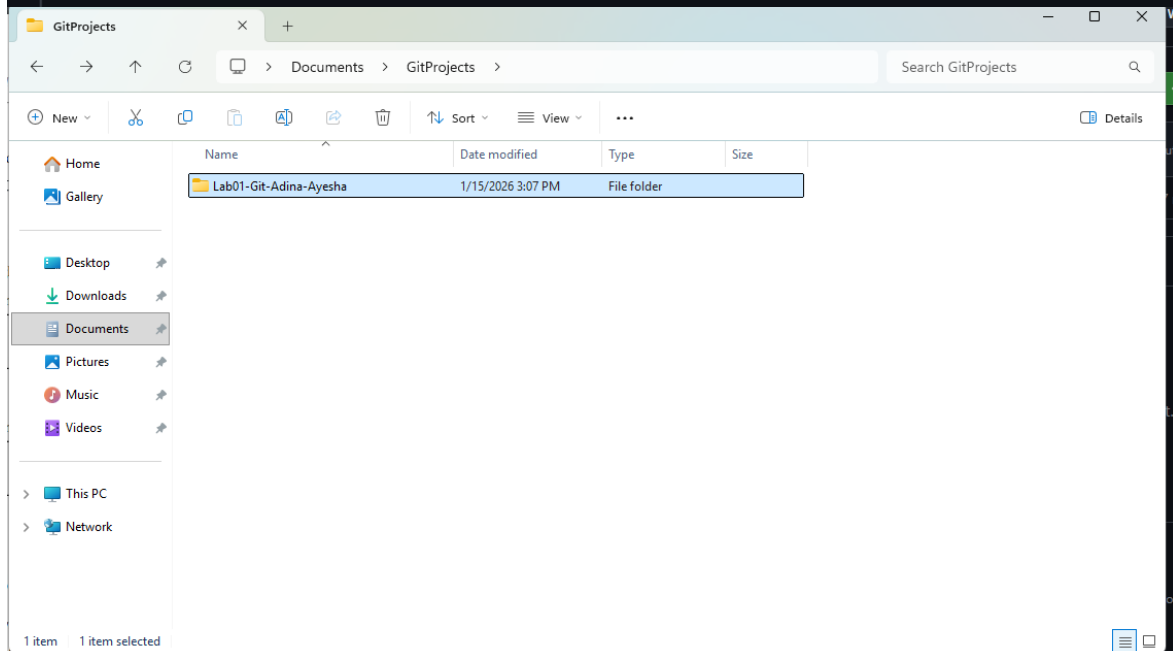
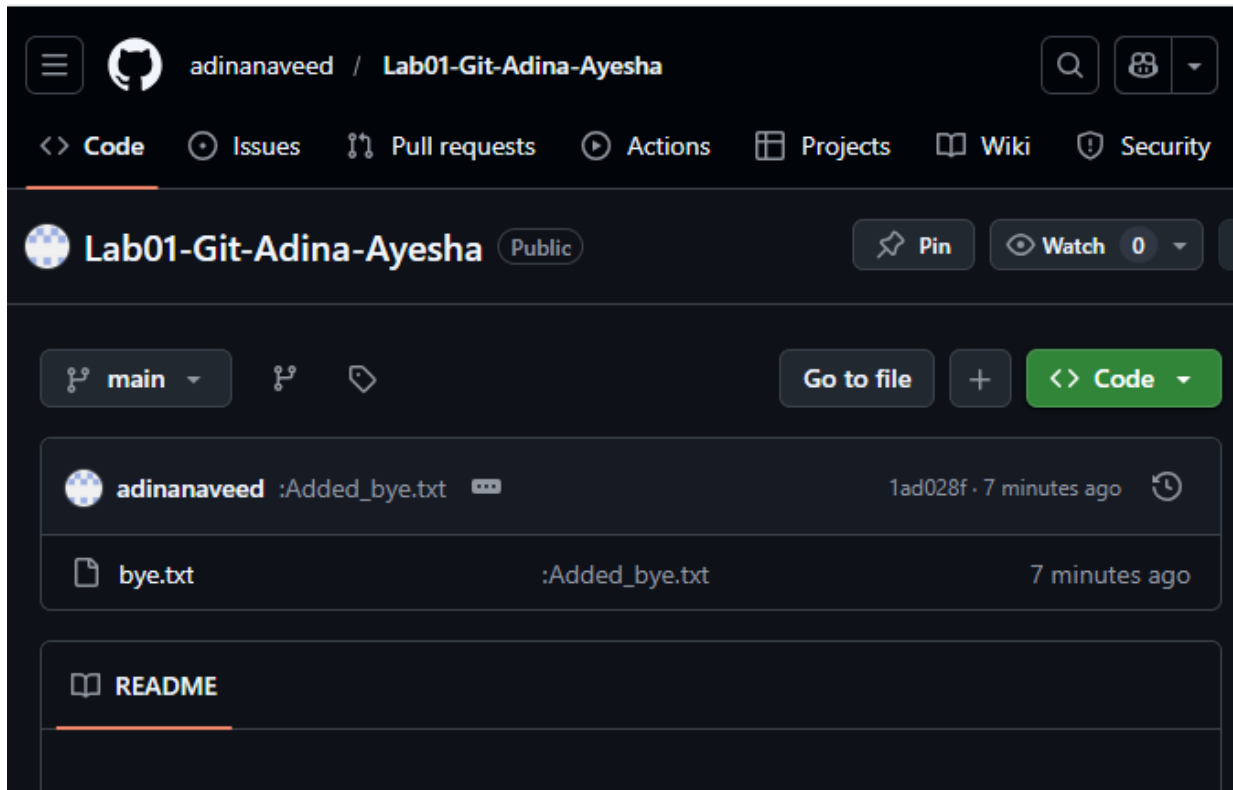


Lab 1: Getting Started with RISC-V (Assembly Language) in VS Code

Task 1:





Lab01-Git-Adina-Ayesha

Documents > GitProjects > Lab01-Git-Adina-Ayesha

Search Lab01-Git-Adina-Ayesha

New | Cut | Copy | Paste | Delete | Sort | View | Details

Name	Date modified	Type	Size
bye	1/15/2026 3:07 PM	Text Document	1 KB
hello	1/15/2026 3:06 PM	Text Document	0 KB

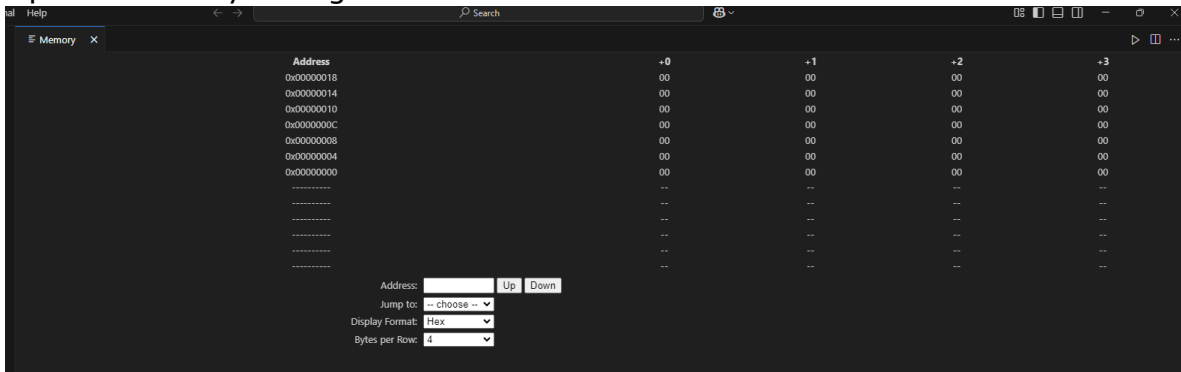
Home | Gallery | Desktop | Downloads | Documents | Pictures | Music | Videos | This PC | Network

2 items



Task 02: Setting Up VS Code (RISC-V Simulation Environment)

Opened memory through command:



Typed code with register:

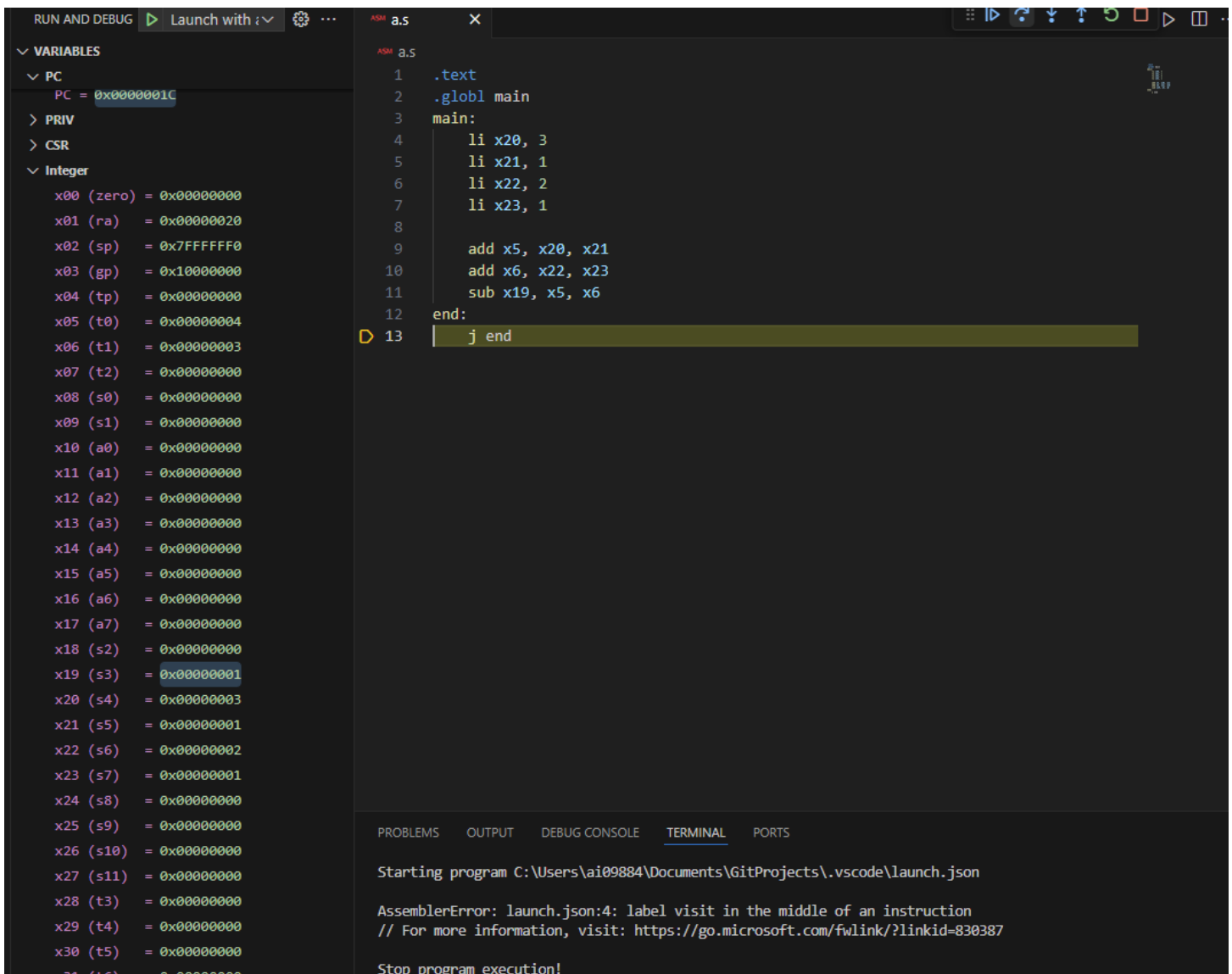


Figure 1.4: RISC-V Memory

Task 3

Convert the following statement to RISC V. You can use the same registers as given in

```
1 int a = 5;
2 int b = 0 + 0;
3 a = b + 32;
4 int d = (a + b) - 5;
5 int e = (((a - d) + (b - a)) + d);
e = a + b + d + e;
```

Code

```
ASM task3.s
1  .text
2  .globl main
3  main:
4      li x11, 5 #a= 5
5      addi x12, x12, 0 #b= 0+0
6      li x13, 32
7      add x11, x13, x12 #a=b+32
8
9      add x14, x11, x12 # d = a+b
10     addi x14, x14, -5 # d = (a+b) - 5
11
12     sub x5, x11, x14 #a-d
13     sub x6, x12, x11 #b-a
14     add x15, x5, x6 #e = (a-d) + (b-a)
15     add x15, x15, x14 # e = ((a-d) + (b-a)) + d
16
17     add x15, x15, x11 # e = a+e
18     add x15, x15, x12 # e = a+e+b
19     add x15, x15, x14 # e = a+e+b+d
20 end:
21     j end
```

Decimal=5 . Hex=5

Dec=0 , Hex=0;

d=27 → Hex = 1B

a = 32 → hex = 20

e = 59 → hex = 3B



Output

The screenshot shows the Visual Studio Code interface with the 'RUN AND DEBUG' window open. The window displays a list of variables and their values, organized into sections: VARIABLES, PC, PRIV, CSR, and Integer. The Integer section lists registers x00 through x31, with their corresponding values in hexadecimal. The PC register is set to 0x00000034. The PRIV and CSR sections are currently collapsed. The Integer section is expanded, showing the following values:

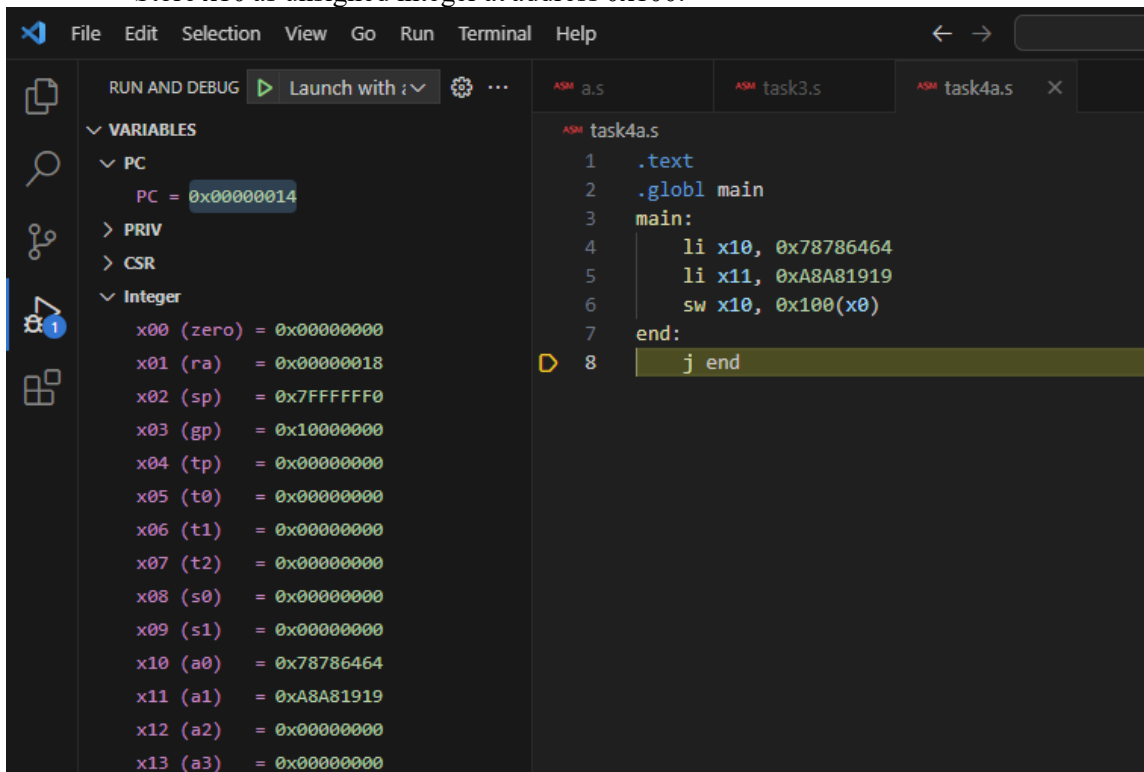
Register	Value
x00 (zero)	0x00000000
x01 (ra)	0x00000038
x02 (sp)	0x7FFFFFF0
x03 (gp)	0x10000000
x04 (tp)	0x00000000
x05 (t0)	0x00000005
x06 (t1)	0xFFFFFE0
x07 (t2)	0x00000000
x08 (s0)	0x00000000
x09 (s1)	0x00000000
x10 (a0)	0x00000000
x11 (a1)	0x00000020
x12 (a2)	0x00000000
x13 (a3)	0x00000020
x14 (a4)	0x0000001B
x15 (a5)	0x0000003B
x16 (a6)	0x00000000
x17 (a7)	0x00000000
x18 (s2)	0x00000000
x19 (s3)	0x00000000
x20 (s4)	0x00000000
x21 (s5)	0x00000000
x22 (s6)	0x00000000
x23 (s7)	0x00000000
x24 (s8)	0x00000000
x25 (s9)	0x00000000
x26 (s10)	0x00000000
x27 (s11)	0x00000000
x28 (t3)	0x00000000
x29 (t4)	0x00000000
x30 (t5)	0x00000000
x31 (t6)	0x00000000

Task 4a

Initialize the register x10 and x11 with values 0x78786464, 0xA8A81919, respectively manually.

Write the RISC-V assembly code for each item below. Try guessing the result in each destination before executing the instruction and corroborate it after execution:

Store x10 as unsigned integer at address 0x100.



The screenshot shows the RISC-V IDE with the assembly code for task4a.s and the register values.

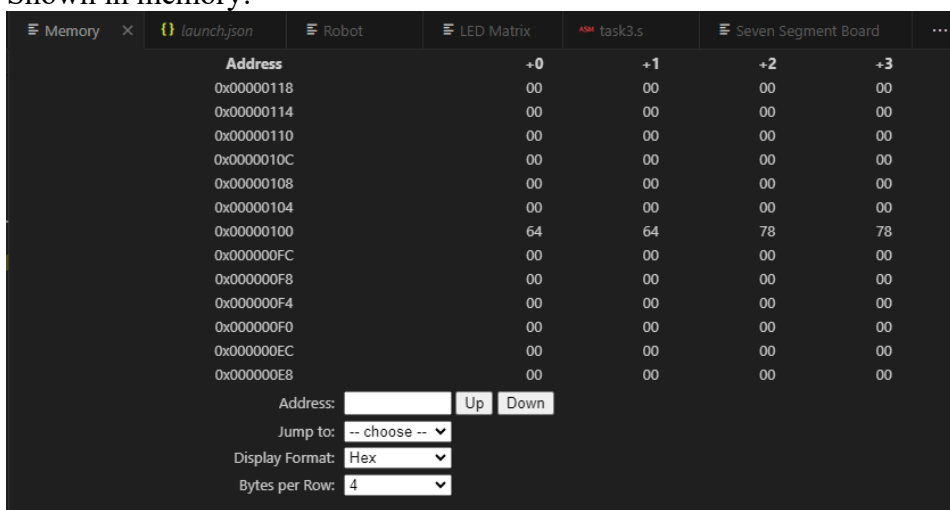
Assembly Code (task4a.s):

```
1 .text
2 .globl main
3 main:
4     li x10, 0x78786464
5     li x11, 0xA8A81919
6     sw x10, 0x100(x0)
7 end:
8 j end
```

Register Values:

Register	Value
x00 (zero)	0x00000000
x01 (ra)	0x00000018
x02 (sp)	0x7FFFFFF0
x03 (gp)	0x10000000
x04 (tp)	0x00000000
x05 (t0)	0x00000000
x06 (t1)	0x00000000
x07 (t2)	0x00000000
x08 (s0)	0x00000000
x09 (s1)	0x00000000
x10 (a0)	0x78786464
x11 (a1)	0xA8A81919
x12 (a2)	0x00000000
x13 (a3)	0x00000000

Shown in memory:



The screenshot shows the memory view with the following data:

Address	+0	+1	+2	+3
0x00000118	00	00	00	00
0x00000114	00	00	00	00
0x00000110	00	00	00	00
0x0000010C	00	00	00	00
0x00000108	00	00	00	00
0x00000104	00	00	00	00
0x00000100	64	64	78	78
0x000000FC	00	00	00	00
0x000000F8	00	00	00	00
0x000000F4	00	00	00	00
0x000000F0	00	00	00	00
0x000000EC	00	00	00	00
0x000000E8	00	00	00	00

Address: Up Down

Jump to:

Display Format:

Bytes per Row:



Store x11 as unsigned integer at address 0x1F0.

The screenshot shows an IDE with the following components:

- File Explorer:** Shows files `a.s`, `task3.s`, and `task4a.s`.
- Assembly Editor:** Displays the assembly code for `task4a.s`:

```
1 .text
2 .globl main
3 main:
4     li x10, 0x78786464
5     li x11, 0xA8A81919
6     sw x11, 0x1F0(x0)
7 end:
8 j end
```
- Variables Panel:** Lists the state of registers and memory locations:
 - `PC = 0x00000014`
 - `PRIV`
 - `CSR`
 - Integer:**
 - `x00 (zero) = 0x00000000`
 - `x01 (ra) = 0x00000018`
 - `x02 (sp) = 0x7FFFFFF0`
 - `x03 (gp) = 0x10000000`
 - `x04 (tp) = 0x00000000`
 - `x05 (t0) = 0x00000000`
 - `x06 (t1) = 0x00000000`
 - `x07 (t2) = 0x00000000`
 - `x08 (s0) = 0x00000000`
 - `x09 (s1) = 0x00000000`
 - `x10 (a0) = 0x78786464`
 - `x11 (a1) = 0xA8A81919`
 - `x12 (a2) = 0x00000000`

Shown in memory:

The screenshot shows a memory viewer window with the following data:

Address	+0	+1	+2	+3
0x00000208	00	00	00	00
0x00000204	00	00	00	00
0x00000200	00	00	00	00
0x000001FC	00	00	00	00
0x000001F8	00	00	00	00
0x000001F4	00	00	00	00
0x000001F0	91	91	81	8A
0x000001EC	00	00	00	00
0x000001E8	00	00	00	00
0x000001E4	00	00	00	00
0x000001E0	00	00	00	00
0x000001DC	00	00	00	00
0x000001D8	00	00	00	00

Below the table, there are controls for the memory viewer:

- Address: Up Down
- Jump to: -- choose --
- Display Format: Hex
- Bytes per Row: 4



Load an unsigned short integer (two bytes) from address 0x100 in x12.

The screenshot shows the VS Code IDE with the assembly code for `task4a.s` open. The code is as follows:

```
1 .text
2 .globl main
3 main:
4     li x10, 0x78786464
5     li x11, 0xA8A819191
6     sw x10, 0x100(x0)
7     sw x11, 0x1F0(x0)
8     lhu x12, 0x100(x0)
9 end:
10 j end
```

The VARIABLES pane on the left shows the state of the registers. The value of `x12 (a2)` is `0x00006464`, which is the result of the `lhu` instruction loading the short integer from address `0x100` in register `x0`.

Register	Value
x0 (zero)	0x00000000
x1 (ra)	0x00000020
x2 (sp)	0x7FFFFFF0
x3 (gp)	0x10000000
x4 (tp)	0x00000000
x5 (t0)	0x00000000
x6 (t1)	0x00000000
x7 (t2)	0x00000000
x8 (s0)	0x00000000
x9 (s1)	0x00000000
x10 (a0)	0x78786464
x11 (a1)	0xA8A819191
x12 (a2)	0x00006464
x13 (a3)	0x00000000
x14 (a4)	0x00000000

Load a short integer from address 0x1F0 in register x13.

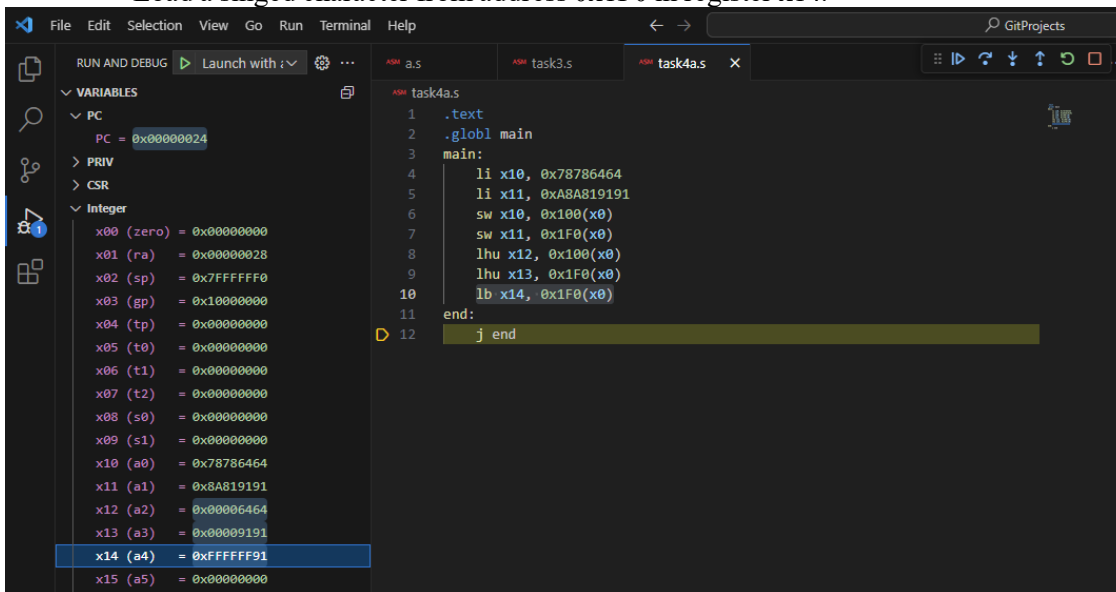
The screenshot shows the VS Code IDE with the assembly code for `task4a.s` open. The code is as follows:

```
1 .text
2 .globl main
3 main:
4     li x10, 0x78786464
5     li x11, 0xA8A819191
6     sw x10, 0x100(x0)
7     sw x11, 0x1F0(x0)
8     lhu x12, 0x100(x0)
9     lhu x13, 0x1F0(x0)
10 end:
11 j end
```

The VARIABLES pane on the left shows the state of the registers. The value of `x13 (a3)` is `0x00009191`, which is the result of the `lhu` instruction loading the short integer from address `0x1F0` in register `x0`.

Register	Value
x0 (zero)	0x00000000
x1 (ra)	0x00000024
x2 (sp)	0x7FFFFFF0
x3 (gp)	0x10000000
x4 (tp)	0x00000000
x5 (t0)	0x00000000
x6 (t1)	0x00000000
x7 (t2)	0x00000000
x8 (s0)	0x00000000
x9 (s1)	0x00000000
x10 (a0)	0x78786464
x11 (a1)	0xA8A819191
x12 (a2)	0x00006464
x13 (a3)	0x00009191
x14 (a4)	0x00000000

Load a signed character from address 0x1F0 in register x14.



The screenshot shows the RISC-V IDE with the assembly code for task4a.s. The code is as follows:

```

1 .text
2 .globl main
3 main:
4     li x10, 0x78786464
5     li x11, 0xA8A819191
6     sw x10, 0x100(x0)
7     sw x11, 0x1F0(x0)
8     lhu x12, 0x100(x0)
9     lhu x13, 0x1F0(x0)
10    lb x14, 0x1F0(x0)
11 end:
12 j end

```

The register file on the left shows the following values:

Register	Value
x00 (zero)	0x00000000
x01 (ra)	0x00000028
x02 (sp)	0x7FFFFFF0
x03 (gp)	0x10000000
x04 (tp)	0x00000000
x05 (t0)	0x00000000
x06 (t1)	0x00000000
x07 (t2)	0x00000000
x08 (s0)	0x00000000
x09 (s1)	0x00000000
x10 (a0)	0x78786464
x11 (a1)	0xA8A819191
x12 (a2)	0x00006464
x13 (a3)	0x00009191
x14 (a4)	0xFFFFF91
x15 (a5)	0x00000000

*

Task 4b -- Loop unrolling

Assume there are three character arrays a, b, and c located at addresses 0x100, 0x200, 0x300 respectively.

```

for (int i=0 ; i<4; i++ )
c [ i ]=a [ i ]+b [ i ] ; # c [ 0 ]=a [ 0 ]+b [ 0 ] ;

```

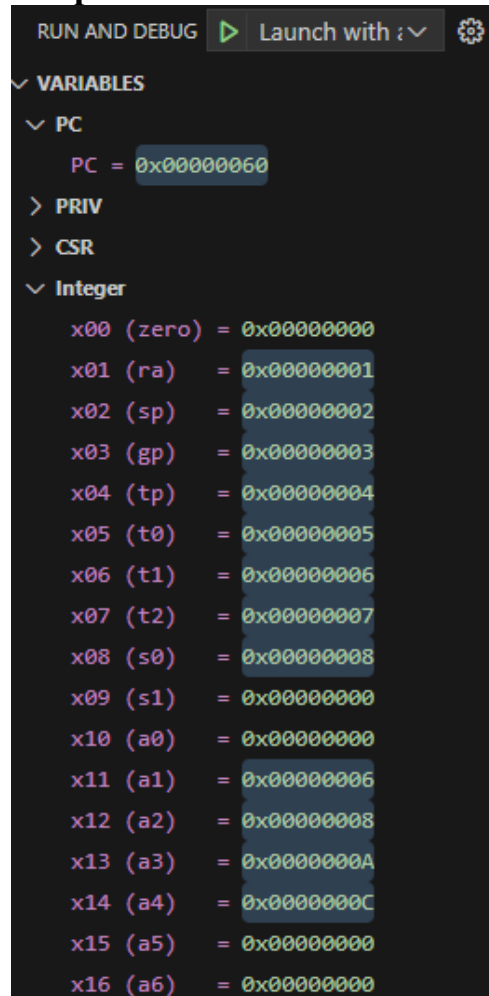
Write equivalent RISC-V code for the piece of code given. You have not studied loops yet, but the above code is manageable without loop instructions. Also assume that A is a character array, B is a short array, and C is an unsigned integer array.

Code

```
task4b.s x
ASM task4b.s
1 .text
2 .globl main
3 main:
4     #a
5     li x1, 1
6     li x2, 2
7     li x3, 3
8     li x4, 4
9     #b
10    li x5, 5
11    li x6, 6
12    li x7, 7
13    li x8, 8
14    #storing a
15    sw x1, 0x100(x0)
16    sw x2, 0x101(x0)
17    sw x3, 0x102(x0)
18    sw x4, 0x103(x0)
19    #storing b
20    sw x5, 0x200(x0)
21    sw x6, 0x202(x0)
22    sw x7, 0x204(x0)
23    sw x8, 0x206(x0)
24    # calculating c
25    add x11, x1, x5
26    add x12, x2, x6
27    add x13, x3, x7
28    add x14, x4, x8
29    # storing c
30    sw x11, 0x300(x0)
31    sw x12, 0x304(x0)
32    sw x13, 0x308(x0)
33    sw x14, 0x30C(x0)
34 end:
35 j end
36
37
```



Output



Stored in memory:

Memory		LED Matrix		Seven Seg		▶ □ ...	
Address	+0	+1	+2	+3			
0x00000118	00	00	00	00			
0x00000114	00	00	00	00			
0x00000110	00	00	00	00			
0x0000010C	00	00	00	00			
0x00000108	00	00	00	00			
0x00000104	00	00	00	00			
0x00000100	01	02	03	04			
0x000000FC	00	00	00	00			
0x000000F8	00	00	00	00			
0x000000F4	00	00	00	00			
0x000000F0	00	00	00	00			
0x000000EC	00	00	00	00			
0x000000E8	00	00	00	00			
Address:	<input type="text"/>	Up	Down				
Jump to:	-- choose --						
Display Format:	Hex						
Bytes per Row:	4						



Memory X

LED Matrix

Seven Seg > [] ...

Address	+0	+1	+2	+3
0x00000218	00	00	00	00
0x00000214	00	00	00	00
0x00000210	00	00	00	00
0x0000020C	00	00	00	00
0x00000208	00	00	00	00
0x00000204	07	00	08	00
0x00000200	05	00	06	00
0x000001FC	00	00	00	00
0x000001F8	00	00	00	00
0x000001F4	00	00	00	00
0x000001F0	00	00	00	00
0x000001EC	00	00	00	00
0x000001E8	00	00	00	00

Address: Up Down

Jump to: -- choose -- ▾

Display Format: Hex ▾

Bytes per Row: 4 ▾

Memory X

LED Matrix

Seven Segment Bo ...

Address: Up Down

Jump to: -- choose -- ▾

Display Format: Hex ▾

Bytes per Row: 4 ▾



Assessment Rubric

Lab 1: Getting Started with RISC-V (Assembly Language) in VS Code

Name: Ayesha Imran, Adina Naveed	Student ID: ai09884, an09842	section*: T3
----------------------------------	------------------------------	--------------

Points Distribution

	Task No.	LR 2 Code	LR 5 Results
In - Lab	Task 1	/0	/15
	Task 2	/0	/15
	Task 3	/10	/5
	Task 4a	/10	/5
	Task 4b	/10	/10
Total Points: 100		/30	/50
CLO Mapped		CLO 2	

Affective Domain Rubric		Points	CLO Mapped
AR7	Report Submission & Git Upload	/10 & /10	CLO 2

CLO	Total Points	Points Obtained
2	100	
Total	100	

For description of different levels of the mapped rubrics, please refer to the Lab Evaluation Assessment Rubrics and Affective Domain Assessment Rubrics provided here.

