Date: 09-01-2023 **Name:** Ashvath S.P **Reg No**: 2162014

COA LAB

Experiment – 1

Problem statement: Write an assembly language program to perform arithmetic operations.

Algorithm:

Step 1: Define the Base Register Address value during the program creation.
Step 2: Move the first operand in the General-Purpose Register R1.
Step 3: Move the second operand in the General-Purpose Register R2.
Step 4: Perform the arithmetic operation with the values in the registers.
Step 5: Result will be stored in the destination register.
Step 6: Store the resultant value in a data memory location.
Step 7: Terminate the program.

Assembly Language Code:

- **Addition**

  MOV #10, R01 *;Store value of 10 in register R01*

  MOV #5, R02 *;Store value of 05 in register R02*

  ADD R02, R01 *;Add the register R01 and R02 values and store the resultant value in register R01*

  STB R01, 00 *;Store the resultant value of R01 in memory location 00*

  HLT *;Stop the simulator*

- **Subtraction**

  MOV #20, R03 *;Store value of 20 in register R03*

  MOV #15, R04 *;Store value of 15 in register R04*

  SUB R04, R03 *;Subtract the register R03 and R04 values and store the resultant value in register R03*

  STB R03, 08 *;Store the resultant value of R03 in memory location 08*

  HLT *;Stop the simulator*

- **Multiplication**

  MOV #6, R05 *;Store value of 06 in register R05*

MOV #3, R06 *;Store value of 03 in register R06*

MUL R06, R05 *;Multiplicate the register R05 and R06 values and store the resultant value in register R05*

STB R05, 16 *;Store the resultant value of R05 in memory location 16*

HLT *;Stop the simulator*

- **Division**

MOV #8, R07 *;Store value of 08 in register R07*

MOV #2, R08 *;Store value of 02 in register R08*

DIV R08, R07 *;Divide the register R07 and R08 values and store the resultant value in register R07*

STB R07, 24 *;Store the resultant value of R07 in memory location 24*

HLT *;Stop the simulator*

Result:

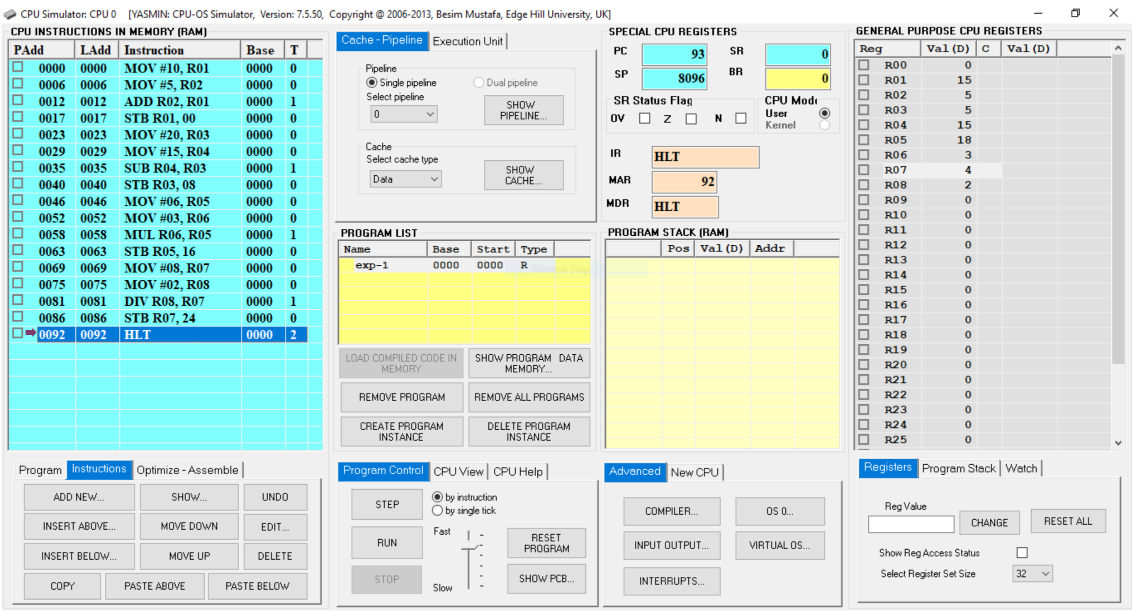


Fig. 1: CPU Simulator Window

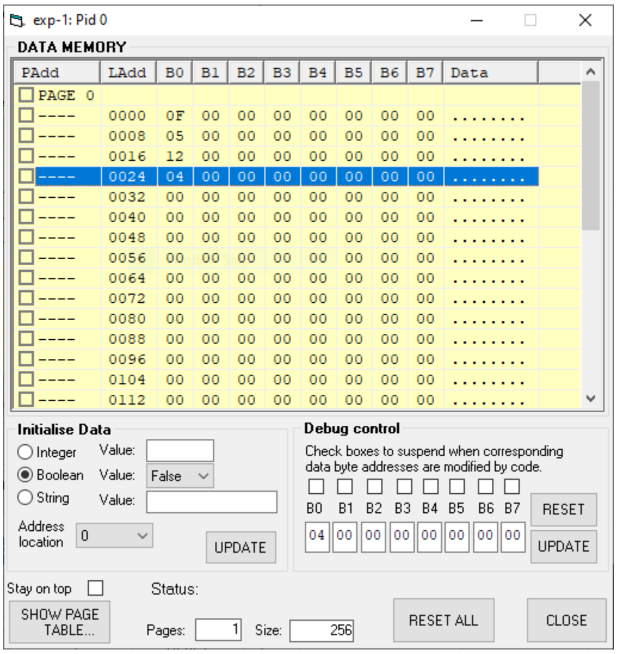


Fig. 2: Data Memory Window

Step 01 - Addition Program Starts	
PC	6
IR	MOV #10, R01
MAR	0
MDR	MOV #10, R01
R01	10
Step 02	
PC	12
IR	MOV #5, R02
MAR	6
MDR	MOV #5, R02
R01	10
R02	5
Step 03	
PC	17
IR	ADD R02, R01
MAR	12
MDR	ADD R02, R01
R01	15
R02	5
Step 04 - Addition Program Ends	
PC	23
IR	STB R01,00
MAR	0
MDR	15

R01	15
R02	5
00	0F
Step 05 - Subtraction Program Starts	
PC	29
IR	MOV #20, R03
MAR	23
MDR	MOV #20, R03
R01	15
R02	5
R03	20
00	0F
Step 06	
PC	35
IR	MOV #15, R04
MAR	29
MDR	MOV #15, R04
R01	15
R02	5
R03	20
R04	15
00	0F
Step 07	
PC	40
IR	SUB R04, R03
MAR	29
MDR	SUB R04, R03
R01	15
R02	5
R03	5
R04	15
00	0F
Step 08 - Subtraction Program Ends	
PC	46
IR	STB R03, 08
MAR	8
MDR	5
R01	15
R02	5
R03	5
R04	15
00	0F

08	05
Step 09 - Multiplication Program Starts	
PC	52
IR	MOV #06, R05
MAR	46
MDR	MOV #06, R05
R01	15
R02	5
R03	5
R04	15
R05	6
00	0F
08	05
Step 10	
PC	58
IR	MOV #03, R06
MAR	52
MDR	MOV #03, R06
R01	15
R02	5
R03	5
R04	15
R05	6
R06	3
00	0F
08	05
Step 11	
PC	63
IR	MUL R06, R05
MAR	58
MDR	MUL R06, R05
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
00	0F
08	05
Step 12 - Multiplication Program Ends	
PC	69
IR	STB R05, 16

MAR	16
MDR	18
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
00	0F
08	05
16	12
Step 13 - Division Program Starts	
PC	75
IR	MOV #08, R07
MAR	69
MDR	MOV #08, R07
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
R07	8
00	0F
08	05
16	12
Step 14	
PC	81
IR	MOV #02, R08
MAR	75
MDR	MOV #02, R08
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
R07	8
R08	2
00	0F
08	05
16	12

Step 15	
PC	86
IR	DIV R08, R07
MAR	81
MDR	DIV R08, R07
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
R07	4
R08	2
00	0F
08	05
16	12
Step 16 - Division Program Ends	
PC	92
IR	STB R07, 24
MAR	24
MDR	4
R01	15
R02	5
R03	5
R04	15
R05	18
R06	3
R07	4
R08	2
00	0F
08	05
16	12
24	04
Step 17 - Simulation is Terminated	
PC	93
IR	HLT
MAR	92
MDR	HLT
R01	15
R02	5
R03	5
R04	15

R05	18
R06	3
R07	4
R08	2
00	0F
08	05
16	12
24	04

COA LAB

Experiment – 2

Problem Statement: Write an assembly language program to compute the average of two numbers.

Algorithm:

Step 1: Define the Base Register Address value during the program creation.
Step 2: Move the first operand in the General-Purpose Register R1.
Step 3: Move the second operand in the General-Purpose Register R2.
Step 4: Perform the addition operation with the values in the registers.
Step 5: Result will be stored in the destination register.
Step 6: Divide the destination register value by 2, and the result will be stored in the destination register.
Step 7: Store the resultant value in a data memory location.
Step 8: Terminate the program.

Assembly Language code:

MOV #6, R01 *;Store value of 6 in register R01*

MOV #4, R02 *;Store value of 4 in register R02*

ADD R01, R02 *;Add the register R01 and R02 values and store the resultant value in register R02*

DIV #2, RO2 *;Divide register RO2 by value 2 and store the resultant value in register R02*

STB R01, 00 *;Store the resultant value of R01 in memory location 00*

HLT *;Stop the simulator*

Result:

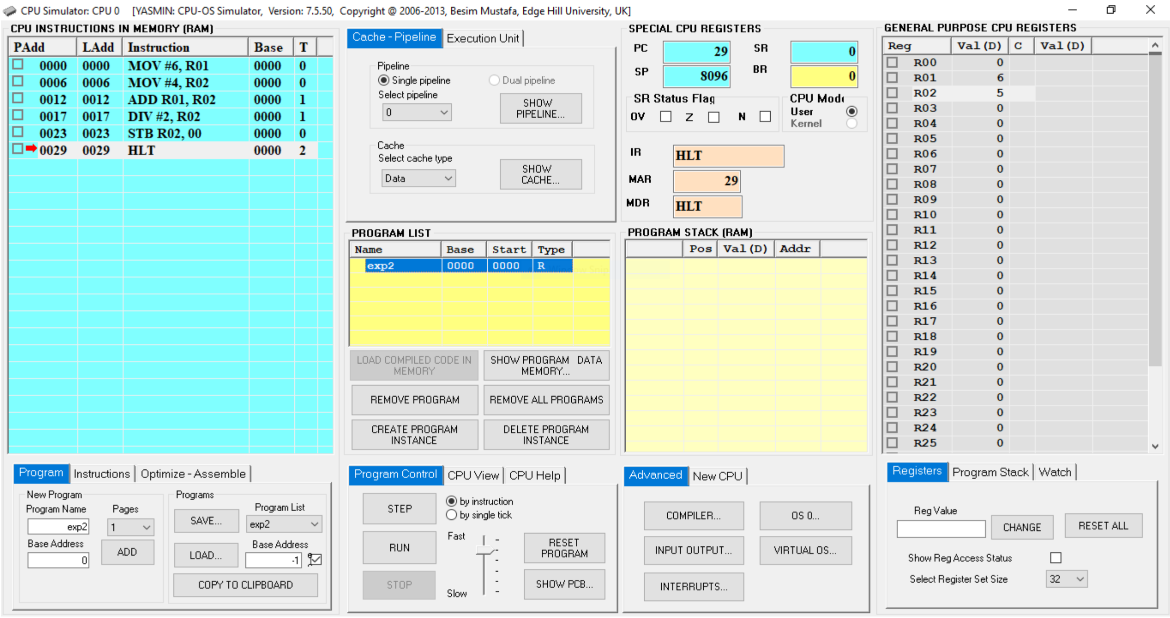


Fig. 1: CPU Simulator Window

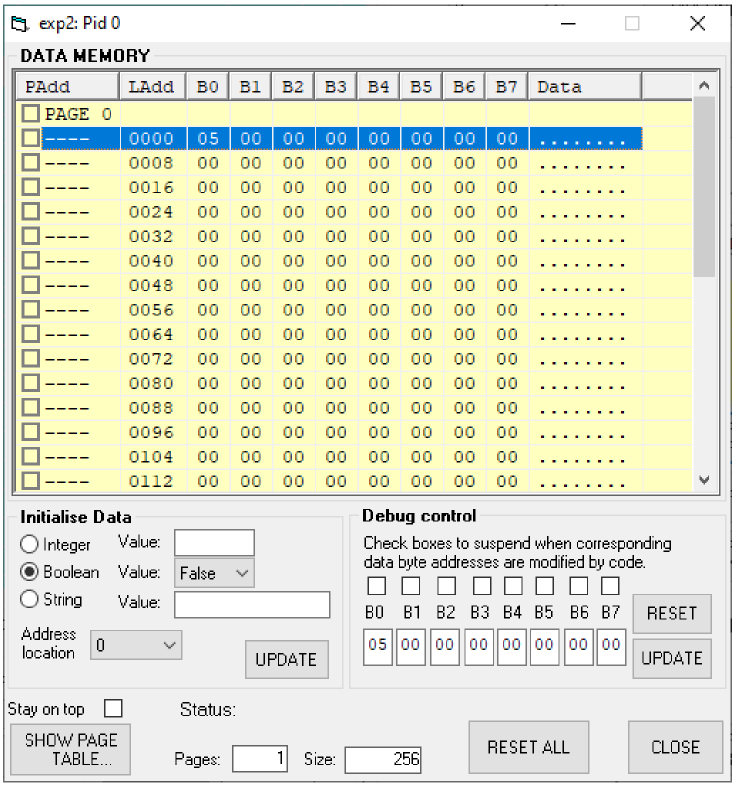


Fig. 2: Data Memory Window

Step 01	
PC	6
IR	MOV #6, R01
MAR	0
MDR	MOV #6, R01
R01	6
Step 02	

PC	12
IR	MOV #4, R02
MAR	6
MDR	MOV #4, R02
R01	6
R02	4
Step 03	
PC	17
IR	ADD R01, R02
MAR	12
MDR	ADD R01, R02
R01	6
R02	10
Step 04	
PC	23
IR	DIV #2, R02
MAR	17
MDR	DIV #2, R02
R01	6
R02	5
00	05
Step 05	
PC	29
IR	STB R02, 00
MAR	0
MDR	5
R01	6
R02	5
00	05
Step 06	
PC	30
IR	HLT
MAR	29
MDR	HLT
R01	6
R02	5
00	05

COA LAB

Experiment – 3

Problem Statement: Write an assembly language program to compute factorial for a given number.

Algorithm:

Step 1: Define the Base Register Address value during the creation of the program
Step 2: Move the operand to Register R1 for which you need to find out the factorial
Step 3: Move the Register R1 value to the R0
Step 4: Move the value 1 to Register R2
Step 5: Create a label named 'factorial'
Step 6: Multiply Register R1 with register R2 and store result in R2 register
Step 7: Decrement Register R1 value
Step 8: Compare Register R1 with value 1
Step 9: If the Register R1 is greater than 1, jump to the 'factorial' label
Step 10: If the Register R1 is lower than or equal to 1, store the resultant factorial value in the memory location
Step 11: Halt the simulator

Assembly Language code:

MOV #5, R01 *;Store value of 5 in register R01*

MOV R01, R00 *;Move register R01 value to R00.*

MOV #1, R02 *;Store value of 1 in register R02*

factorial: *;Label for factorial*

MUL R01, RO2 *;Multiply registers RO1 to R02 and store the resultant value in register R02*

DEC R01 *;Decrement register R01 value by 1*

CMP #1, R01 *;Compare register R01 value by 1*

JGT $factorial *;If register R01 value is greater than 1, jump to the 'factorial' label*

STB R02, 0 ;*Store register R02 value in memory location 0*

JLT 35 *;If register R01 value is less than or equal to 1, jump to the statement of Padd 35*

HLT *;Stop the simulator*

Result:

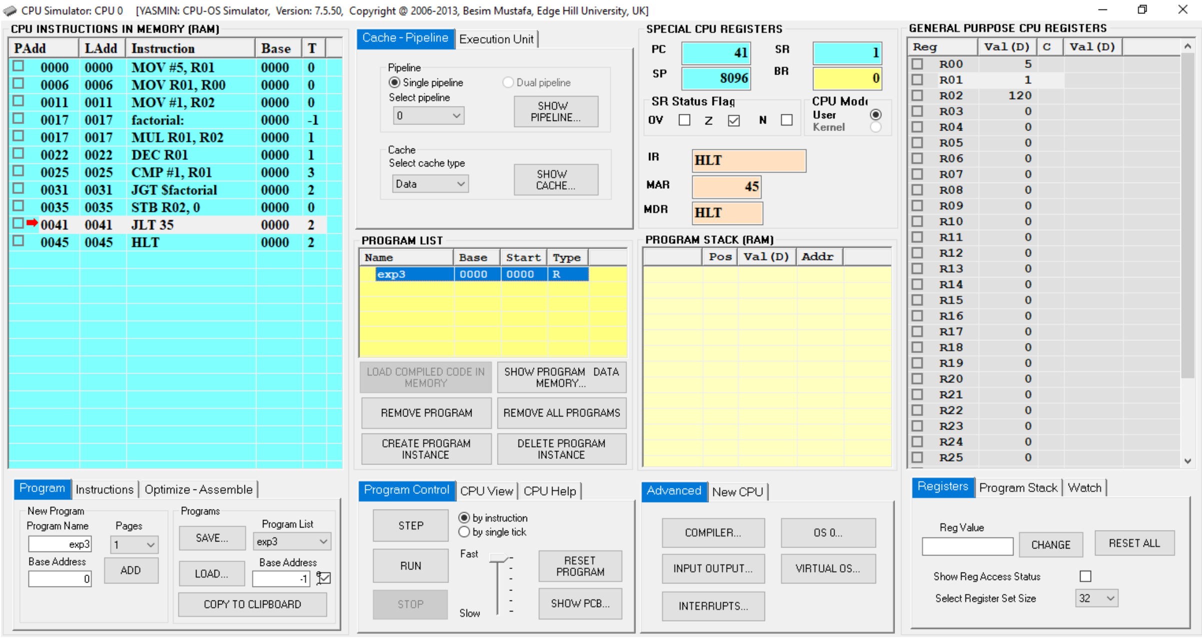


Fig. 1: CPU Simulator Window

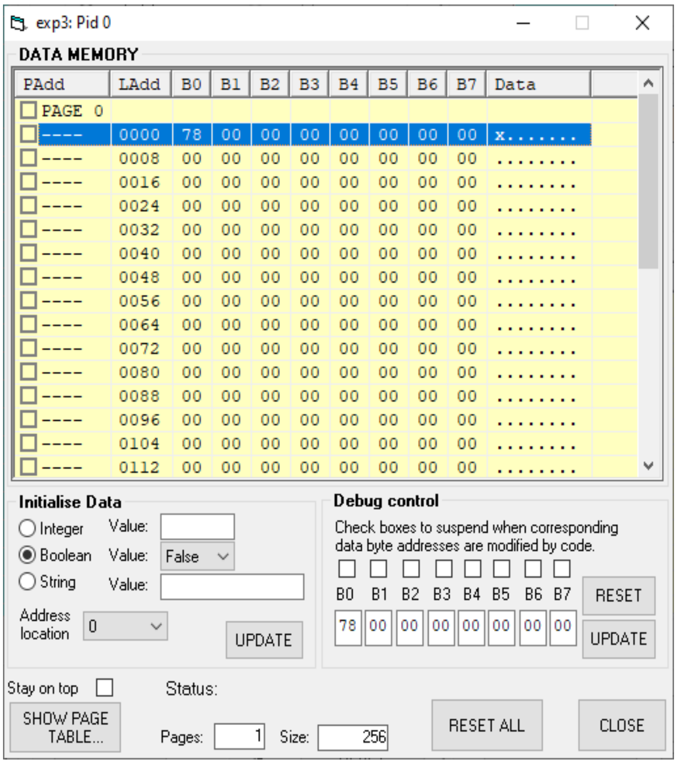


Fig. 2: Data Memory Window

Date: 06-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

COA LAB
Experiment – 4

Problem Statement: Write an assembly language program to check whether a given number is odd or even.

Algorithm:

1. Define the Base Register Address value during the creation of the program
2. Move the operand to the Register R1
3. Move the Register R1 value to the R0
4. Perform bitwise and operation on R1value and decimal number 1
5. Compare whether the resulting value is zero
6. If the resulting value is zero, jump to the Even label, set the Register R4 to 1 and exit
7. If the resulting value is not zero, jump to the Odd label, set the Register R5 to 1 and exit

Assembly Language code:

MOV #37, R01 *;Store value of 37 in register R01*

MOV R01, R00 *;Move register R01 value to R00.*

AND #1, R01 *;AND Operation on R1 value and decimal number 1*

CMP #0, R01 *;Compare register R01 value by 0*

JEQ $EVEN *;If register R01 value is equal to 0, jump to the 'EVEN' label*

JNE $ODD *;If the register R01 value is not equal to 0, jump to the 'ODD' label*

EVEN: *;Label for identifying even numbers*

MOV #1, R04 *;Store value of 1 in register R04*

HLT *;Stop the simulator*

EVEN: *;Label for identifying odd numbers*

MOV #1, R05 *;Store value of 1 in register R05*

HLT *;Stop the simulator*

Date: 06-02-2023 Name: Ashvath S.P Reg No: 2162014

Result:

Case 1: Odd

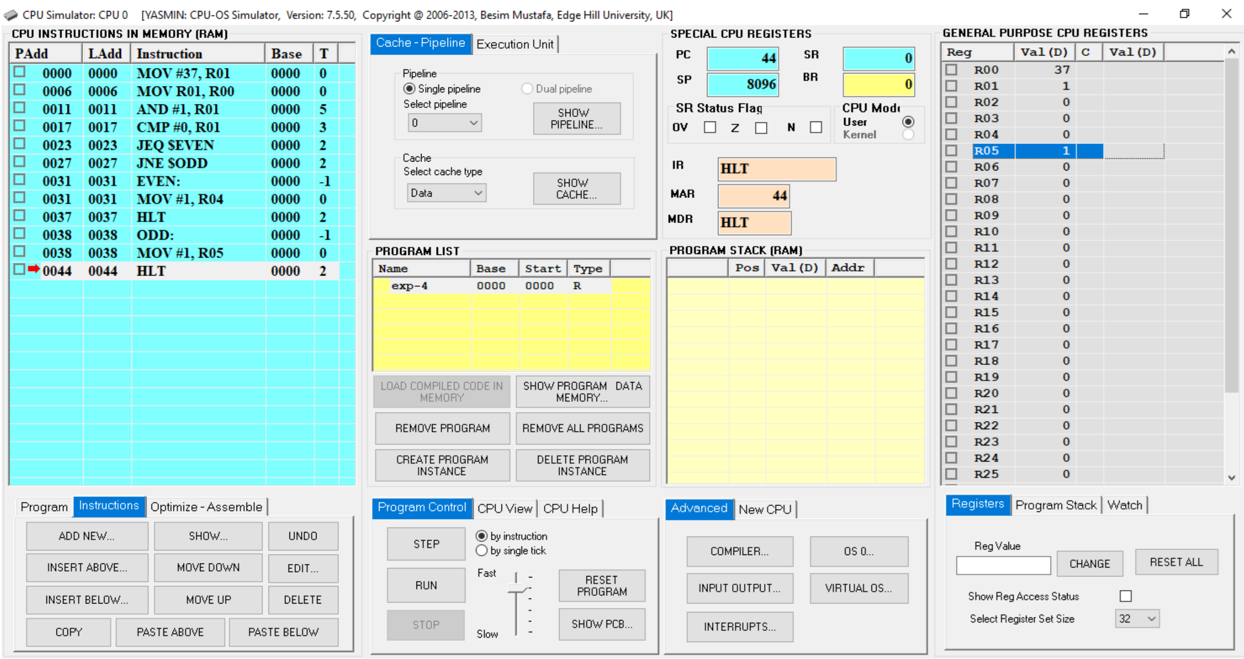


Fig.1: CPU Simulator Window

Case 2: Even

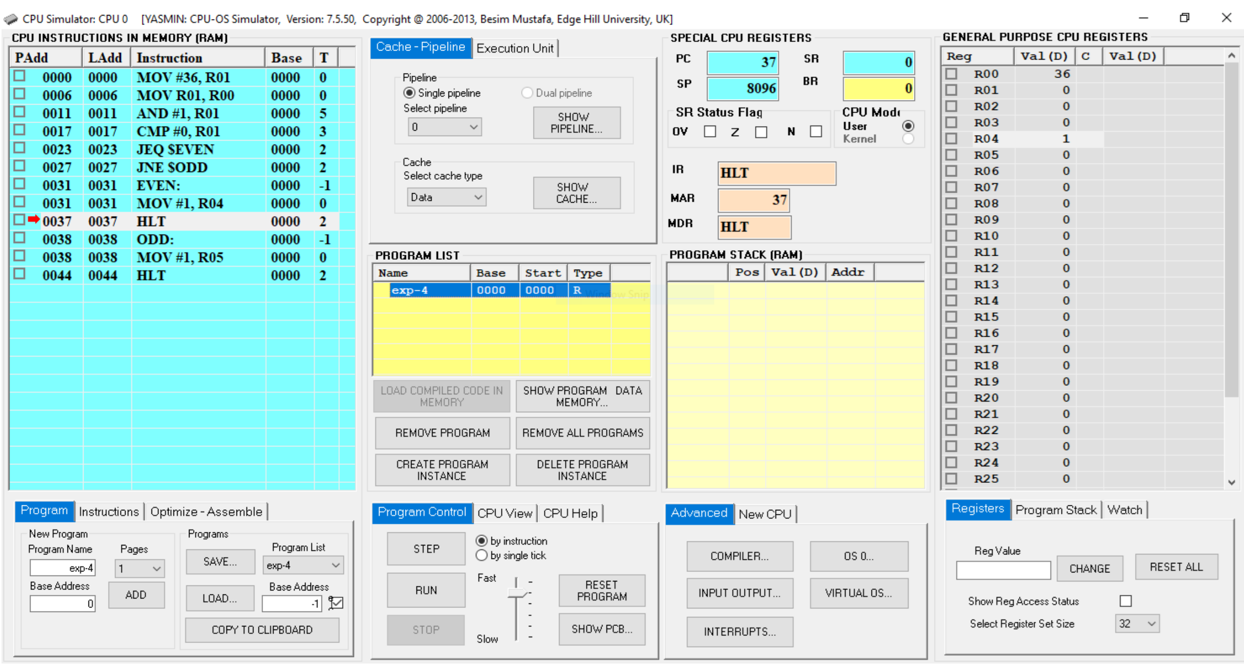


Fig.2: CPU Simulator Window

Date: 13-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

COA LAB
Experiment - 5

Write an assembly language for the following conditional statements:

1. If R02 is greater than R01, R03 is set to 8. (Use R01 as the first operand and R02 as the second operand).
2. If R01 = 0, R03 is set to 5, else R03 is set to R01 plus 1.
3. A loop that repeats 5 times where R02 is incremented by 2 every time the loop repeats.
4. A loop that repeats while R04 is > 0. Set the initial value of R04 to 8.
5. A loop that repeats until R05 is > R09. Set the initial values of R05 to 0 and R09 to 12.

Program no. 1

Algorithm:

1. Load the value 15 into R01
2. Load the value 5 into R02
3. Compare R01 and R02 with the CMP instruction
4. If R02 is greater than R01, jump to the ADD8 label
5. If R02 is not greater than R01, halt the program with the HLT instruction
6. At the ADD8 label, load the value 8 into R03
7. Halt the program with the HLT instruction

Assembly Language code:

```
MOV #15, R01 ;Load the value 15 into R01
MOV #5, R02 ;Load the value 5 into R02
CMP R01, R02 ;Compare R01 and R02
JGT ADD8 ;If R02 is greater than R01, jump to the ADD8 label
HLT ;If R02 is not greater than R01, halt the program
ADD8: ;Label for the ADD8 instruction
MOV #8, R03 ;Load the value 8 into R03
HLT ;Halt the program
```

Result:

Case 1: R02 > R01

Date: 13-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

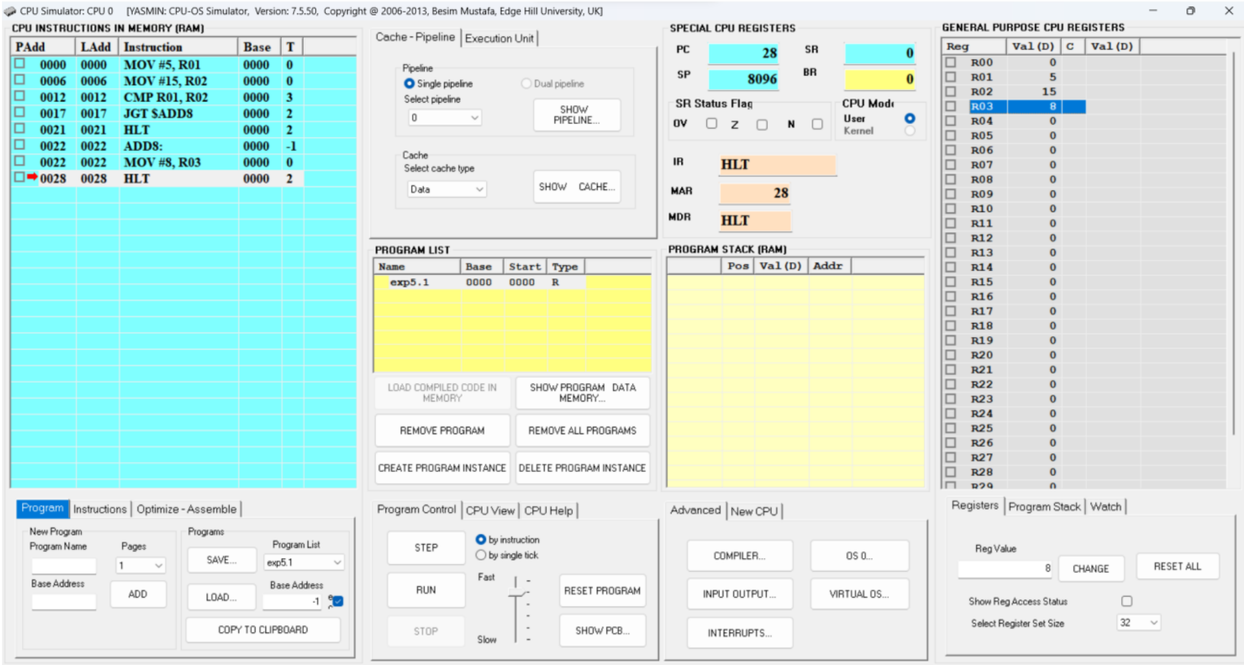


Fig.1: CPU Simulator Window

Case 2: R02 < R01

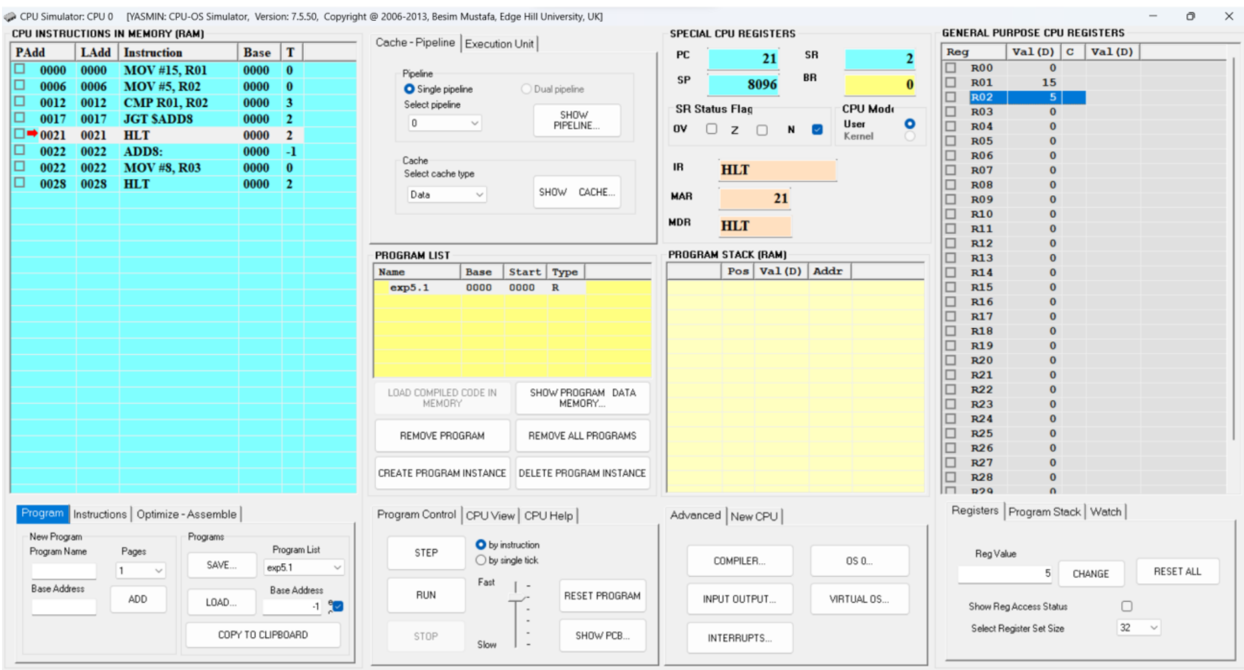


Fig.2: CPU Simulator Window

Program no. 2

Algorithm:

1. Initialize R01 to 0
2. Compare R01 with 0

3. If R01 equals 0, go to IF statement
4. If R01 does not equal 0, go to ELSE statement
5. At IF statement, set R03 to 5
6. At ELSE statement, add 1 to R01 and set R03 to R01
7. Halt the program.

Assembly Language code:

MOV #0, R01 *;Initialize R01 to 0*

CMP #0, R01 *;Compare R01 with 0*

JEQ $IF ;*If R01 equals 0, go to IF statement*

JNE $ELSE *;If R01 does not equal 0, go to ELSE statement*

$IF: *;At IF statement, set R03 to 5*

MOV #5, R03

HLT *;Halt the program*

$ELSE: *;At ELSE statement, add 1 to R01 and set R03 to R01*

ADD #1, R01

MOV R01, R03

HLT *;Halt the program*

Result:

Case 1: R01 = 0

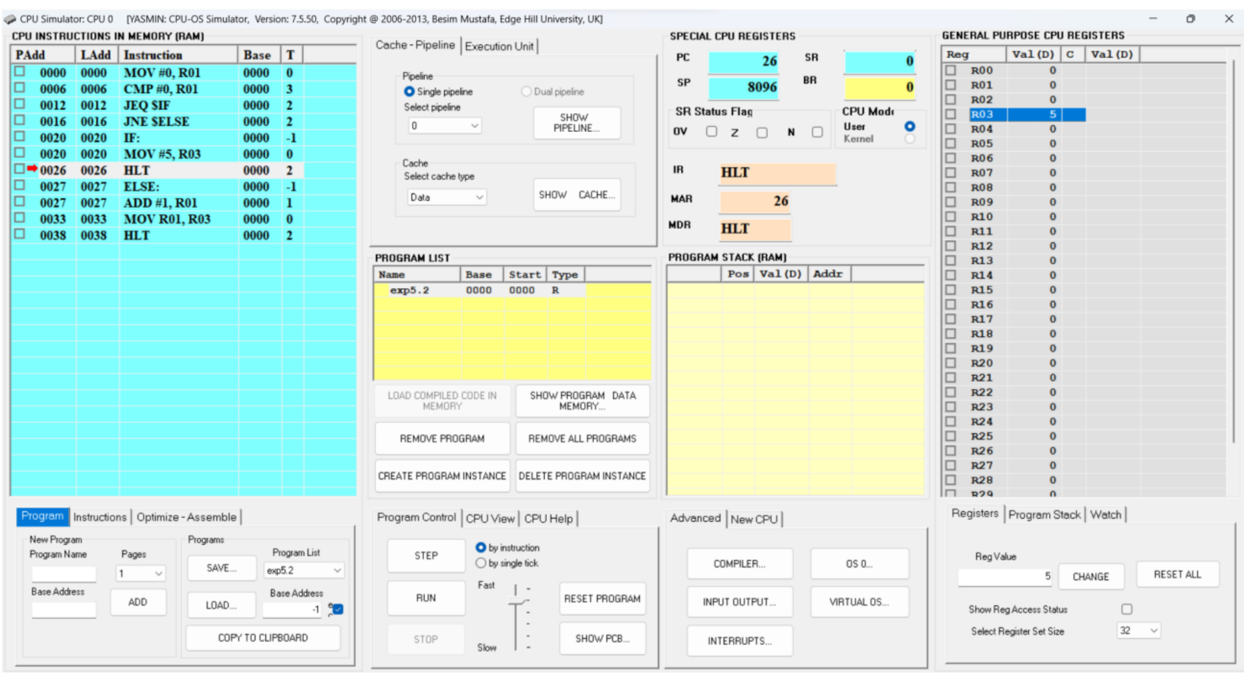


Fig.3: CPU Simulator Window

Case 2: R01 != 0

Date: 13-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

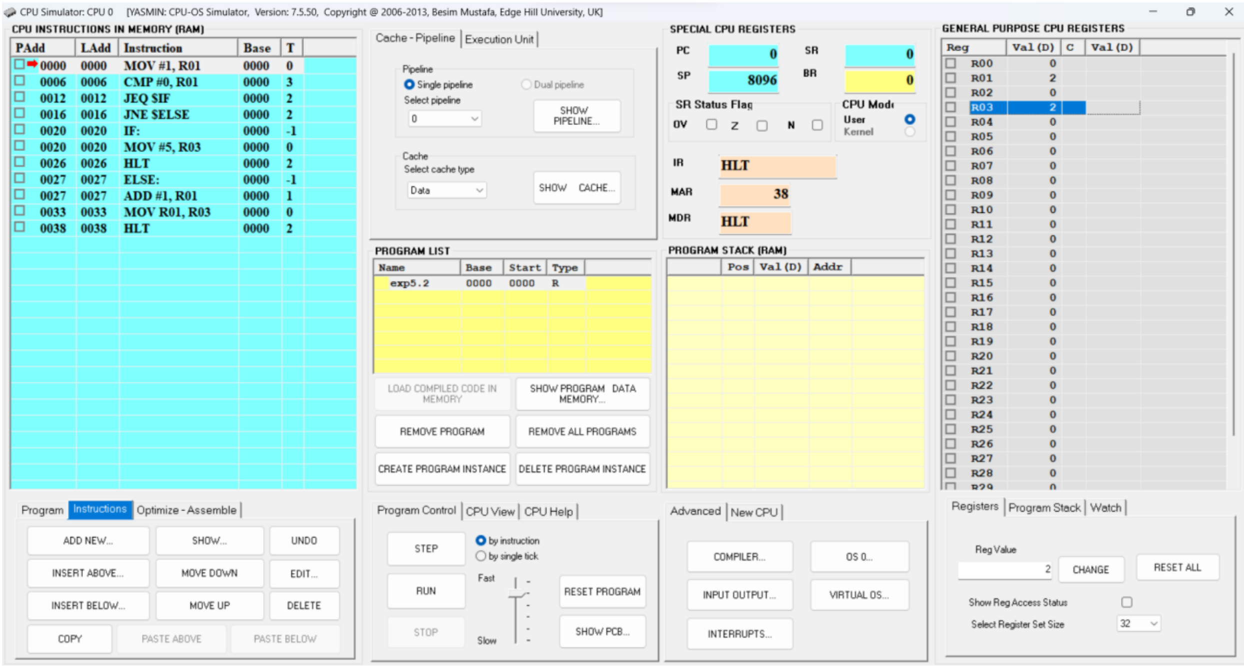


Fig.4: CPU Simulator Window

Program no. 3

Algorithm:

1. Initialize the value of R01 to 1.
2. Initialize the value of R02 to 5.
3. Repeat the following steps until R01 is equal to 5:
    a. Increment R02 by 2.
    b. Increment R01 by 1.
4. Halt the program.

Assembly Language code:

MOV #1, R01 *;move the value 1 into register R01*
MOV #5, R02 *;move the value 5 into register R02*
LOOP: *;label the start of the loop as "LOOP"*
ADD #2, R02 *;add the value 2 to register R02*
ADD #1, R01 *;add the value 1 to register R01*
CMP #5, R01 *;compare the value in register R01 to 5*
JNE $LOOP *;jump to "LOOP" if R01 is not equal to 5*
HLT *;halt the program*

Result:

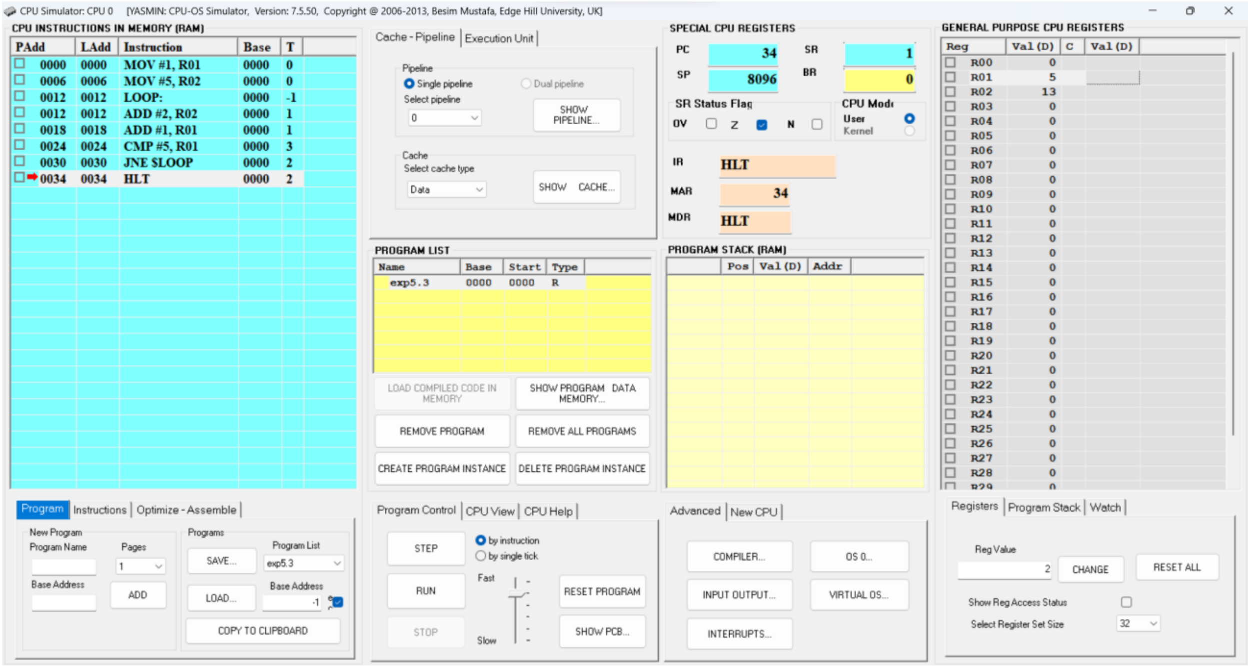


Fig.5: CPU Simulator Window

Program no. 4

Algorithm:

1. Initialize R04 to 8.
2. Start the loop: "LOOP"
3. Subtract 1 from R04.
4. Compare R04 to 0.
5. If R04 is not equal to 0, go back to the loop.
6. If R04 is equal to 0, halt the program.

Assembly Language code:

MOV #8, R04 *;Move value 8 into register R04*
LOOP: *;Label for the start of the loop*
SUB #1, R04 *;Subtract value 1 from the contents of R04*
CMP #0, R04 *;Compare the contents of R04 with value 0*
JNE LOOP *;If R04 is not equal to 0, jump back to the label LOOP*
HLT *;Halt the program execution.*

Result:

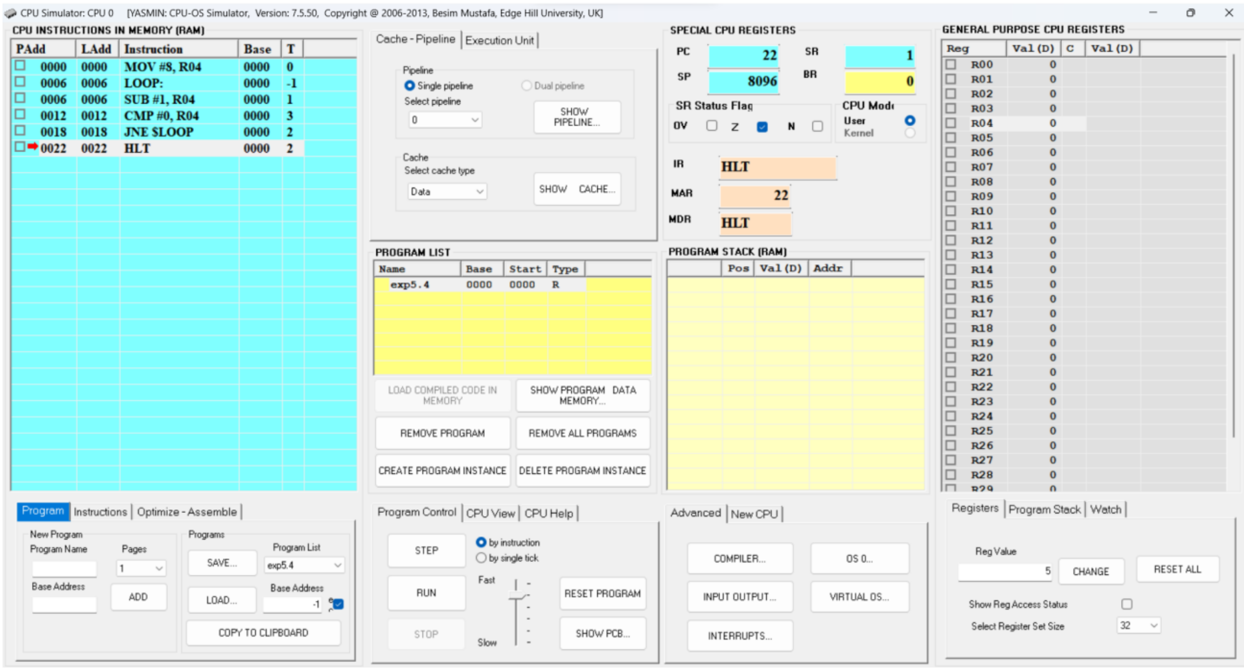


Fig.6: CPU Simulator Window

Program no. 5

Algorithm:

1. Initialize R05 to 0 and R09 to 12.
2. Start the loop with the label "LOOP".
3. Decrement R09 by 1.
4. Compare R05 and R09. If R05 is greater than R09, go to step 7.
5. If R05 is not greater than R09, repeat from step 3.
6. End the loop.
7. Halt the program.

Assembly Language code:

MOV #0, R05 *;Initialize R05 to 0*

MOV #12, R09 *;Initialize R09 to 12*

LOOP: *;Start the loop*

SUB #1, R09 *;Decrement R09 by 1*

CMP R05, R09 *;Compare R05 and R09*

JGT $LOOP *;If R05 is greater than R09, jump to the end of the loop*

HLT *;Halt the program*

Result:

Date: 13-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

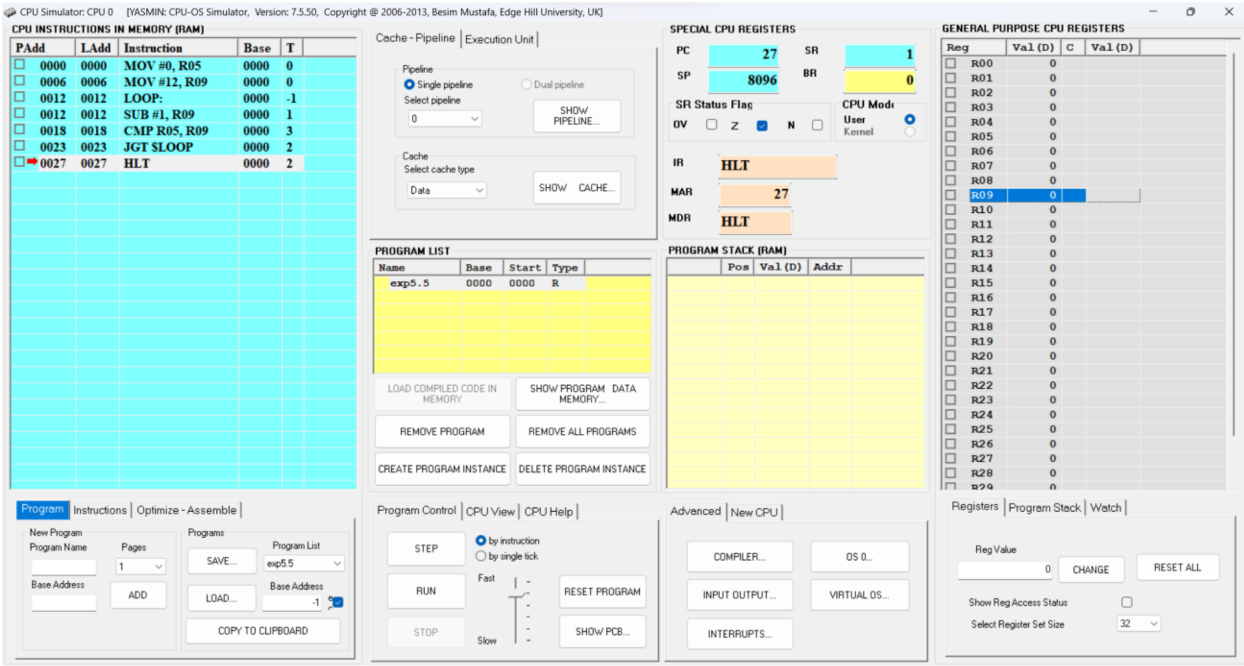


Fig.7: CPU Simulator Window

Date: 27-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

COA LAB
Experiment – 6

Write an assembly language program for the following problem statements:

1. A routine that pushes numbers 6 and 4 on top of the stack, then pops the two numbers one by one from stack, add them and pushes the result back to the top of stack.
2. First loop places 15 numbers from 1-15 on top of stack using push instruction. In the second loop, use the pop instruction to pop two numbers from the top of the stack, add them, and push the result back to the stack. Repeat the second loop until only one number is left on top of the stack.

Program no. 1

Algorithm:

1. Mark the start of the frame
2. Create a label ‘pushpopadd’ to represent the routine name.
3. Call routine ‘pushpopadd’.
4. Place the halt instruction below the call instruction and move the label ‘pushpopadd’ below the halt instruction.
5. Inside the label/routine ‘pushpopadd’, push numbers 6 and 4 on top of stack, then pop the two numbers one by one from stack, add them and push the result back to the top of stack.
6. Pop to any general-purpose register to get the return address on top of the stack
7. Returns to Halt instruction and exit.

Assembly Language code:

MSF *;Main start function*
CAL $pushpopadd *;Call the function pushpopadd*
HLT *;Halt the program execution*
pushpopadd: *;Define the function pushpopadd*
PSH #6 *;Push the value 6 onto the stack*
PSH #4 *;Push the value 4 onto the stack*
POP R01 *;Pop the top value from the stack into register R01*
POP R02 *;Pop the second value from the stack into register R02*
ADD R01, R02 *;Add the values in R01 and R02 and store the result in R02*
PSH R02 *;Push the result onto the stack*
POP R03 *;Pop the result from the stack into register R03*
RET *;Return from the function*

Result:

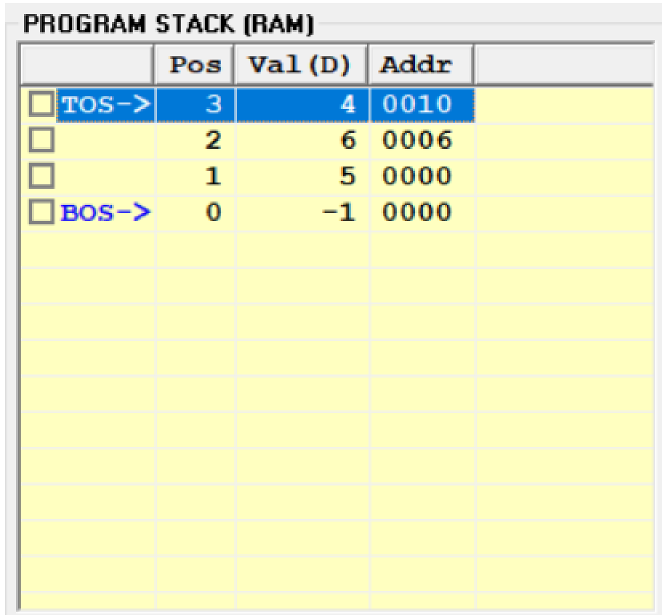


Fig-6.1.a: Program Stack window after pushing all the values

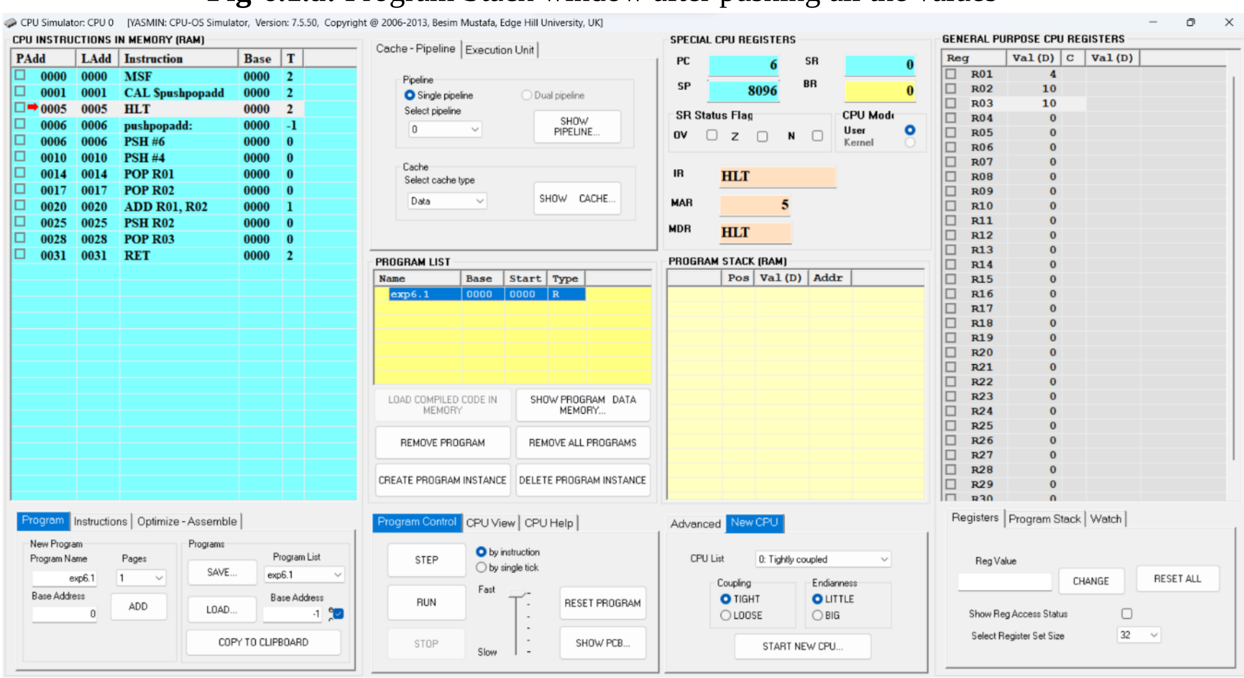


Fig-6.1.b: CPU Simulator Window

Program no. 2

Algorithm:

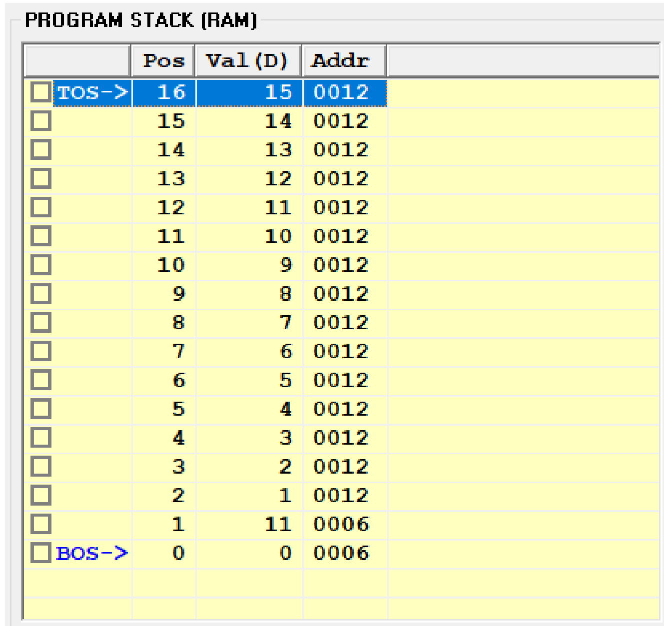
1. Initialize register R01 to 1 to hold the value of the first number to be pushed onto the stack.
2. Enter a loop to push 15 numbers onto the stack using the PUSH instruction, incrementing R01 each time to hold the value of the next number.

3. After the first loop completes, enter a second loop to pop two numbers from the top of the stack, add them, and push the result back to the stack.
4. Repeat the second loop until only one number is left on top of the stack.
5. The final result will be the value left on top of the stack.
6. Halt the program after the second loop completes.

Assembly Language code:

MOV #1, R01 *;Initialize register R01 to 1*
MSF *;Main start function*
CAL $pushpoploop *;Call the function pushpoploop*
HLT *;Halt the program execution*
pushpoploop: *;Define the function pushpoploop*
PSH R01 *;Push the value of register R01 onto the stack*
INC R01 *;Incrementing value of R01 by 1*
CMP #15, R01 *;Compare value of R01 with 15*
JLE $pushpoploop *;Jump back to pushpoploop if R01 <= 15*
addloop *;Define the function* addloop
POP R01 *;Pop the top value from the stack into register R01*
POP R02 *;Pop the second value from the stack into register R02*
ADD R01, R02 *;Add the values in R01 and R02 and store the result in R02*
PSH R02 *;Push the value of register R02 onto the stack*
CMP #120, R02 *;Compare value of R02 with 120*
JNE $addloop *;Jump back to addloop if the result is not equal to 120*
POP R03 *;Pop the final result from the top of the stack*
RET *; Return from the subroutine*

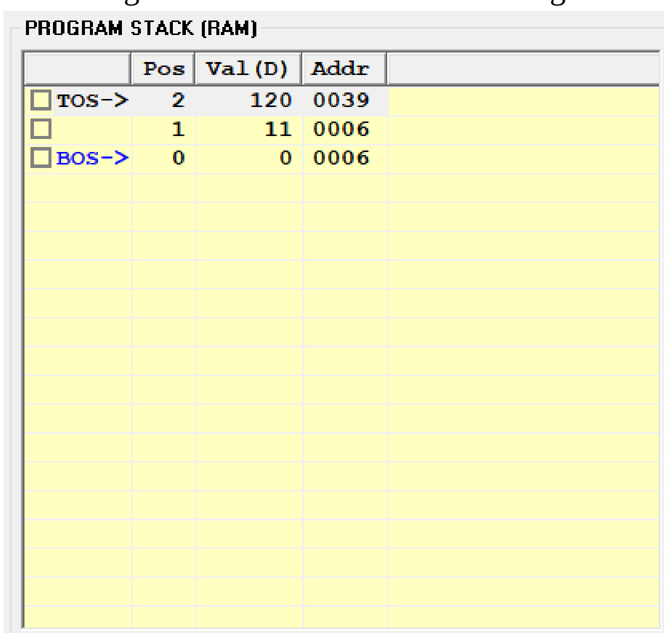
Result:



PROGRAM STACK (RAM)

	Pos	Val (D)	Addr
TOS->	16	15	0012
	15	14	0012
	14	13	0012
	13	12	0012
	12	11	0012
	11	10	0012
	10	9	0012
	9	8	0012
	8	7	0012
	7	6	0012
	6	5	0012
	5	4	0012
	4	3	0012
	3	2	0012
	2	1	0012
	1	11	0006
BOS->	0	0	0006

Fig-6.2.a: Program Stack window before exiting the first loop



PROGRAM STACK (RAM)

	Pos	Val (D)	Addr
TOS->	2	120	0039
	1	11	0006
BOS->	0	0	0006

Fig-6.2.b: Program Stack window before exiting the second loop

Date: 27-02-2023 **Name:** Ashvath S.P **Reg No**: 2162014

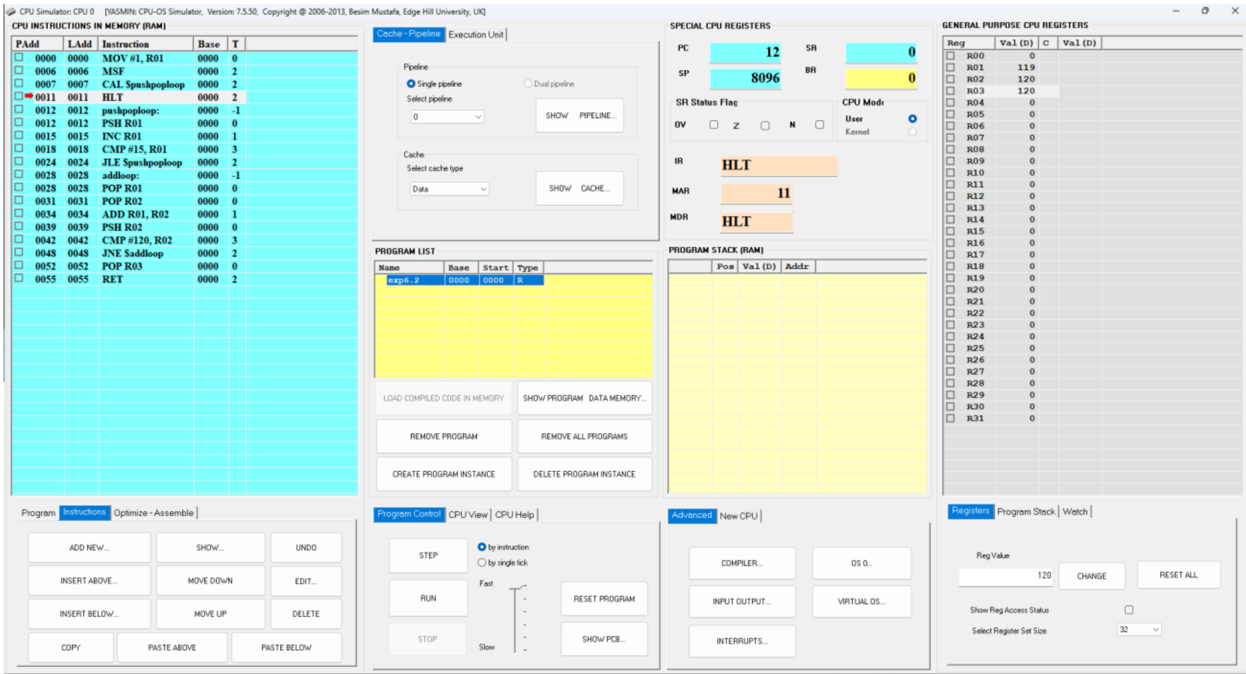


Fig-6.2.c: CPU Simulator Window