



Department of Electrical Engineering

Faculty Member: Dr. Nasir Mahmood

Dated: 30/3/2018

Course/Section: BSCS-6-A

Semester: 4th Semester

**Computer Organization and
Architecture (EE321)**

Lab #
CPU Simulator

Name	Reg. no.	Report Marks / 10	Viva Marks / 5	Total/15
Abdul Razzaque Jawad	191408			
Awais Latif Khatti	199189			
Abdullah Rafaqat	146905			



Lab # CPU Simulator

Objectives: At the end of this lab you should be able to:

- Create instructions to move data to CPU registers
- Create instructions to compare values in CPU registers
- Push data to the stack, pop data from the stack
- Create instructions to jump to address locations
- Add values in CPU registers
- Demonstrate compare instruction's effect on the status register flags
- Use direct and indirect addressing modes
- Create iterative loop of instructions
- Display text on console using IO instruction
- Create a sub-routine
- Call the sub-routine and return from the sub-routine

Lab Instructions

- ✓ This lab activity comprises two parts, namely Lab tasks, and Post-Lab Viva session.
- ✓ Each group to upload completed lab report on LMS for grading.
- ✓ The students will start lab task and demonstrate each lab task separately for step-wise evaluation
- ✓ There are related questions in this activity give complete answers. Also provide complete code and results.



Introduction

The CPU simulator can run programs manually created. This requires four stages:

- 1) Create a CPU program
- 2) Enter CPU instructions in the program
- 3) Run the program
- 4) Observe and control simulations

Below is the description of these four main stages:

1. Creating a CPU program that will contain the instructions

The image shows two side-by-side dialog boxes. The 'New Program' dialog has fields for 'Program Name' (containing 'MyProgram'), 'Pages' (a dropdown menu showing '1'), and 'Base Address' (containing '0'). There is an 'ADD' button. The 'Program File' dialog has a 'Program List' dropdown menu (showing 'MyProgram'), a 'Base Address' field (containing '-1'), and a checked checkbox. There are 'SAVE...' and 'LOAD...' buttons.

Fig. 1 – New program frame

Enter the program's name in the **Program Name** text box, e.g. MyProgram. Enter a number in the **Base Address** text box (suggest you use 0 in this case). Next click the **ADD** button. The name of the program will appear in the PROGRAM LIST frame (see Fig. 2).

PROGRAM LIST			
Name	Base	Start	Type
MyProgram	0000	0000	R

Fig 2 – Program list frame

2. Adding CPU instructions in the program



Fig. 3 – Program instructions frame

In the **PROGRAM INSTRUCTIONS** frame the **ADD NEW...** button is enabled. Click this button to view the CPU instructions you can select for your program created above (see Fig. 4 below).

Selecting CPU instructions to manually enter in the program:

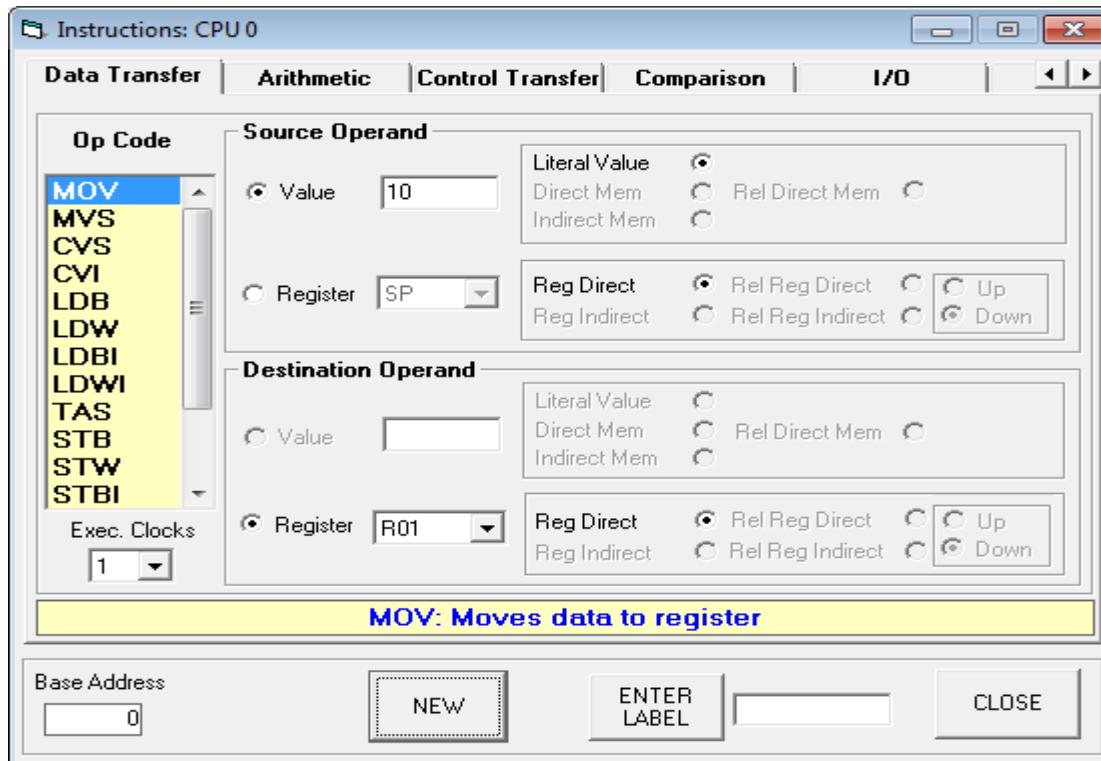


Fig. 4 – Selecting CPU instructions

In the **Instructions** window shown above, you can select any of the available CPU instructions for your program. Instructions that are related are categorized into different groups. You select a group by clicking on a tab. Select the desired instruction from the list under the **Op Code**. If the selected instruction requires operand(s) then the available types will be enabled under the **Source Operand** and the **Destination Operand** frames. Next, click the **NEW** button to add the instruction. You can add multiple instructions without closing this window.

The instruction will appear in CPU program memory area as shown in Fig. 5 below. The CPU simulator requires that the last instruction in the program should be the **HLT** instruction (this instruction stops the simulation from running). This is included in Fig. 5. The program is now ready to run.



CPU instructions in program:

INSTRUCTION MEMORY (RAM)				
PAdd	LAdd	Instruction	B	
<input type="checkbox"/> 0000	0000	MOV #10, R01	00	
<input type="checkbox"/> 0006	0006	HLT	00	

Fig. 5 – CPU program memory

Each entry in the **INSTRUCTION MEMORY** view includes the following information:

PAdd (Physical Address), **LAdd** (Logical Address) and the instruction. Other information is also available but this is not relevant at this stage.

Editing the program instructions:

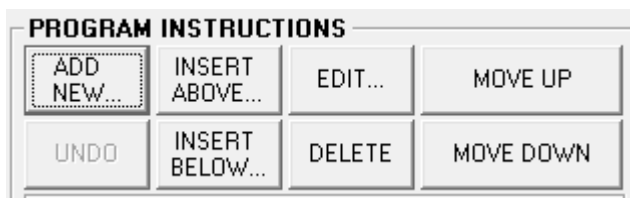


Fig. 6 – Program edit functions

Once the program instructions are entered they can be edited. To do this, select the desired instruction and use one of the editing functions (**EDIT**, **DELETE**, **MOVE UP**, **MOVE DOWN**) shown in Fig. 6 to edit the selected instruction. Use the **INSERT ABOVE...** and the **INSERT BELOW...** buttons to insert a new instruction above or below the selected instruction.

Removing the program:



Fig. 7 – Program removal

A CPU program can be removed by clicking the **REMOVE PROGRAM** button shown in Fig. 7. Once the program is removed it is no longer available and is lost. However you can save it so that you can re-load it at a later time (see Fig. 12).

3. Running the program

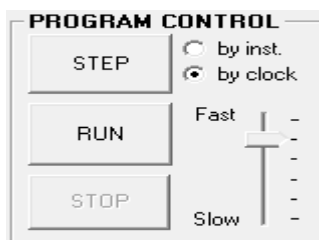


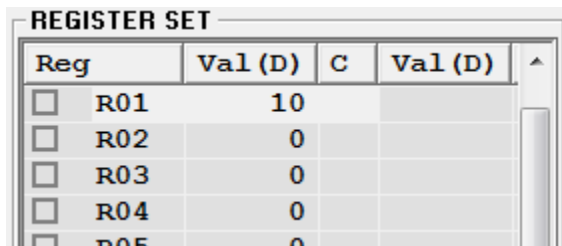
Fig. 8 – Program control

A CPU program can run in two different ways: 1) instruction by instruction, 2) automatically in one go. To run a desired instruction by itself, first select it as in Fig. 5 and double-click it. Alternatively, you can click the **STEP** button shown in Fig. 8 (make sure the **by inst.** option is selected). Use the **STOP** button to stop the running program. Use the slider control to speed up or slow down the running program.



4. Observing and controlling the simulations

Observing/altering the CPU registers:



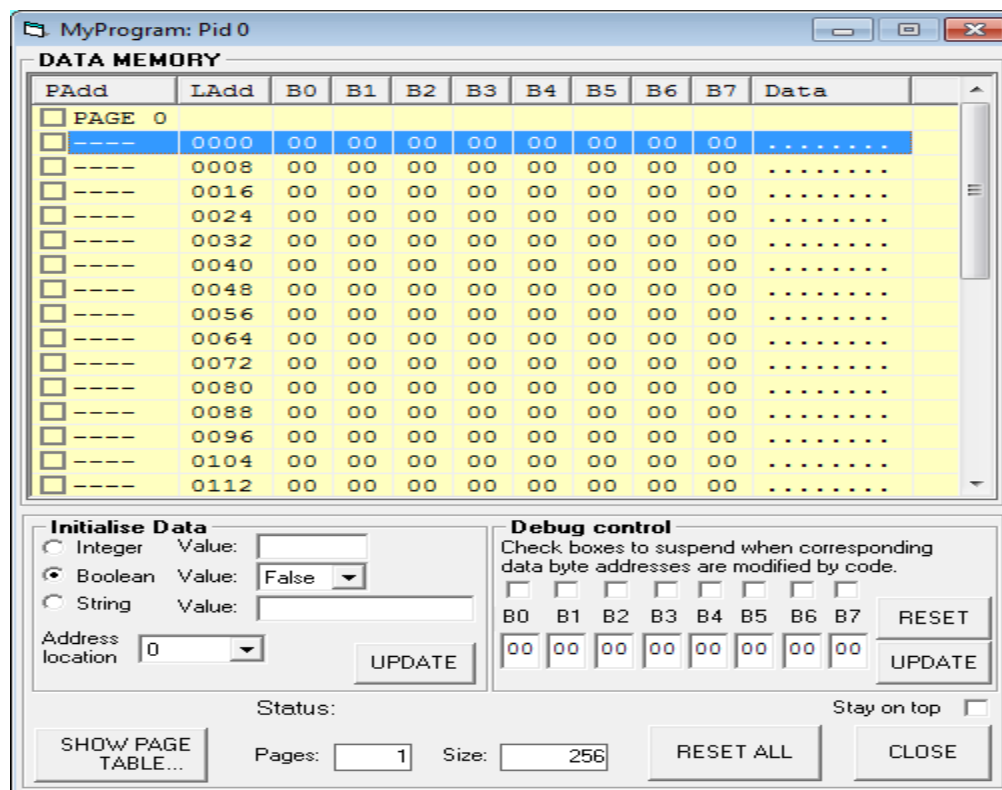
Reg	Val (D)	C	Val (D)
<input type="checkbox"/> R01	10		
<input type="checkbox"/> R02	0		
<input type="checkbox"/> R03	0		
<input type="checkbox"/> R04	0		
<input type="checkbox"/> R05	0		

Fig. 9 – Register set view

An instruction which writes or reads a register accesses the **REGISTER SET** frame and the accessed register is highlighted. This frame shows the registers and their values. Click on the **Val** (value) column to change the values from decimal (**D**) to hex (**H**) format and vice versa.

Note: To manually alter a register's value, first select it then enter the new value in the **Reg Value** field and click on the **CHANGE** button (not shown in Fig. 9).

Observing/altering the program data:



PAdd	LAdd	B0	B1	B2	B3	B4	B5	B6	B7	Data
<input type="checkbox"/> PAGE 0										
<input type="checkbox"/> ----	0000	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0008	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0016	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0024	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0032	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0040	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0048	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0056	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0064	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0072	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0080	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0088	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0096	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0104	00	00	00	00	00	00	00	00
<input type="checkbox"/> ----	0112	00	00	00	00	00	00	00	00

Initialise Data
☐ Integer Value:
☒ Boolean Value:
☐ String Value:
Address location:

Debug control
Check boxes to suspend when corresponding data byte addresses are modified by code.
☐ B0 ☐ B1 ☐ B2 ☐ B3 ☐ B4 ☐ B5 ☐ B6 ☐ B7

Status: Stay on top: ☐
 Pages: Size:

Fig. 10 – Data memory page

The CPU instructions that access that part of the memory containing data can write or read the data accessed. This information is available in the memory pages window shown in Fig. 10 above. You can display this window by clicking the **SHOW PROGRAM DATA MEMORY...** button shown in Fig. 7 above. In this window you can also edit the contents of the data.



The **LAdd** column shows the starting address of each line in the display. Each line of the display represents 8 bytes of information, so the **LAdd** numbers are incremented by 8 for each line down the display. Columns **B0** through to **B7** contain bytes 0 to 7. The **Data** column shows the displayable characters corresponding to the 8 bytes (hence 8 dots). Those bytes that correspond to non-displayable characters are shown as dots. The data bytes are displayed in hex format only.

To change the values of any bytes, first select the line(s) containing the bytes. Then use the information in the **Initialize Data** frame to modify the values of the bytes in the selected line(s) as **Integer**, **Boolean** or **String** formats. You need to click the **UPDATE** button to make the change.

Check the **Stay on top** check box to make sure the window always stays on top of other windows while still allowing access to the windows below it.

Observing/altering the program stack:

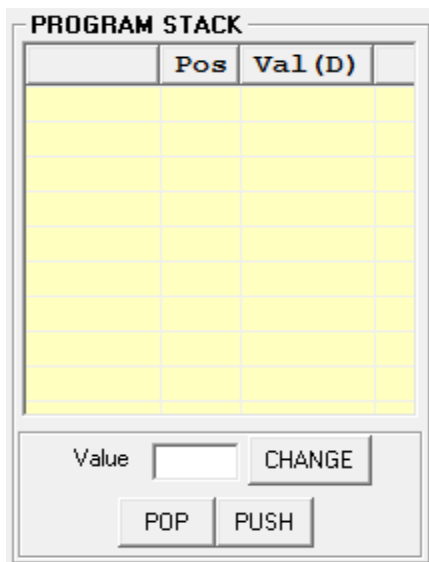


Fig. 11 – Program stack frame

Programs make use of the **PROGRAM STACK** for temporarily storing important information such as subroutine return addresses and subroutine parameters as well as other relevant information. There are instructions that can push (**PSH**) data on top of this stack and that can pop (**POP**) data from top of the stack to a register.

You can manually push and pop data by clicking the **PUSH** and **POP** buttons. You can also modify a stack entry by selecting it, entering the new value in the Value text box

Special CPU registers view

This view shows the set of CPU registers, which have pre-defined specialist functions:

PC: Program Counter contains the address of the next instruction to be executed.

IR: Instruction Register contains the instruction currently being executed.

SR: Status Register contains information pertaining to the result of the last executed instruction.

SP: Stack Pointer register points to the value maintained at the top of the program stack

BR: Base Register contains current base address.

MAR: Memory Address Register contains the memory address currently being accessed.

Status bits: **OV:** Overflow; **Z:** Zero; **N:** Negative



Saving and loading the CPU programs:

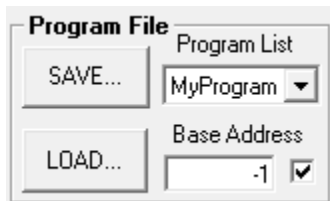


Fig. 12 – Saving and loading programs

To save a program select it from the pull down list and click the **SAVE...** button. To load a saved program click the **LOAD...** button.

Observing the displayed information:

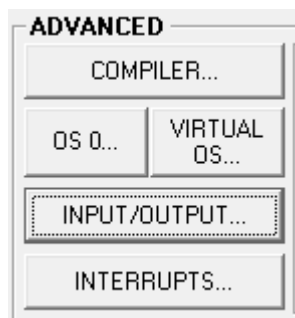


Fig. 13 – Console button

Programs are able to display information on and accept data from the simulated console. The **OUT** instruction is used to display information and the **IN** instruction is used to accept input from the console. To show the console click the **INPUT/OUTPUT...** button shown in Fig. 13. The console window is shown in Fig. 14 below.

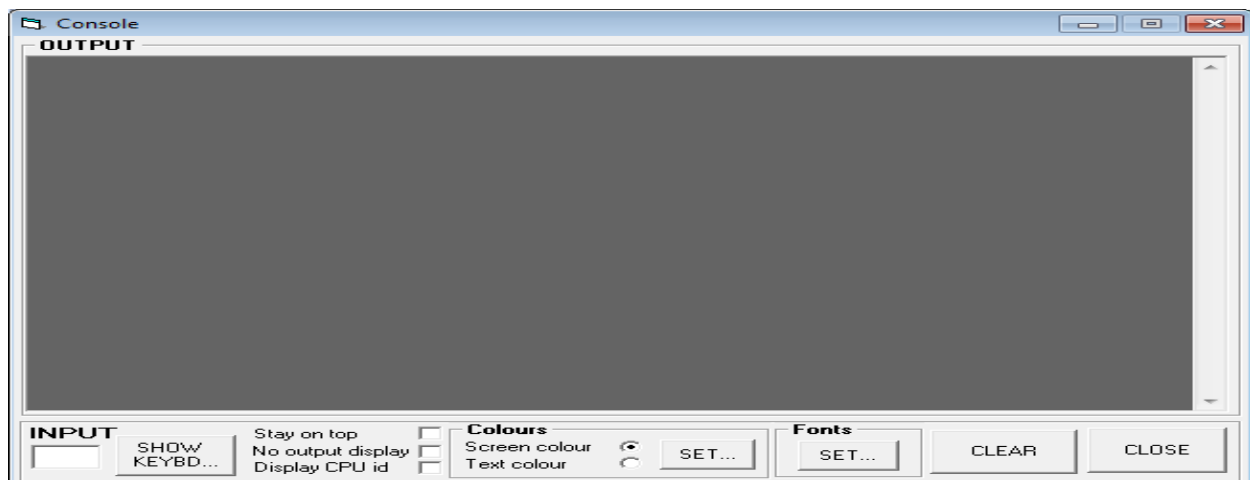


Fig. 14 – Console window

Check the **Stay on top** check box to make sure the window always stays on top of other windows while still allowing access to the windows below it. Click on the **SHOW KEYBD...** to show a small virtual keyboard used to input data (see Fig. 15 below).

Check the **Lower Case** check box to input lower case characters. This keyboard is a subset of a standard keyboard. You can also supply input to a program by typing in the **INPUT** text box shown in Fig. 14 above.



Fig. 15 – Virtual keyboard

Lab Exercise 1

You now need to enter instructions into the CPU simulator. You do this by first creating a new program and then clicking on the ADD NEW... button in the Edit Program tab. This will display the Instructions: CPU0 window. Use this window to enter the instructions. You'll find a list of useful CPU simulator instructions and examples at the end of the above document. You can also edit the instructions by using the edit commands available in the Edit Program tab.

Please record your work in the boxes as you go along.

1. Create an instruction that moves number 5 to register R00.

CPU INSTRUCTIONS IN MEMORY (RAM)			M)	
PAdd	LAdd	Instruction	Base	T
<input type="checkbox"/> 0000	0000	MOV #5, R00	0000	0
<input checked="" type="checkbox"/> 0006	0006	HLT	0000	2

2. Execute the above instruction by simply double clicking on it in the **CPU INSTRUCTIONS IN MEMORY (RAM)** view.

GENERAL PURPOSE CPU RE		
Reg	Val (D)	C
<input type="checkbox"/> R00	5	

3. Create an instruction that moves number 8 to register R01.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base		T
PAdd	LAdd	Instruction			
<input checked="" type="checkbox"/> 0000	0000	MOV #8, R01	0000	0	
<input type="checkbox"/> 0006	0006	HLT	0000	2	



4. Execute it.

GENERAL PURPOSE CPU RE		
Reg	Val (D)	C
<input type="checkbox"/> R00	5	
<input type="checkbox"/> R01	8	

5. Observe the contents of R00 and R01 in the **Register Set** view.

R00 = 5, R01 = 8.

6. Create an instruction that adds the contents of R00 and R01.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base	T
PAdd	LAdd	Instruction		
<input type="checkbox"/> 0000	0000	ADD R00, R01	0000	1
<input checked="" type="checkbox"/> 0005	0005	HLT	0000	2

7. Execute it.

GENERAL PURPOSE CPU RE		
Reg	Val (D)	C
<input type="checkbox"/> R00	5	
<input type="checkbox"/> R01	13	

8. Write down which register the result is stored in.

R01, because in the instruction we set R01 as its destination.



9. Create an instruction that pushes the above result to the top of the program stack, and then execute it. Observe the result in **PROGRAM STACK (RAM)** view.

CPU INSTRUCTIONS IN MEMORY (RAM)					PROGRAM STACK (RAM)			
PAdd	LAdd	Instruction	Base	T		Pos	Val (D)	Addr
<input type="checkbox"/> 0000	0000	PSH R01	0000	0	<input type="checkbox"/> TOS->	0	13	0000
<input checked="" type="checkbox"/> 0003	0003	HLT	0000	2				

10. Create an instruction to push number -2 on top of the stack and execute it.

CPU INSTRUCTIONS IN MEMORY (RAM)					PROGRAM STACK (RAM)			
PAdd	LAdd	Instruction	Base	T		Pos	Val (D)	Addr
<input type="checkbox"/> 0000	0000	PSH #-2	0000	0	<input type="checkbox"/> TOS->	1	-2	0000
<input checked="" type="checkbox"/> 0004	0004	HLT	0000	2	<input type="checkbox"/> BOS->	0	13	0000

11. Create an instruction to compare the values in registers R00 and R01.

CPU INSTRUCTIONS IN MEMORY (RAM)					Base	T
PAdd	LAdd	Instruction				
<input type="checkbox"/> 0000	0000	CMP R01, R00	0000	3		
<input checked="" type="checkbox"/> 0005	0005	HLT	0000	2		

12. Execute it.

SPECIAL CPU REGISTERS

PC	<input type="text" value="5"/>	SR	<input type="text" value="2"/>
SP	<input type="text" value="8102"/>	BR	<input type="text" value="0"/>

SR Status Flag

OV ☐ Z ☐ N ☒

CPU Mode

User ☒ Kernel ☐



13. Record the status (i.e. set or reset) of the **Z/N** flags of the status register. If the box is checked then means set otherwise means not set.

As shown in above figure, 'N' flag is checked (set). Which means $R0 < R1$ and thus, on subtraction results in a negative number. **Note:** CMP instruction is executed in reverse order of source and destination in this simulator.

14. Create an instruction to unconditionally jump to the first instruction.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base	T
PAdd	LAdd	Instruction		
<input checked="" type="checkbox"/> 0000	0000	CMP R01, R00	0000	3
<input type="checkbox"/> 0005	0005	JMP 0	0000	2
<input type="checkbox"/> 0009	0009	HLT	0000	2

15. Execute it.

SPECIAL CPU REGISTERS

PC SR
SP BR

SR Status Flag
OV ☐ Z ☐ N ☒

CPU Mode
User ☒ Kernel ☐

IR
MAR
MDR

16. Observe the value in the PC register. What address is it pointing to? Explain.

As shown in above figure, On executing instruction of JMP in IR, PC contains the 0 address. Next instruction which is the first one in the program having 0 address.

17. Create an instruction to pop the value on top of the program stack into register R02.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base	T
PAdd	LAdd	Instruction		
<input checked="" type="checkbox"/> 0000	0000	POP R02	0000	0
<input type="checkbox"/> 0003	0003	HLT	0000	2

18. Execute it.





19. Create an instruction to pop the value on top of the program stack into register R03.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base	T
PAdd	LAdd	Instruction		
<input type="checkbox"/> 0000	0000	POP R03	0000	0
<input type="checkbox"/> 0003	0003	HLT	0000	2

20. Execute it.

GENERAL PURPOSE CPU REG		
Reg	Val (D)	C
<input type="checkbox"/> R00	5	
<input type="checkbox"/> R01	13	
<input type="checkbox"/> R02	-2	
<input type="checkbox"/> R03	13	

21. Execute the last instruction again. What happened? Briefly explain.

On executing the instruction, it gives underflow exception as stack was empty so no pop instruction is valid to execute.

22. Create a compare instruction that compares values in registers R04 and R05.

CPU INSTRUCTIONS IN MEMORY (RAM)			Base	T
PAdd	LAdd	Instruction		
<input type="checkbox"/> 0000	0000	CMP R05, R04	0000	3
<input type="checkbox"/> 0005	0005	HLT	0000	2

SR Status Flag

OV ☐ Z ☒ N ☐

23. Manually insert two equal values in registers R04 and R05.

<input type="checkbox"/> R04	7
<input type="checkbox"/> R05	7

24. Execute the compare instruction in step 22 above.

SR Status Flag		
OV	<input type="checkbox"/>	Z <input checked="" type="checkbox"/> N <input type="checkbox"/>

25. Which of the status flags **Z/N** is set? Why?



Zero flag is set(as checked), because both values are equal and hence on subtraction of both values it results 0 value and thus, zero flag is set.

26. Manually insert a value in register R05 greater than that in register R04.

<input type="checkbox"/>	R04	7
<input type="checkbox"/>	R05	14

27. Execute the compare instruction in step 22 above.

SR Status Flag

OV ☐ Z ☐ N ☒

28. Which of the status flags **Z/N** is set? Why?

Negative sign flag is set, because R4 value is smaller than the R5 value and on subtracting R5 from R4 it gives negative answer and thus, N flag is set.

29. Manually insert a value in register R04 greater than that in register R05.

<input type="checkbox"/>	R04	15
<input type="checkbox"/>	R05	14

30. Execute the compare instruction in step 22 above.

SR Status Flag

OV ☐ Z ☐ N ☐



31. Which of the status flags **Z/N** is set? Why?

No flag is set, because R4 is greater than R5 and on subtracting R5 from R4 it produce positive answer and non zero. So no zero, overflow and negative flag is set in this case.

32. Create an instruction that will conditionally jump to the first instruction if the values in registers R04 and R05 are equal (**Note:** You will need to execute the compare instruction first before you execute the jump instruction if you change values in R04 and R05)

PAdd	LAdd	Instruction	Base	T
<input type="checkbox"/> 0000	0000	CMP R05, R04	0000	3
<input type="checkbox"/> 0005	0005	JEQ 0	0000	2
<input type="checkbox"/> 0009	0009	HLT	0000	2

33. Test this instruction by manually putting values in registers R04 and R05 and then first executing the compare instruction and then executing the jump instruction (**i.e.** You will need to execute the compare instruction first before you execute the jump instruction every time you change values in R04 and R05)

R4 > R5:

SR Status Flag		<input type="checkbox"/> R04	15
OV <input type="checkbox"/>	Z <input type="checkbox"/>	<input type="checkbox"/> R05	14

R4 < R5:

<input type="checkbox"/> R04	12	SR Status Flag	
<input type="checkbox"/> R05	14	OV <input type="checkbox"/>	Z <input type="checkbox"/>
		N <input checked="" type="checkbox"/>	

R4 = R5:

SR Status Flag		<input type="checkbox"/> R04	14
OV <input type="checkbox"/>	Z <input checked="" type="checkbox"/>	<input type="checkbox"/> R05	14



Appendix - Simulator Instruction Sub-set

Inst.	Description
Data transfer instructions	
MOV	Move data to register; move register to register e.g. MOV #2, R01 moves number 2 into register R01 MOV R01, R03 moves contents of register R01 into register R03
LDB	Load a byte from memory to register e.g. LDB 1022, R03 loads a byte from memory address 1022 into R03 LDB @R02, R05 loads a byte from memory the address of which is in R02
LDW	Load a word (2 bytes) from memory to register Same as in LDB but a word (i.e. 2 bytes) is loaded into a register
STB	Store a byte from register to memory STB R07, 2146 stores a byte from R07 into memory address 2146 STB R04, @R08 stores a byte from R04 into memory address of which is in R08
STW	Store a word (2 bytes) from register to memory Same as in STB but a word (i.e. 2 bytes) is loaded stored in memory
PSH	Push data to top of hardware stack (TOS); push register to TOS e.g. PSH #6 pushes number 6 on top of the stack PSH R03 pushes the contents of register R03 on top of the stack
POP	Pop data from top of hardware stack to register e.g. POP R05 pops contents of top of stack into register R05 Note: If you try to POP from an empty stack you will get the error message "Stack underflow".
Arithmetic instructions	
ADD	Add number to register; add register to register e.g. ADD #3, R02 adds number 3 to contents of register R02 and stores the result in register R02. ADD R00, R01 adds contents of register R00 to contents of register R01 and stores the result in register R01.
SUB	Subtract number from register; subtract register from register
MUL	Multiply number with register; multiply register with register
DIV	Divide number with register; divide register with register
Control transfer instructions	
JMP	Jump to instruction address <u>unconditionally</u> e.g. JMP 100 unconditionally jumps to address location 100 where there is another instruction



JLT	Jump to instruction address if less than (after last comparison)
JGT	Jump to instruction address if greater than (after last comparison)
JEQ	Jump to instruction address if equal (after last comparison instruction) e.g. JEQ 200 jumps to address location 200 if the previous comparison instruction result indicates that the two numbers are equal, i.e. the Z status flag is set (the Z box will be checked in this case).
JNE	Jump to instruction address if not equal (after last comparison)
MSF	Mark Stack Frame instruction is used in conjunction with the CAL instruction. e.g. MSF reserve a space for the return address on program stack CAL 1456 save the return address in the reserved space and jump to subroutine in address location 1456
CAL	Jump to subroutine address (saves the return address on program stack) This instruction is used in conjunction with the MSF instruction. You'll need an MSF instruction before the CAL instruction. See the example above
RET	Return from subroutine (uses the return address on stack)
SWI	Software interrupt (used to request OS help)
HLT	Halt simulation
Comparison instruction	
CMP	Compare number with register; compare register with register e.g. CMP #5, R02 compare number 5 with the contents of register R02 CMP R01, R03 compare the contents of registers R01 and R03 Note: If $R01 = R03$ then the status flag Z will be set, i.e. the Z box is checked. If $R01 < R03$ then none of the status flags will be set, i.e. none of the status flag boxes are checked. If $R01 > R03$ then the status flag N will be set, i.e. the N status box is checked.
Input, output instructions	
IN	Get input data (if available) from an external IO device
OUT	Output data to an external IO device e.g. OUT 16, 0 outputs contents of data in location 16 to the console (the second parameter must always be a 0)