Lab 04 – Worksheet

Name: Shaheer Qureshi, Areeba Izhar	ID: sq09647, ai09625	Section: T1

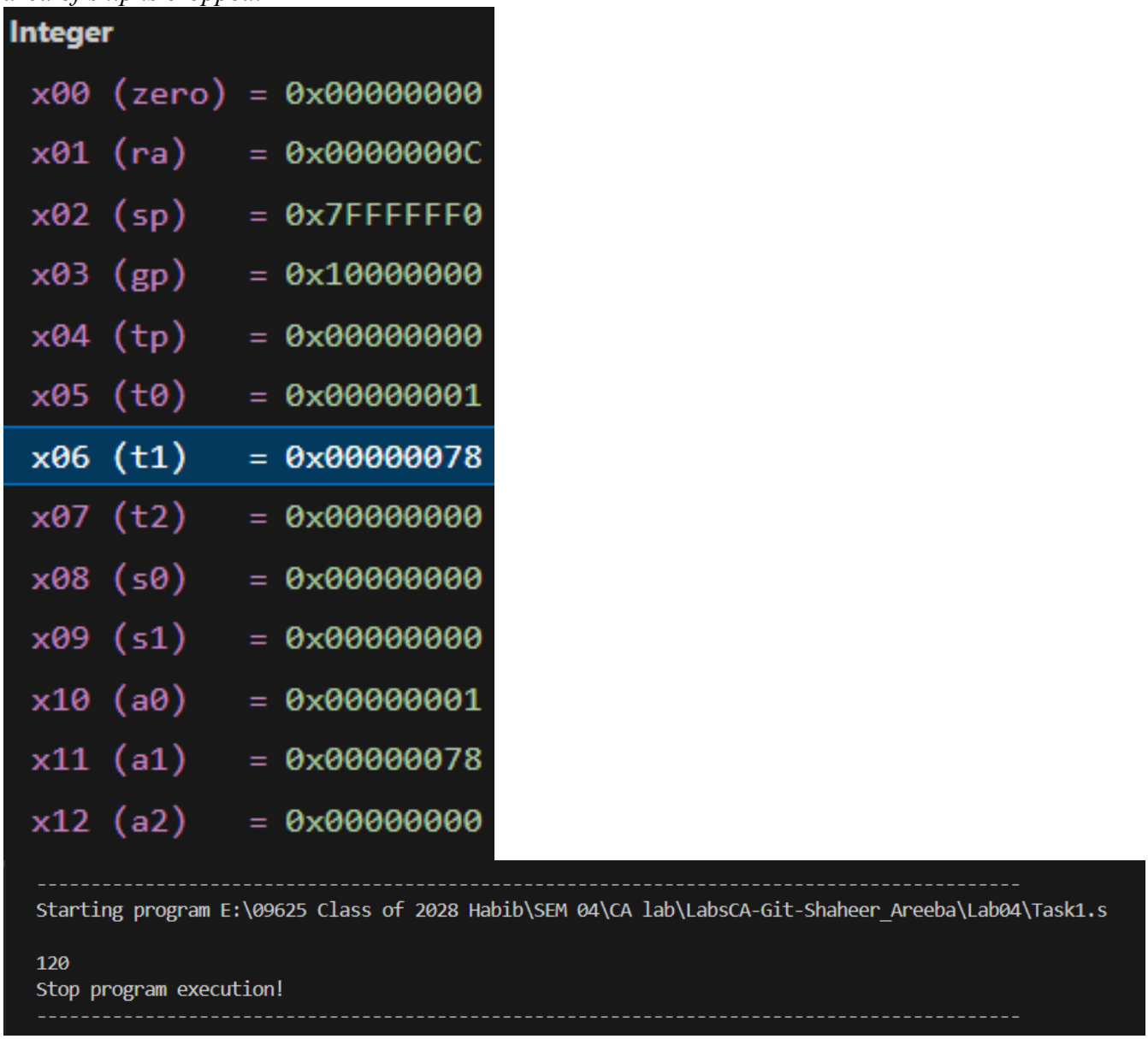
Note: Assumptions and logics should be explained separately in tasks after the task results.

Task 1

Provide appropriately commented codes.

```
1     .text
2     .globl main
3     main:
4         li x6, 1 #answer
5         li x10, 5 #n (120)
6         jal x1, loop #jump to loop
7         addi x11, x10,0
8         li x10, 1
9         ecall
10        j exit
11
12    loop:
13        li x5, 1 #1 to compare with the blt statement (if n<= 1)
14        blt x10, x5, end_loop
15        mul x6, x6, x10 #n- n*(n-1)
16        addi x10, x10, -1 #decrement n
17        j loop
18
19    end_loop:
20        addi x10, x6, 0 #store answer in x10
21        jalr x0, 0(x1) #return from main
22
23    end:
24        j end
25
26    exit:
27        j end
```

**Add snip of relevant Registers / Memory locations where you think results are obtained. Make sure the irrelevant area of snip is cropped. Make sure the irrelevant area of snip is cropped.*



Comments

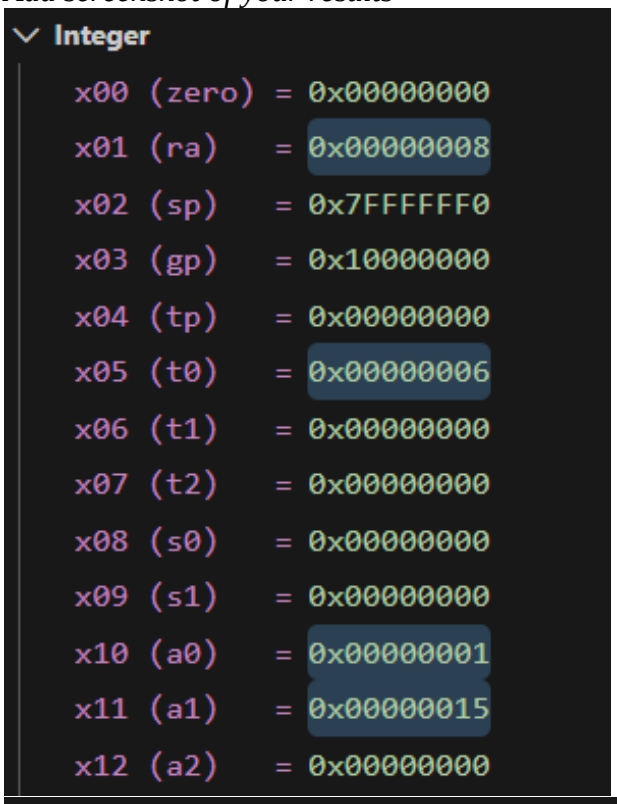
In the main loop, we stated the value of our n, and called for the loop. Loop compares if the value is the base case of recursive call. If not, it causes a decrement in value on n and stores its value in x6, calls loop again until n=1. Then it reaches the end_loop and from there the value is traced back in x10 back to all the older cases.

Task 2

Provide appropriately commented codes

```
1     .text
2     .globl main
3     main:
4         li x10, 6 #term to sum till #ans =21
5         jal x1, ntri #jump to ntri
6         addi x11, x10,0 #store answer in x11
7         li x10, 1
8         ecall
9         j exit
10
11    ntri:
12        li x5, 1
13        ble x10, x5, end_ntri
14        addi sp, sp, -8
15        sw x1, 4(sp) #store address on stack
16        sw x10, 0(sp) #store n on stack
17
18        addi x10, x10, -1 #decrement x10 by 1
19        jal x1, ntri #recursive call to ntri
20        lw x5, 0(sp) #restore address from stack
21        lw x1, 4(sp) #restore n from stack
22        addi sp, sp, 8 #restore stack pointer
23
24        add x10, x10, x5#add n to answer from recursive call
25        jalr x0, 0(x1) #return from ntri
26
27    end_ntri:
28        li x10, 1 #base case return 1
29        jalr x0, 0(x1) #return from ntri
30
31    end:
32        j end
33
34    exit:
35        j end
```

Add screenshot of your results




```
Starting program E:\09625 Class of 2028 Habib\SEM 04\CA lab\LabsCA-Git-Shaheer_Areeba\Lab04\Task2.s

21
Stop program execution!
```

Comments :

Since task 2 was based on iterative approach, we introduced a stack pointer.
We initialized a counter register (x5) to 1 and an accumulator register (like x11) to 0. When we jump to loop, it adds the counter in x11 and does increment counter by 1, and branches back to the start of the loop as long as the counter is less than or equal to our target x10. Once the counter exceeds , the loop terminates, and the total sum is saved in x11. With return address the value is printed from x11 in terminal using ecall.

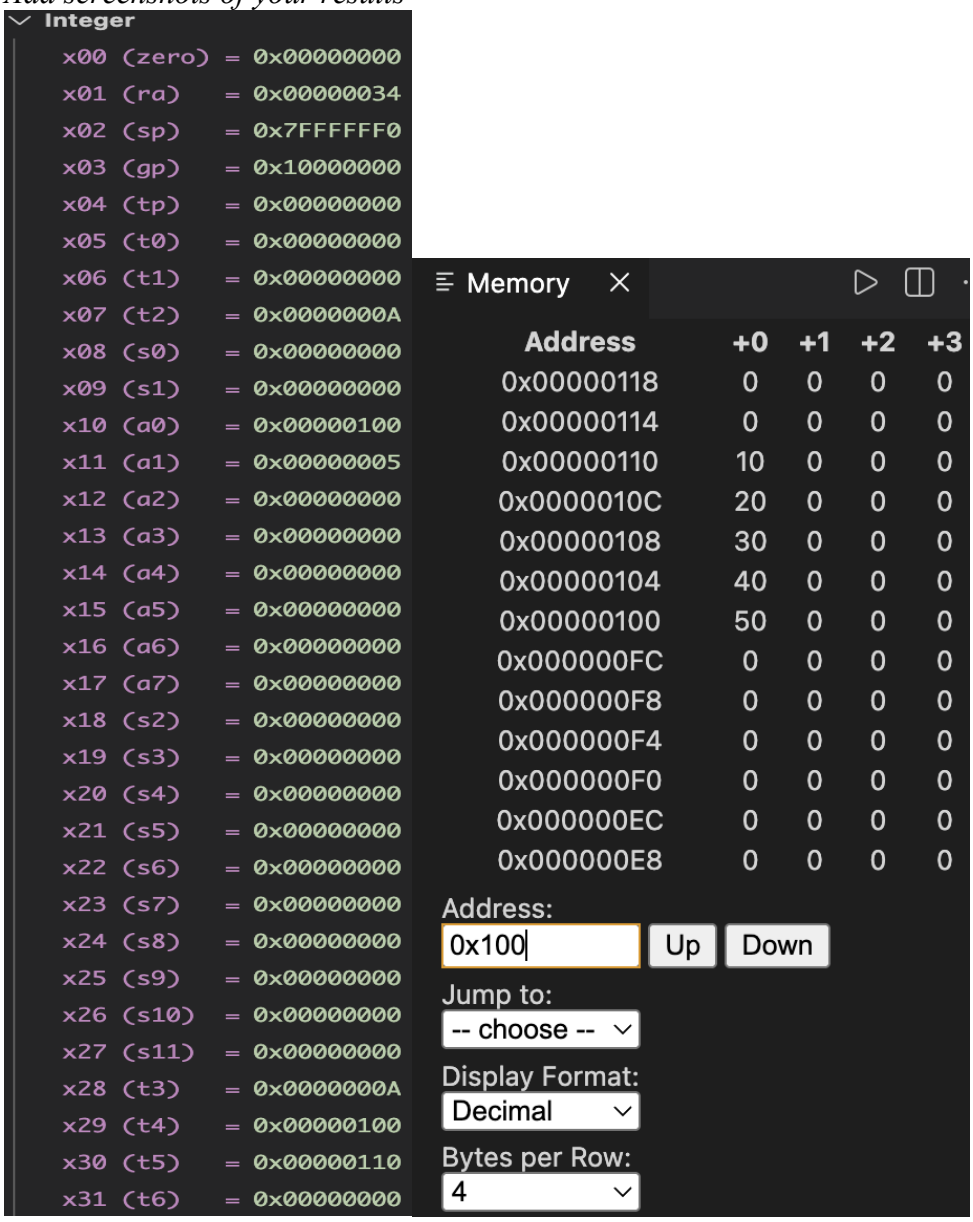
Task 3: Provide appropriately commented codes

```
1       .text
2       .globl main
3       main:
4           li x10, 0x100 #base add for array
5           li x6, 50
6           sw x6, 0(x10)
7           li x6, 40
8           sw x6, 4(x10)
9           li x6, 30
10          sw x6, 8(x10)
11          li x6, 20
12          sw x6, 12(x10)
13          li x6, 10
14          sw x6, 16(x10)
15          li x11, 5 #len
16          jal x1, bubble
17
18      bubble:
19          beq x11, x0, escape #if len is 0 return
20          beq x10, x0, escape #if base add is 0
21          li x5, 0 #i
22
23      outer_loop:
24          beq x5, x11, escape #if i ==len so end loop
25          addi x6, x5, 0 #j==i
```

(code continued on next page)

```
27      inner_loop:
28          beq x6, x11, end_outer #if j == len so end loop
29          slli x29, x5, 2 #i*4
30          add x29, x29, x10 #base add + i*4
31          slli x30, x6, 2 #j*4
32          add x30, x30, x10 #base add + j*4
33          lw x7, 0(x29) #load a[i] in the same
34          lw x28, 0(x30) #load a[j] in the same
35          bge x7, x28, skip #if a[j] < a[i]
36          sw x28, 0(x29) #swap a[i] and a[j]
37          sw x7, 0(x30)
38
39          j skip
40
41      end_outer:
42          addi x5, x5, 1 #increment i
43          j outer_loop
44
45      escape:
46          jalr x0, 0(x1)
47
48      skip:
49          addi x6, x6, 1 #increment i
50          j inner_loop
51
52      end:
53          j end
54
55      exit:
56          j end
```

Add screenshots of your results



Comments:

The code performs selection sort (sorting array in descending order) The ‘bubble’ part checks if values of x10, x11 are 0 or not (if 0 the sorting would simply be exit) The outerloop part runes for i iteration (in C++) (x5 = i) for every i iteration a j loop is ran over (j =x6)

The j loop, or inner loop checks element at i and j, (index at x5 is shifted 2 bits and is added to base address in x10, its result stored in x29). Same is followed with index in x6, and its address is fetched and stored in x30. The value from these addresses are stored in x7 and x28 and compared. if x7>= x28, the swap is skipped. Else if i is small, its swapped. Once the x6 (j loop) reaches its max length (x11) it exits and the value of i loop is incremented by 1 and it continues until i loop reaches its max length. (and then exits using escape label)

Task 4

Provide appropriately commented codes

Task4.s

Users > sheeeru > Downloads > Task4.s

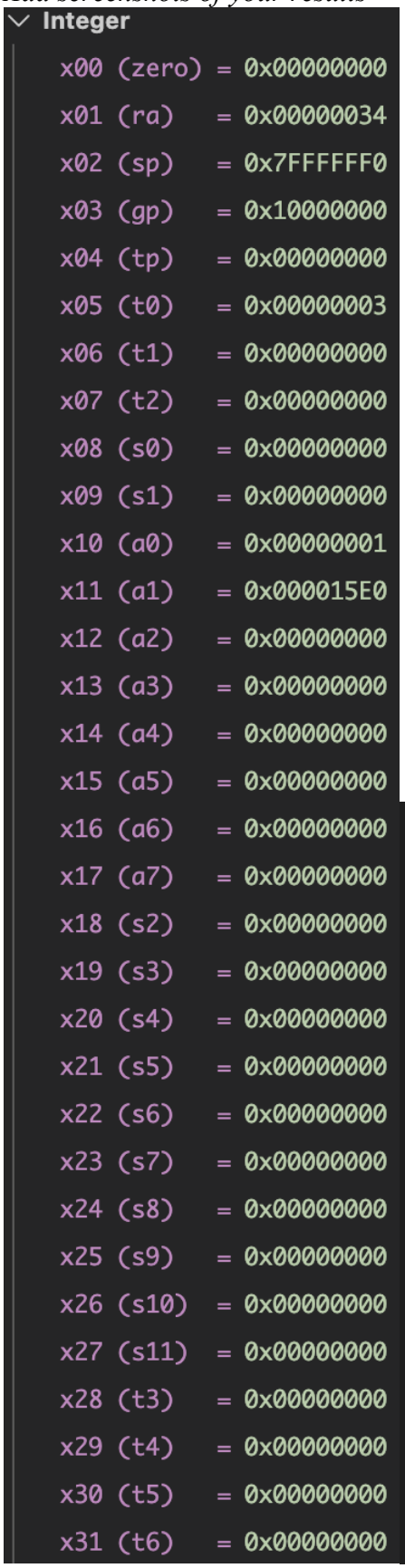
```
.text
.globl main
main:
    addi x10, x0, 0x100

    # ID 3 = Spectre, ID 2 = Sheriff, ID 1 = Classic
    # a[0] = 3 (Spectre)
    addi x5, x0, 3
    sw x5, 0(x10)
    # a[1] = 3 (Spectre)
    addi x5, x0, 3
    sw x5, 4(x10)
    # a[2] = 2 (Sheriff)
    addi x5, x0, 2
    sw x5, 8(x10)
    # a[3] = 1 (Classic)
    addi x5, x0, 1
    sw x5, 12(x10)
    # a[4] = 3 (Spectre)
    addi x5, x0, 3
    sw x5, 16(x10)

    addi x11, x0, 5 # Argument a1: Count = 5 players
    jal x1, calc_team_value

    addi x11, x10, 0
    li x10, 1
    ecall
    j exit

# "Non-Leaf Procedure" (The Manager)
# Loops through array. Calls 'get_price' for each item.
calc_team_value:
    addi sp, sp, -16
    sw x1, 8(sp) # Save Return Address
    sw s0, 4(sp) # Save s0 Array Pointer
    sw s1, 0(sp) # Save s1 Total Value
    sw s2, 12(sp) # Save s2 Counter

    # We move arguments to 's' registers. The Callee (get_price)
    addi s0, x10, 0 # s0 = Array Pointer (Copy from a0)
    addi s1, x0, 0 # s1 = Total Value (Initialize to 0)
    addi s2, x11, 0 # s2 = Counter (Copy from a1)
```

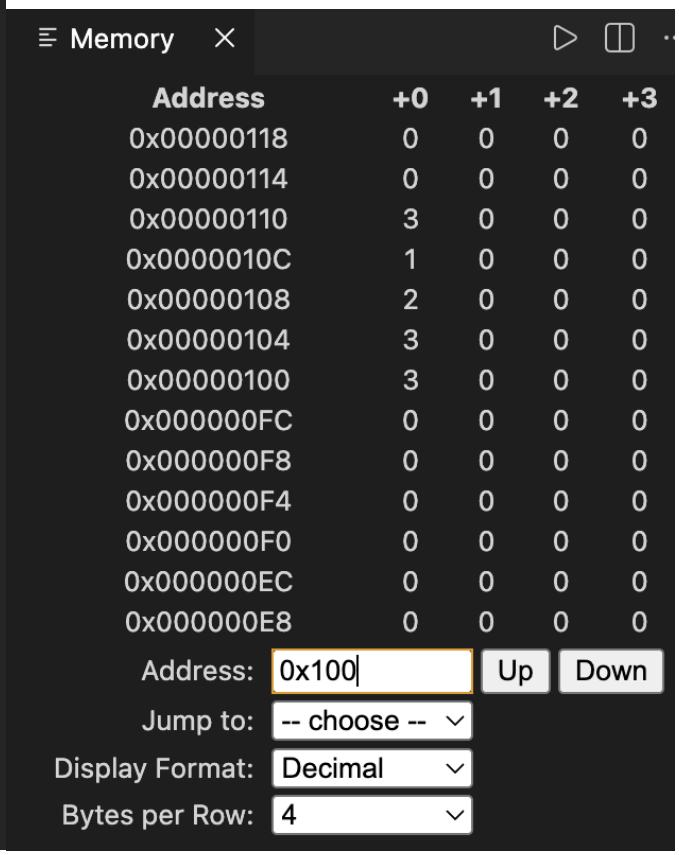
Task4.s

Users > sheeeru > Downloads > Task4.s

```
33  calc_team_value:
44
45  Loop:
46      beq s2, x0, EndLoop  # If Counter == 0, exit loop
47
48      lw x10, 0(s0) # Load Weapon ID from memory into Argument x10
49
50      jal x1, get_price # Call the child. x1 is overwritten here!
51      # (Result comes back in x10)
52
53      add s1, s1, x10 # Total += Price
54
55      addi s0, s0, 4 # Move Pointer to next word (4 bytes)
56      addi s2, s2, -1 # Decrement Counter
57      j Loop # Jump back to start
58
59  EndLoop:
60      # Return Value: Move Total (s1) to Result Register (x10/a0)
61      addi x10, s1, 0
62
63      lw s1, 0(sp) # Restore s1
64      lw s0, 4(sp) # Restore s0
65      lw x1, 8(sp) # Restore Return Address
66      lw s2, 12(sp) # Restore s2
67      addi sp, sp, 16
68
69      jalr x0, 0(x1)
70
71  # Takes ID, returns Price. Calls NO other functions.
72  # NO STACK NEEDED because it's a leaf.
73  get_price:
74      addi t0, x0, 3
75      beq x10, t0, IsSpectre   # If ID == 3, goto Spectre
76
77      addi t0, x0, 2
78      beq x10, t0, IsSheriff  # If ID == 2, goto Sheriff
79
80      # Default (Classic)
81      addi x10, x0, 0         # Price = 0
82      jalr x0, 0(x1)          # Return
83
84  IsSheriff:
85      addi x10, x0, 800 # Price = 800
86      jalr x0, 0(x1)
87
88  IsSpectre:
89      addi x10, x0, 1600 # Price = 1600
90      jalr x0, 0(x1)
91
92  end:
93      j end
94  exit:
95      j end
```

Add screenshots of your results





Comments:
Main function sets up 5 weapons. The loop runs 5 times.

Iteration	Weapon ID	Logic in get_price	Price Returned (x10)	Running Total (s1)
1	**3** (Spectre)	beq to IsSpectre	**1600**	0 + 1600 = **1600**
2	**3** (Spectre)	beq to IsSpectre	**1600**	1600 + 1600 = **3200**
3	**2** (Sheriff)	beq to IsSheriff	**800**	3200 + 800 = **4000**
4	**1** (Classic)	Default (No Match)	**0**	4000 + 0 = **4000**
5	**3** (Spectre)	beq to IsSpectre	**1600**	4000 + 1600 = **5600**

Final Result: The program prints **5600**.

The Stack

Inside calc_team_value, used the stack because we were about to call a "mini-game" (get_price).

- **addi sp, sp, -16**: You dug a hole 16 bytes deep to store your stuff.
- **sw x1, 8(sp)**: **S**aved the Return Address (ra).
- **sw s0, s1, s2**: Saved "Work Variables" (Pointer, Total, Counter).

Memory Offset	Register Saved	Why?
12(sp)	s2 (Counter)	Remembers we have "5" players left.
8(sp)	x1 (Return Addr)	Remembers how to get back to main.
4(sp)	s0 (Pointer)	Remembers which weapon we are looking at.
0(sp)	s1 (Total)	Remembers the current money total.

Jal jumps to the label (e.g., get_price) AND saves the current location into x1 so we can return later.

- main uses jal x1, calc_team_value (Call the manager).
- calc uses jal x1, get_price (Call the worker).
- get_price uses jalr x0, 0(x1) (Return to manager).
- calc uses jalr x0, 0(x1) (Return to main).
- **a Registers (x10-x17):**
  - **In Code:** Weapon ID in x10 to send it to get_price.
  - get_price put the cost in x10 to send it back.
- **s Registers (x8, x9, x18-x27):**
  - **In Code:** calc_team_value used s0, s1, s2.
- **t Registers (x5-x7, x28-x31):**
  - **In Code:** get_price used t0 for comparing numbers. It didn't need to save/restore it.

Lab 04 – Nest Procedures and Sorting
Assessment Rubrics

Points Distribution

Task No.	LR2 Code	LR 5 Results	AR 7 Report Submission & Github
Task 1	/10	/05	/10 & /10
Task 2	/10	/10	
Task 3	/10	/10	
Task 4	/15	/10	
Total Points	/100 Points		
CLO Mapped	CLO 2		

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

#	Assessment Elements	Level 1: Unsatisfactory Points 0-1	Level 2: Developing Points 2	Level 3:Good Points 3	Level 4:Exemplary Points 4
LR2	**Program/Code / Simulation Model/ Network Model**	Program/code/simulation model/network model does not implement the required functionality and has several errors. The student is not able to utilize even the basic tools of the software.	Program/code/simulation model/network model has some errors and does not produce completely accurate results. Student has limited command on the basic tools of the software.	Program/code/simulation model/network model gives correct output but not efficiently implemented or implemented by computationally complex routine.	Program/code/simulation /network model is efficiently implemented and gives correct output. Student has full command on the basic tools of the software.
LR5	**Results & Plots**	Figures/ graphs / tables are not developed or are poorly constructed with erroneous results. Titles, captions, units are not mentioned. Data is presented in an obscure manner.	Figures, graphs and tables are drawn but contain errors. Titles, captions, units are not accurate. Data presentation is not too clear.	All figures, graphs, tables are correctly drawn but contain minor errors or some of the details are missing.	Figures / graphs / tables are correctly drawn and appropriate titles/captions and proper units are mentioned. Data presentation is systematic.
AR9	**Report**	All the in-lab tasks are not included in report and / or the report is submitted too late.	Most of the tasks are included in report but are not well explained. All the necessary figures / plots are not included. Report is submitted after due date.	Good summary of most the in-lab tasks is included in report. The work is supported by figures and plots with explanations. The report is submitted timely.	Detailed summary of the in-lab tasks is provided. All tasks are included and explained well. Data is presented clearly including all the necessary figures, plots and tables.