$t5	13	2
\$t6	14	1
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	6
\$t8	24	268501184
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194444
pc		4194444
hi		0
lo		0

Instructions to run the RV-FPGA program files

- Make sure you are in the directory which contains the file

```
IMT2021018 IMT2021065_Q2B.C  
IMT2021018 IMT2021065_Q2B.S
```

- Execute the following command in VSCODE:

```
run on VSCODE
```

- ADD 'debug_tool=whisper' in .ini file.

Bubble Sort

C equivalent algorithm:

```
for (i = 0; i < n - 1; i++){
    for (j = 0; j < n - i - 1; j++)
    {
        if (arr[j] > arr[j + 1])
        {
            swap(arr[j], arr[j + 1]);
        }
    }
}
```

The RV-FPGA code:

- We implement bubbleSort for N=10 numbers.

V={ 3 , 1 , 4 , 2 , 5 , 10 , 7 , 6 , 9 , 8}

- We define a memory location:

```
#define GPIO_LEDS 0x80001404
```

This is the base address of array V.

- We put array inputs in memory location using WRITE_GPIO.

```
#address = 0x80001404
```

```
#WRITE_GPIO(address,V[i])
```

The command stores V[i] in address location

'address'

- After sorting, store the numbers in memory location '80001414'.

```
#address = 0x80001414
```

```
#WRITE_GPIO(address,V[i])
```

- We print array in terminal using printfNexys().

Storing the inputs in the memory

1	Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2	80001400: 00 00 00 00 03 01 04 02 05 0A 07 06 09 08 00 00
3	80001410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4	80001420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5	80001430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6	80001440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
7	80001450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8	80001460: 00 00 00 00
9	

After sorting, the sorted numbers are stored in memory location with base address 80001414.

1	Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	80001400:	00	00	00	00	03	01	04	02	05	0A	07	06	09	08	00	00
3	80001410:	00	00	00	00	01	02	03	04	05	06	07	08	09	0A	00	00
4	80001420:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	80001430:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	80001440:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	80001450:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	80001460:	00	00	00	00												

Local Variables before sorting

VARIABLES	
Local	
V: [10]	
0:	3
1:	1
2:	4
3:	2
4:	5
5:	10
6:	7
7:	6
8:	9
9:	8
i:	0
address:	-2147478524
> Global	
> Static	

Local Variables after sorting

VARIABLES	
Local	
V: [10]	
0:	1
1:	2
2:	3
3:	4
4:	5
5:	6
6:	7
7:	8
8:	9
9:	10
i:	10
address:	-2147478498
> Global	
> Static	

Finding Gcd

The RV-FPGA code:

- We first take 10 inputs. t0 register stores the memory location(PC value). Register t1 and t2 store the inputs a and b on which we need to perform the operation.

- We define a memory location:

```
#li t0 0x80001400
```

The value a is stored in 0(t0) and value b is stored in 4(t0).

Then we update the value of t0 by incrementing with 10.

- After all the inputs are done, we reset t0 back to initial address 0x80001400.

- We then enter a for loop which iterates 10 times. t5 register stores 0 and s1 register stores value 10. We increment t5 after calculating GCD of each pair.

```
#addi t5, t5, 0
```

- Inside the for loop, we have a while loop which calculates the remainder of a and b and stores it in another register t3.

```
#rem t3, t1, t2
```

- If the remainder (in t3) is equal to 0, we jump outside the while loop and the GCD is the value stored in t1. This is then stored in memory location 8(t0), else we continue.

```
#beq t3, zero, DONE
```

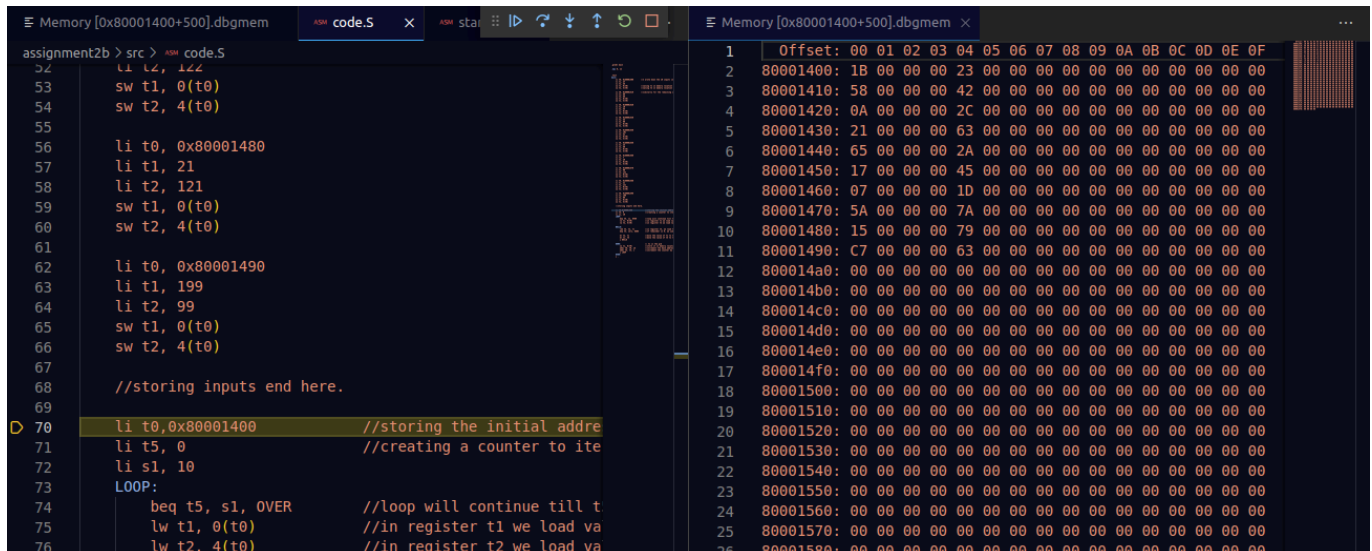
- We store the value of t2 register in t1 and t3(remainder) in t2.

```
#mv t1, t2      stores value of t2 in t1
```

```
#mv t2, t3      stores value of t3 in t2
```

- We repeat the loop till we reach satisfy the bee condition. Once we find the GCD, we increment PC by 10 and do the same for the next set of inputs in 0(t0) and 4(t0) and save it in 8(t0).

Storing The Input In Memory.

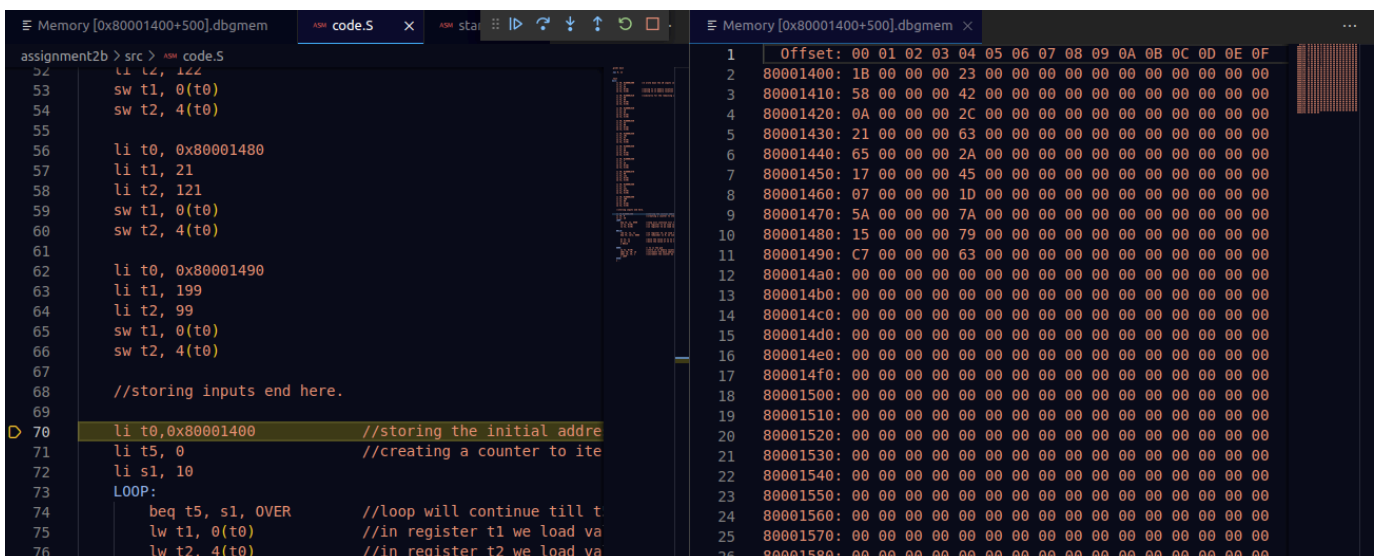


The screenshot shows a debugger window with two panes. The left pane displays assembly code for a program named 'assignment2b'. The code is in MIPS assembly and includes instructions for loading constants, storing values to memory, and a loop. The right pane shows a memory dump starting at address 80001400, displaying hexadecimal values for each byte.

```
assignment2b > src > asm code.S
52      li t2, 122
53      sw t1, 0(t0)
54      sw t2, 4(t0)
55
56      li t0, 0x80001480
57      li t1, 21
58      li t2, 121
59      sw t1, 0(t0)
60      sw t2, 4(t0)
61
62      li t0, 0x80001490
63      li t1, 199
64      li t2, 99
65      sw t1, 0(t0)
66      sw t2, 4(t0)
67
68      //storing inputs end here.
69
70      li t0, 0x80001400 //storing the initial addre
71      li t5, 0          //creating a counter to ite
72      li s1, 10
73      LOOP:
74      beq t5, s1, OVER //loop will continue till t
75      lw t1, 0(t0)     //in register t1 we load va
76      lw t2, 4(t0)     //in register t2 we load va
```

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	80001400	1B	00	00	00	23	00	00	00	00	00	00	00	00	00	00
2	80001410	58	00	00	00	42	00	00	00	00	00	00	00	00	00	00
3	80001420	0A	00	00	00	2C	00	00	00	00	00	00	00	00	00	00
4	80001430	21	00	00	00	63	00	00	00	00	00	00	00	00	00	00
5	80001440	65	00	00	00	2A	00	00	00	00	00	00	00	00	00	00
6	80001450	17	00	00	00	45	00	00	00	00	00	00	00	00	00	00
7	80001460	07	00	00	00	1D	00	00	00	00	00	00	00	00	00	00
8	80001470	5A	00	00	00	7A	00	00	00	00	00	00	00	00	00	00
9	80001480	15	00	00	00	79	00	00	00	00	00	00	00	00	00	00
10	80001490	C7	00	00	00	63	00	00	00	00	00	00	00	00	00	00
11	800014a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
12	800014b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
13	800014c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
14	800014d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
15	800014e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
16	800014f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
17	80001500	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
18	80001510	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
19	80001520	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	80001530	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
21	80001540	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
22	80001550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
23	80001560	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
24	80001570	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
25	80001580	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Storing The Input In Memory.



This screenshot is identical to the one above, showing the same assembly code and memory dump in a debugger window.