COMPUTER SYSTEM ARCHITECHTURE

PRACTICAL FILE

RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

DSC – 02 : Computer System Architecture
SEMESTER – 1
SESSION – 2025–26

Submitted By:-

Name: Panshul Vikram Aggarwal
College Roll No.: 25570072
University Roll No.: 25020570059

Course: B.Sc (Hons)
Computer Science

Submitted To:-

Dr. Kamlesh Kumar Raghuvanshi
Assistant Professor ,Department of Computer Science,Ramanujan College,
University of Delhi,
CR Park, Main Road,
Block – H, Kalkaji, New Delhi
Pincode - 110019

Signature: ____________________________

ACKNOWLEDGEMENT

I would like to extend my deep gratitude to Dr. Kamlesh Kumar Raghuvanshi and Dr. Nikhil Kumar Rajput, Assistant Professors, Ramanujan College, University of Delhi, for their invaluable support and mentorship during the preparation of this practical file for “Computer System Architecture”. Their guidance, expertise, and encouragement have been a constant source of inspiration throughout this work.

I am equally thankful to my classmates and the faculty members of the Computer Science Department for their consistent help, cooperation, and constructive feedback, which have enriched my learning experience and understanding of the subject.

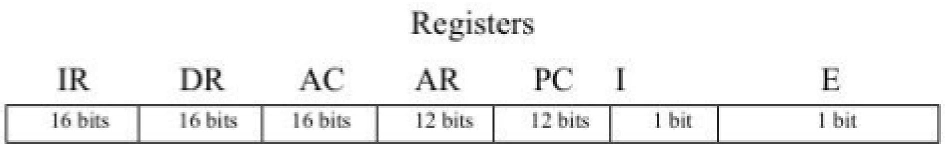
I also wish to acknowledge my family and friends for their continued motivation and belief in me. Their support has been a driving force that helped me complete this practical file with confidence and dedication.

Candidate Sign. _______________

INDEX

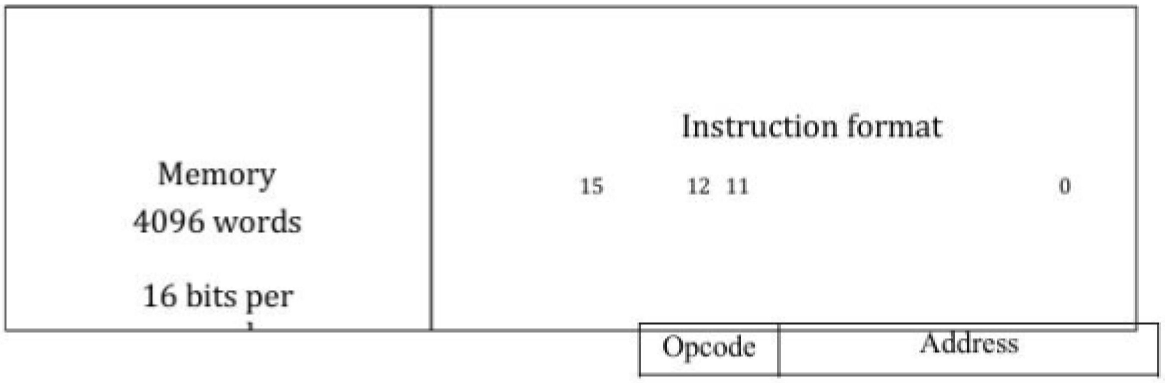
Serial No.	Topic	Page No.	Signature
1.	Q1. Creating a NEW CPUSIM Machine based on certain defined architecture.	1-2	
2.	Q2. Create a Fetch routine of the instruction cycle.	3	
3.	Q3. Write an assembly program to simulate ADD operation on two user-entered numbers.	4	
4.	Q4. Write an assembly program to simulate SUBTRACT operation on two user-entered numbers	5-6	
5.	Q5. Write an assembly program to simulate the following logical operations on two user entered numbers: i. AND ii. OR iii. NOT iv. XOR v. NOR vi. NAND	7-8	
6.	Q6. Write an assembly program for simulating following memory-reference instructions: i. ADD ii. BUN iii. ISZ iv. LDA v. STA	9-10	
7.	Q7. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers indecimal after the execution: i. CLA ii. CMA iii. CME iv. HLT	11	
8.	Q8. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. INC ii. SPA iii. SNA iv. SZE	12	
9.	Q9. Check the diagonalizable property… find the corresponding eigenvalue and verify the Cayley-Hamilton theorem.	13	
10.	Q10. Write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).	14	
11.	Q11. Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.	15	

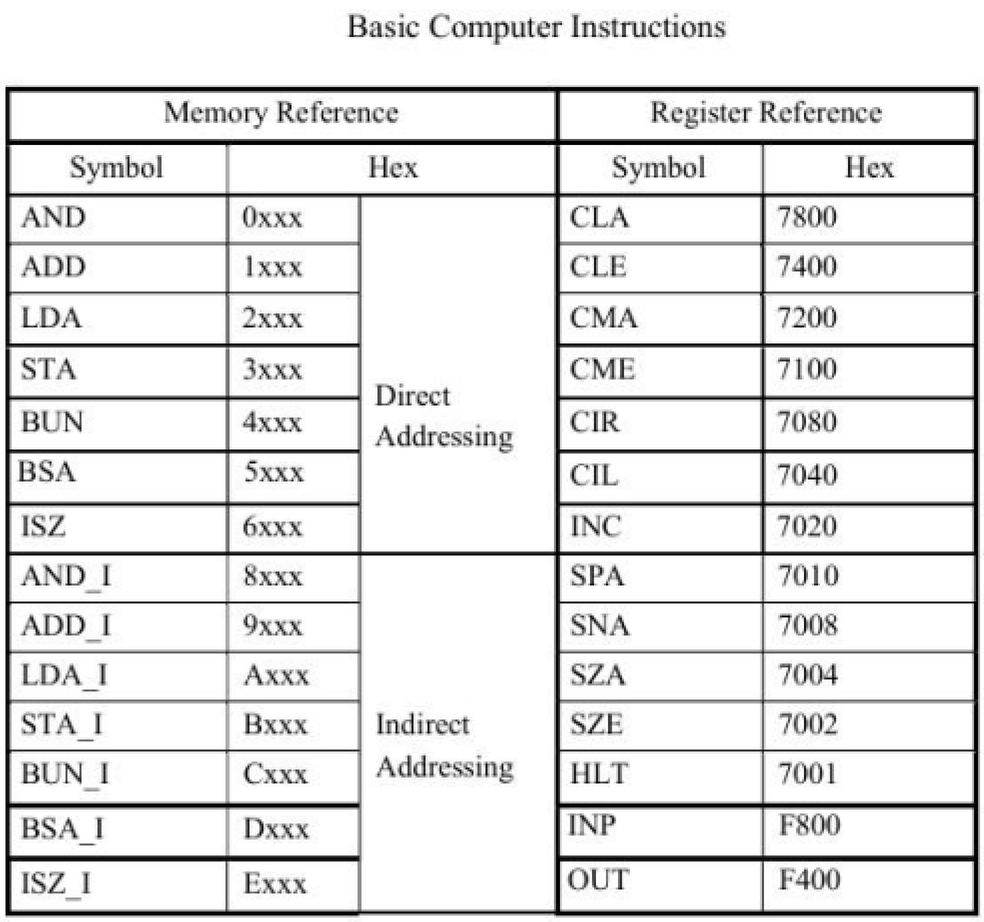
Q1. Creating a NEW CPUSIM Machine based on certain defined architecture.



Registers

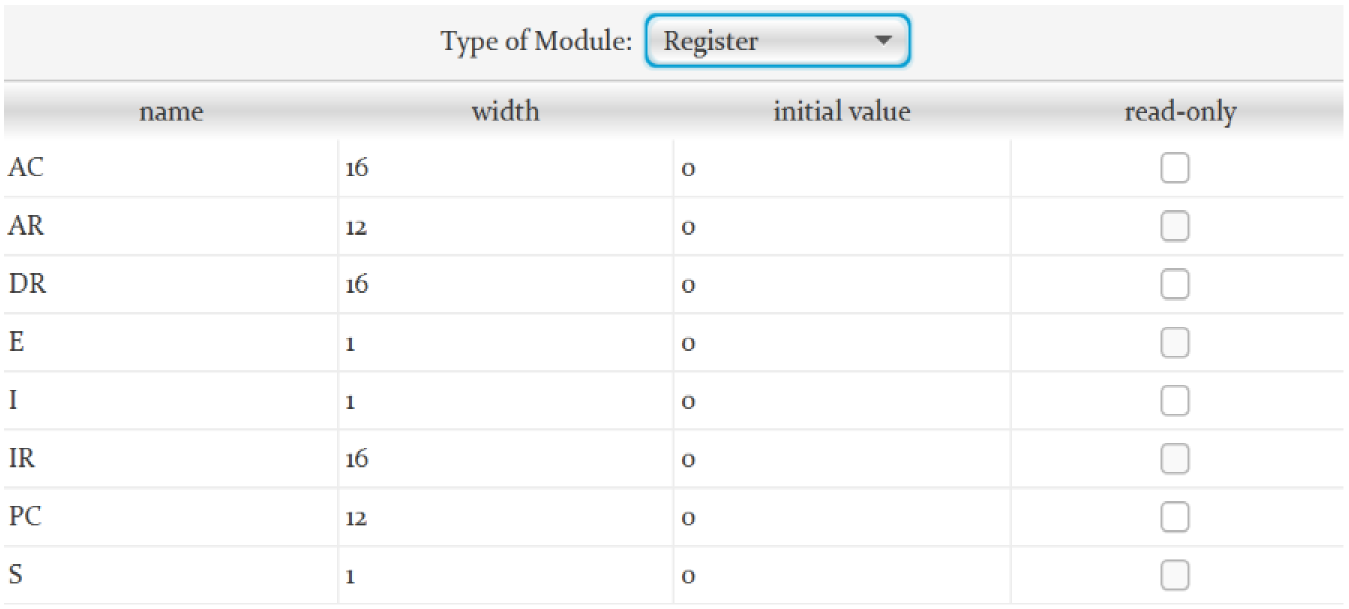
IR	DR	AC	AR	PC	I	E
16 bits	16 bits	16 bits	12 bits	12 bits	1 bit	1 bit





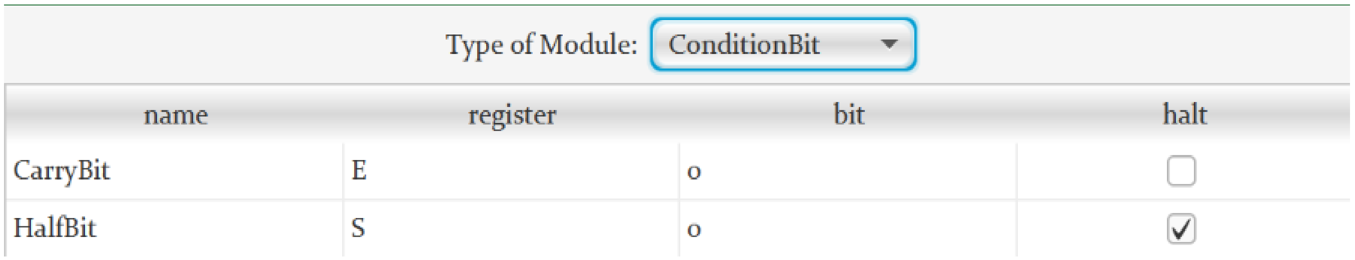
Basic Computer Instructions

Memory Reference			Register Reference	
Symbol	Hex		Symbol	Hex
AND	0xxx		CLA	7800
ADD	1xxx		CLE	7400
LDA	2xxx		CMA	7200
STA	3xxx	Direct Addressing	CME	7100
BUN	4xxx		CIR	7080
BSA	5xxx		CIL	7040
ISZ	6xxx		INC	7020
AND_I	8xxx		SPA	7010
ADD_I	9xxx		SNA	7008
LDA_I	Axxx		SZA	7004
STA_I	Bxxx	Indirect Addressing	SZE	7002
BUN_I	Cxxx		HLT	7001
BSA_I	Dxxx		INP	F800
ISZ_I	Exxx		OUT	F400



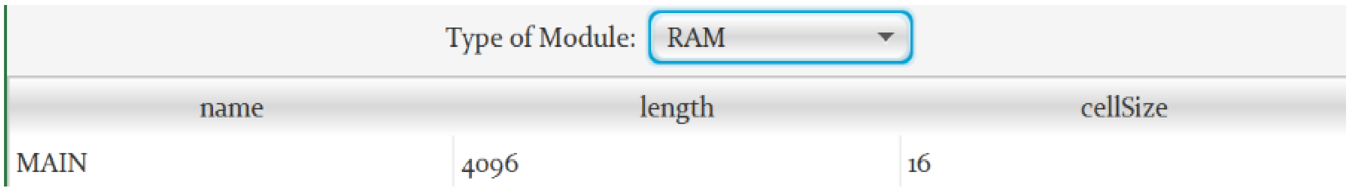
Type of Module: Register

name	width	initial value	read-only
AC	16	0	☐
AR	12	0	☐
DR	16	0	☐
E	1	0	☐
I	1	0	☐
IR	16	0	☐
PC	12	0	☐
S	1	0	☐



Type of Module: ConditionBit

name	register	bit	halt
CarryBit	E	0	☐
HalfBit	S	0	✓

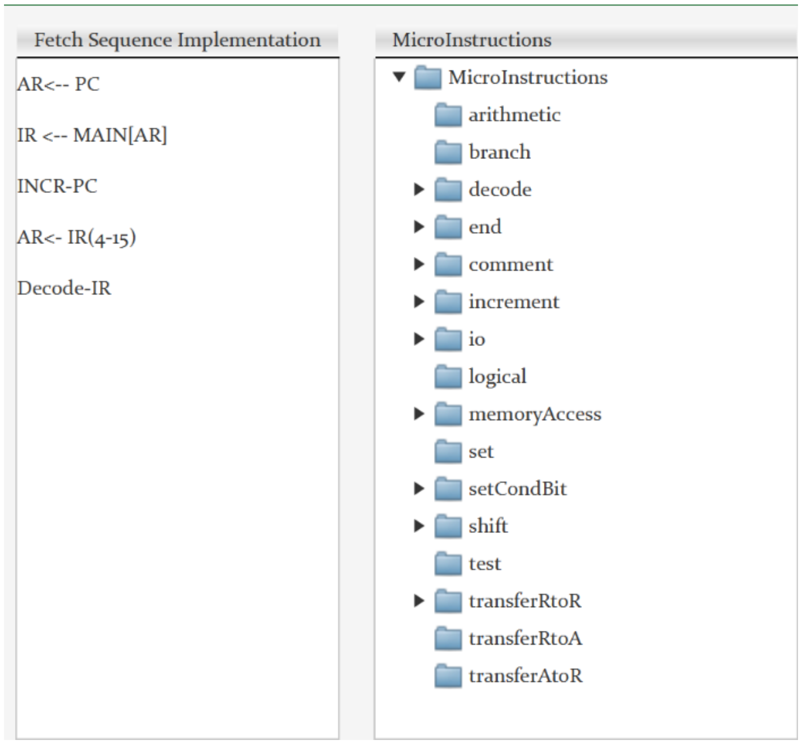


Type of Module: RAM

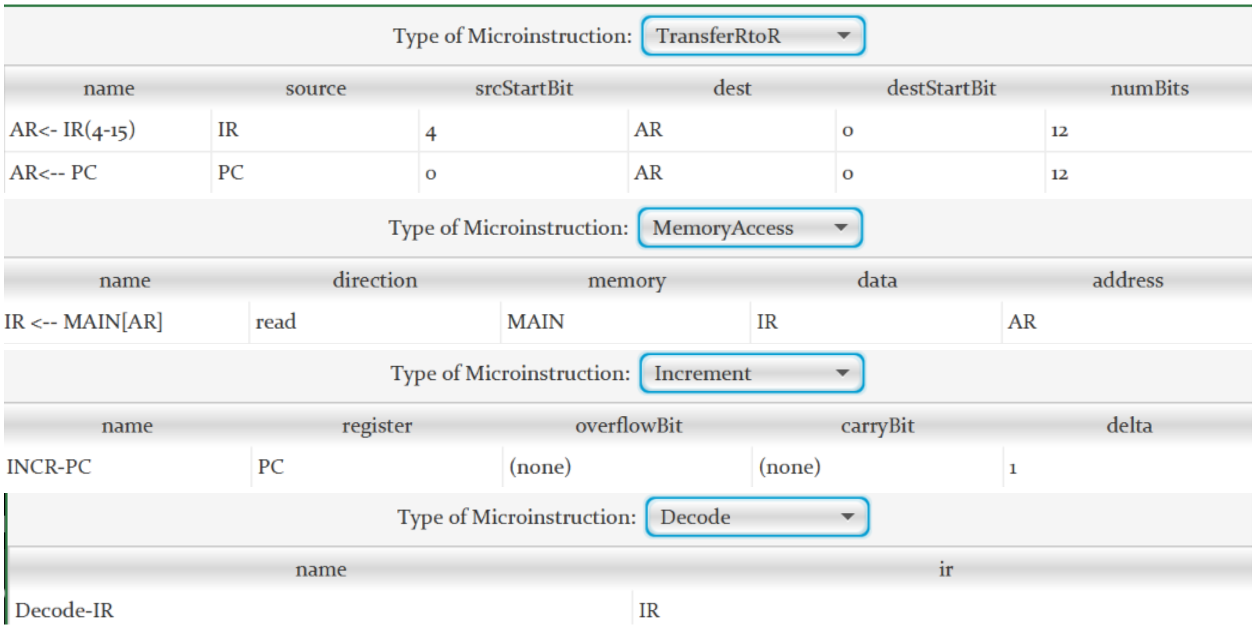
name	length	cellSize
MAIN	4096	16

CLA
CMA
CME
HLT
HLT
ADD
LDA
STA
BUN
ISZ
INC
SPA
SNA
SZE
CIR
CIL

Q2. Create a Fetch routine of the instruction cycle.



Fetch Sequence Implementation
AR<-- PC
IR <-- MAIN[AR]
INCR-PC
AR<- IR(4-15)
Decode-IR



Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AR<- IR(4-15)	IR	4	AR	0	12
AR<-- PC	PC	0	AR	0	12

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
IR <-- MAIN[AR]	read	MAIN	IR	AR

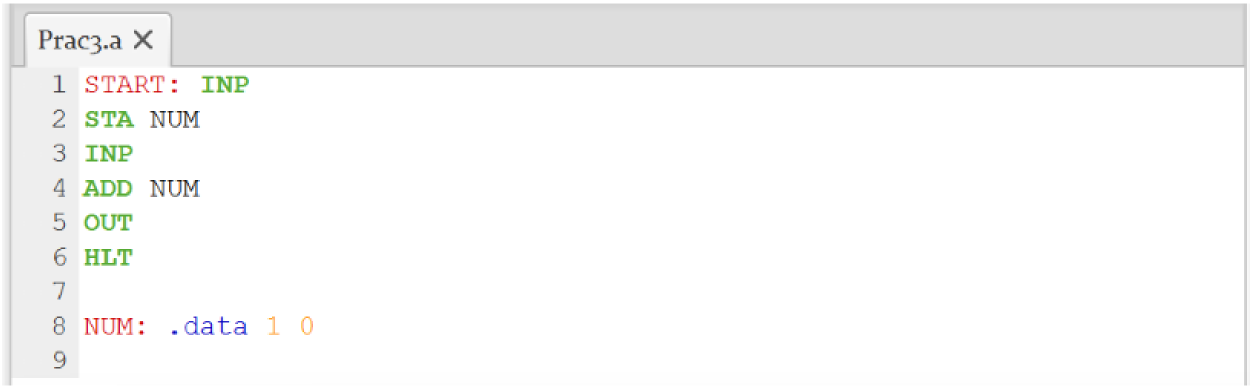
Type of Microinstruction: Increment

name	register	overflowBit	carryBit	delta
INCR-PC	PC	(none)	(none)	1

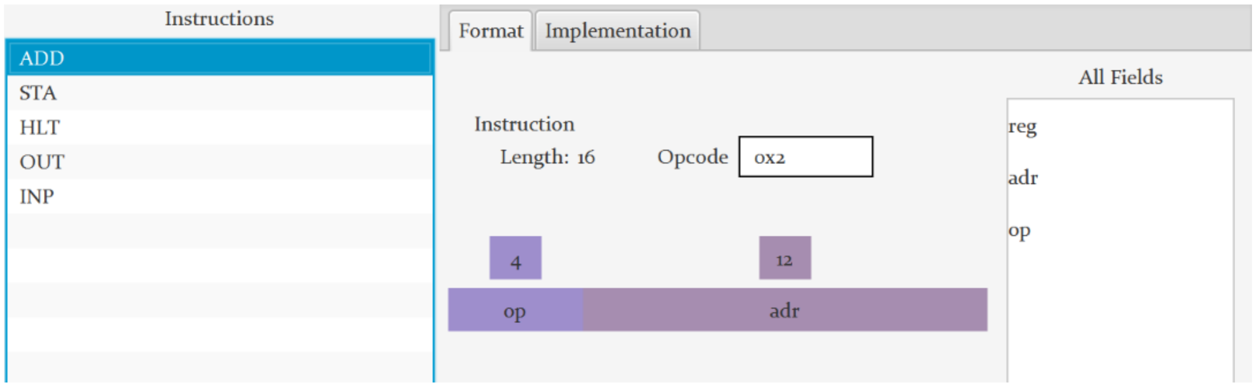
Type of Microinstruction: Decode

name	ir
Decode-IR	IR

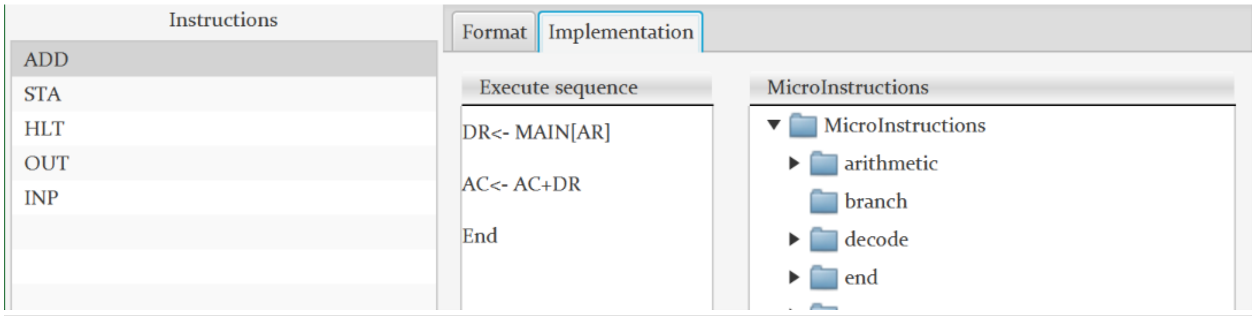
Q3. Write an assembly program to simulate ADD operation on two user-entered numbers.



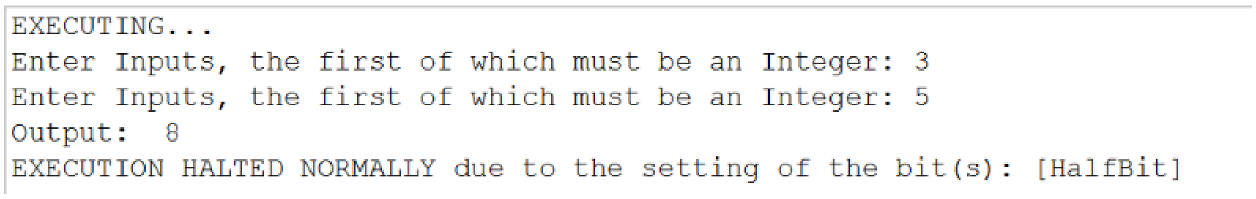
```
START: INP
STA NUM
INP
ADD NUM
OUT
HLT

NUM: .data 1 0

```



Instruction Length: 16 Opcode 0x2

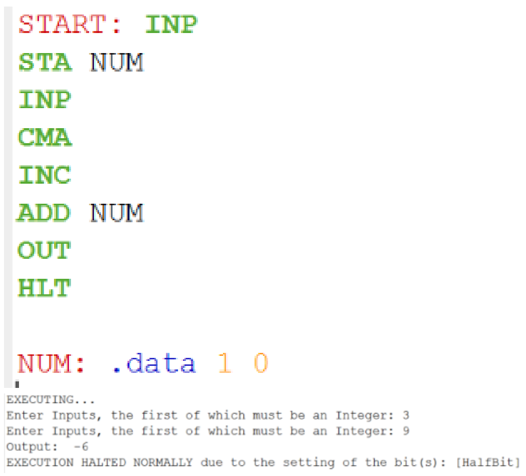


```
DR<- MAIN[AR]
AC<- AC+DR
End
```



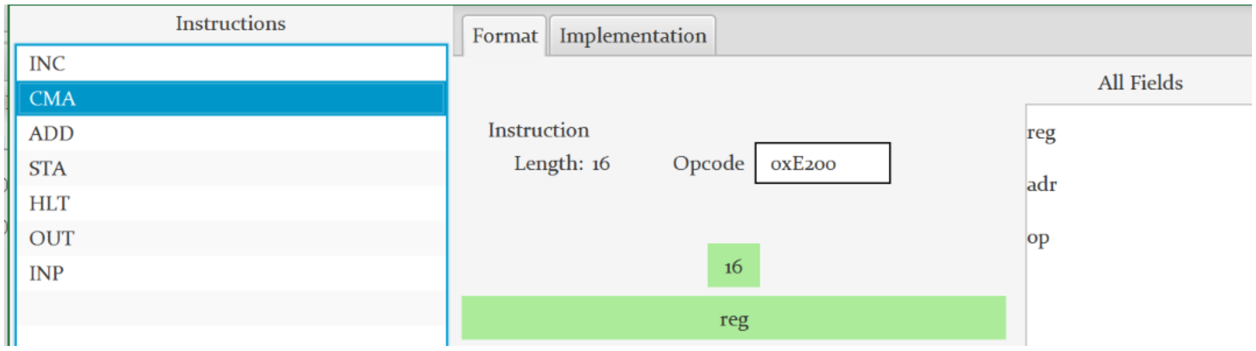
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 3
Enter Inputs, the first of which must be an Integer: 5
Output:  8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

Q4. Write an assembly program to simulate SUBTRACT operation on two user-entered numbers



```
START:  INP
STA  NUM
INP
CMA
INC
ADD  NUM
OUT
HLT

NUM:  .data  1  0
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 3
Enter Inputs, the first of which must be an Integer: 9
Output:  -6
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

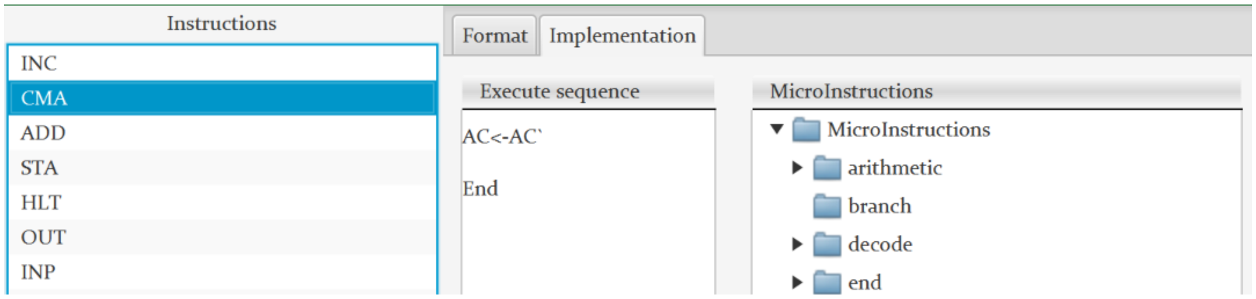


Instructions
INC
CMA
ADD
STA
HLT
OUT
INP

Format | Implementation

Instruction
Length: 16   Opcode: oxE200

All Fields: reg, adr, op

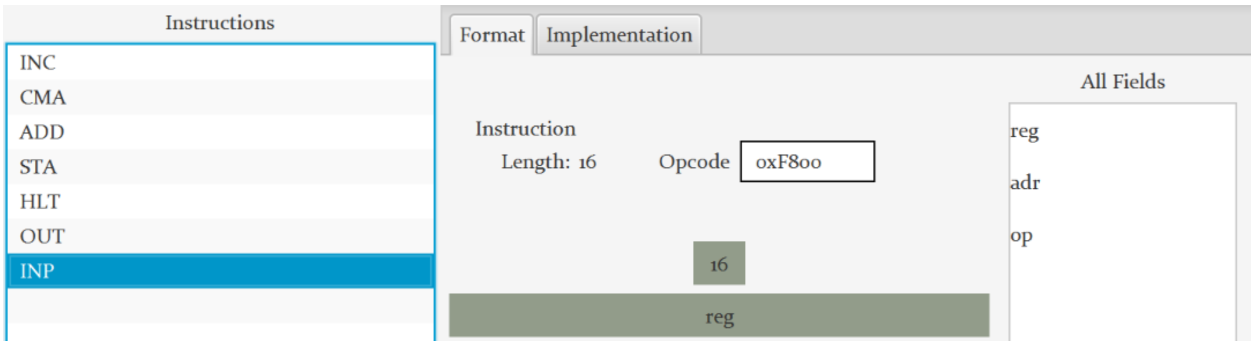


Instructions
INC
CMA
ADD
STA
HLT
OUT
INP

Format | Implementation

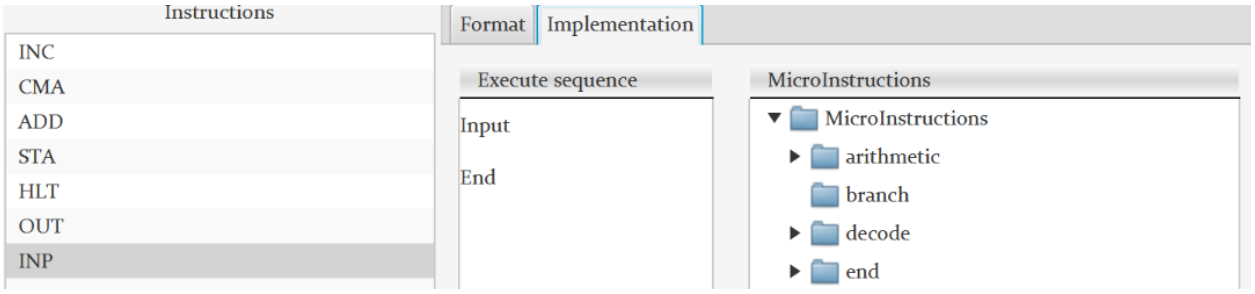
Execute sequence:

```
AC<-AC`
End
```



Instruction Length: 16 Opcode 0xF800

All Fields: reg, adr, op

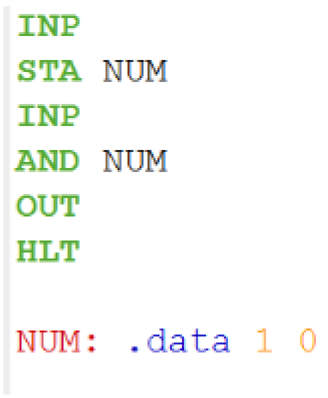


Execute sequence

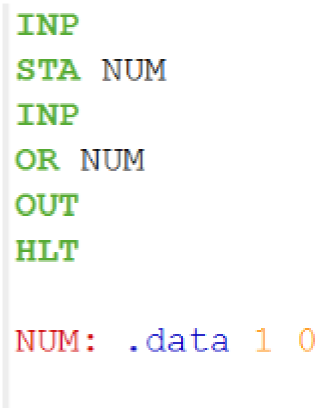
Input

End

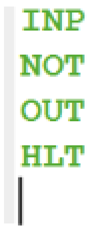
Q5. Write an assembly program to simulate the following logical operations on two userentered numbers: i. AND ii. OR iii. NOT iv. XOR v. NOR vi. NAND



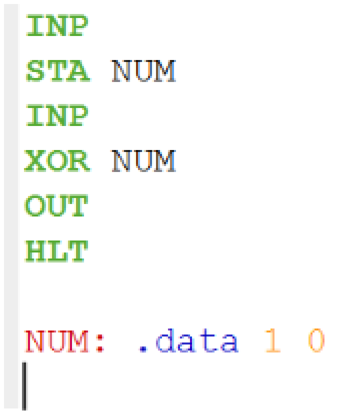
```
INP
STA NUM
INP
AND NUM
OUT
HLT

NUM: .data 1 0
```



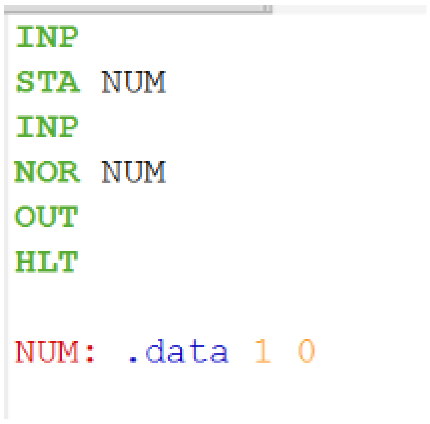
```
INP
STA NUM
INP
OR NUM
OUT
HLT

NUM: .data 1 0
```



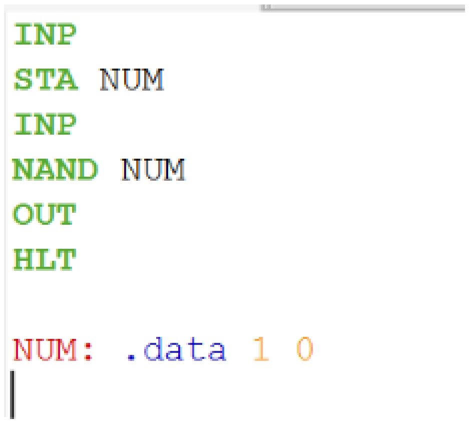
```
INP
NOT
OUT
HLT
```



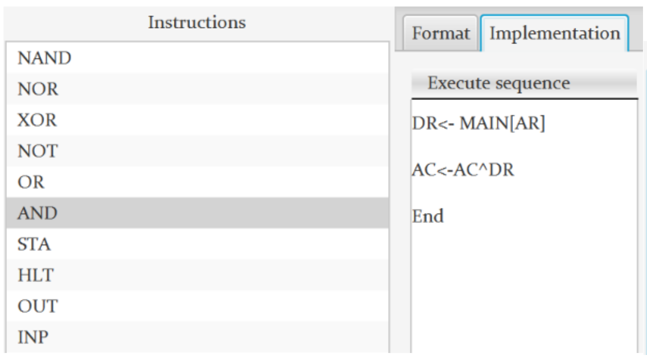
```
INP
STA NUM
INP
XOR NUM
OUT
HLT

NUM: .data 1 0
```



```
INP
STA NUM
INP
NOR NUM
OUT
HLT

NUM: .data 1 0
```

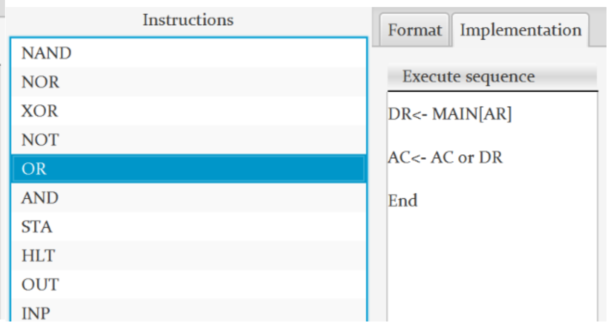


```
INP
STA NUM
INP
NAND NUM
OUT
HLT

NUM: .data 1 0
```



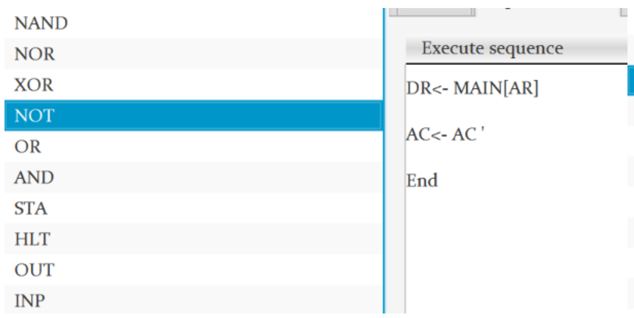
Execute sequence

```
DR<- MAIN[AR]
AC<-AC^DR
End
```



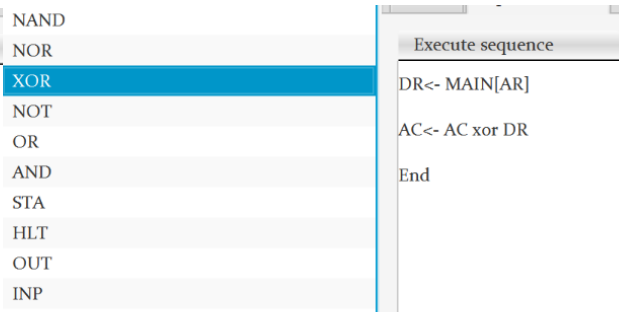
Execute sequence

```
DR<- MAIN[AR]
AC<- AC or DR
End
```



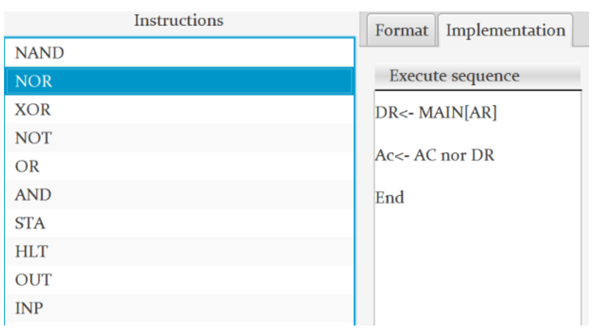
Execute sequence

```
DR<- MAIN[AR]
AC<- AC '
End
```



Execute sequence

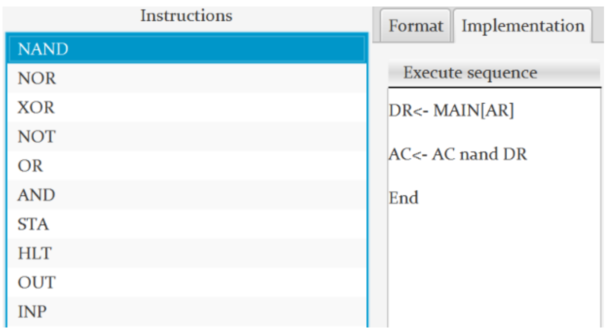
```
DR<- MAIN[AR]
AC<- AC xor DR
End
```



DR<- MAIN[AR]

Ac<- AC nor DR

End



DR<- MAIN[AR]

AC<- AC nand DR

End

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  0
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

```
EXECUTING...1
Enter Inputs, the first of which must be an Integer:1
Enter Inputs, the first of which must be an Integer: 1
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

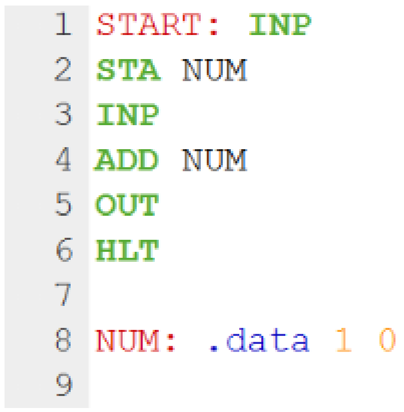
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Output:  -2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

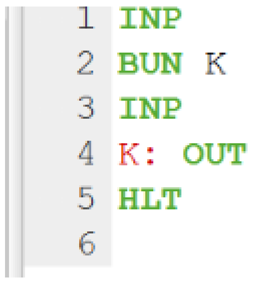
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  -2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 0
Enter Inputs, the first of which must be an Integer: 1
Output:  -1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

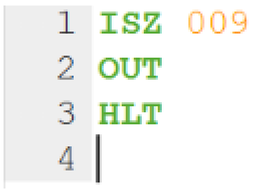
Q6. Write an assembly program for simulating following memory-reference instructions: i. ADD ii. BUN iii. ISZ iv. LDA v. STA



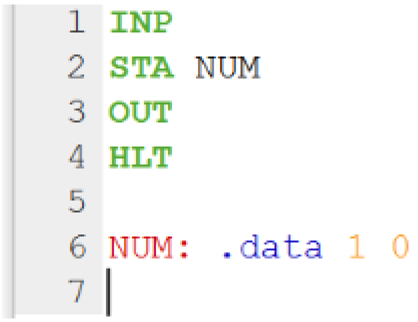
```
START:  INP
STA NUM
INP
ADD NUM
OUT
HLT

NUM:  .data 1 0

```



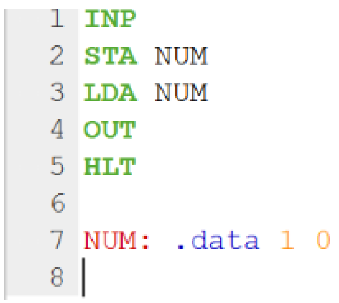
```
INP
BUN K
INP
K: OUT
HLT

```



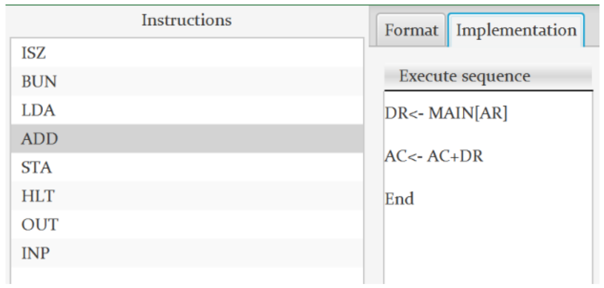
```
ISZ 009
OUT
HLT

```



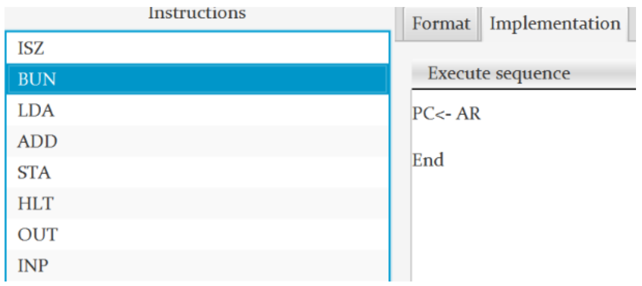
```
INP
STA NUM
OUT
HLT

NUM:  .data 1 0

```



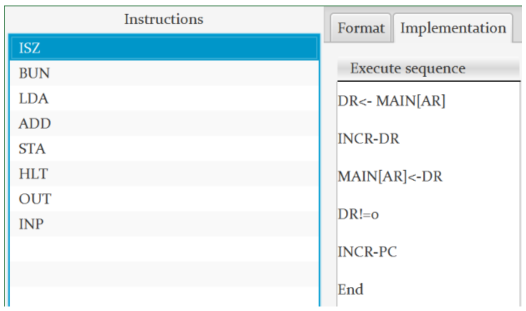
```
INP
STA NUM
LDA NUM
OUT
HLT

NUM:  .data 1 0

```



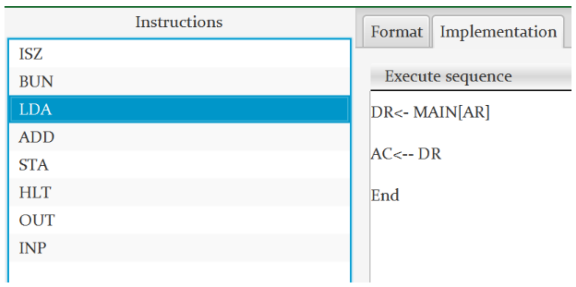
```
DR<- MAIN[AR]

AC<- AC+DR

End
```



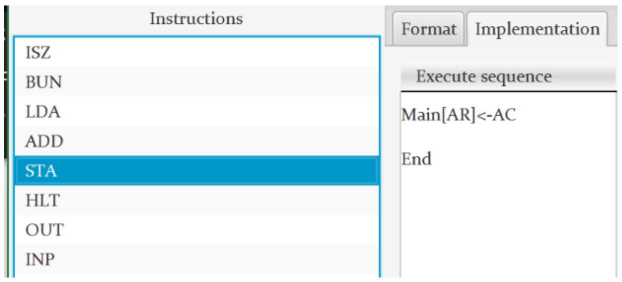
```
PC<- AR

End
```



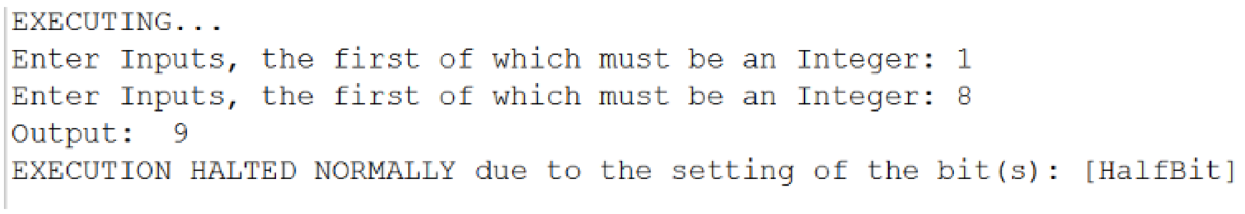
```
DR<- MAIN[AR]

INCR-DR

MAIN[AR]<-DR

DR!=o

INCR-PC

End
```



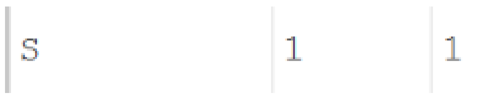
```
DR<- MAIN[AR]

AC<-- DR

End
```



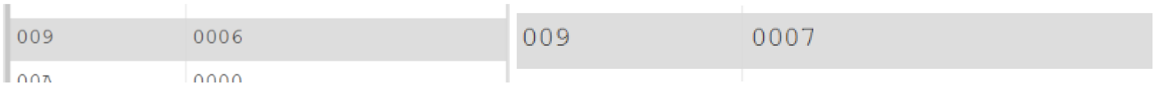
```
Main[AR]<-AC

End
```



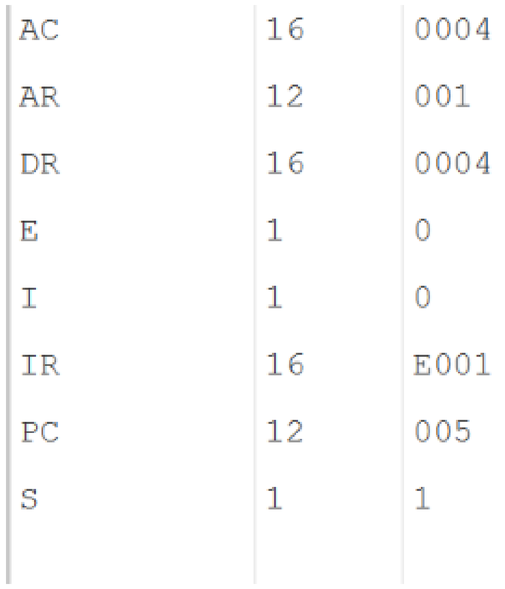
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 8
Output:  9
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```



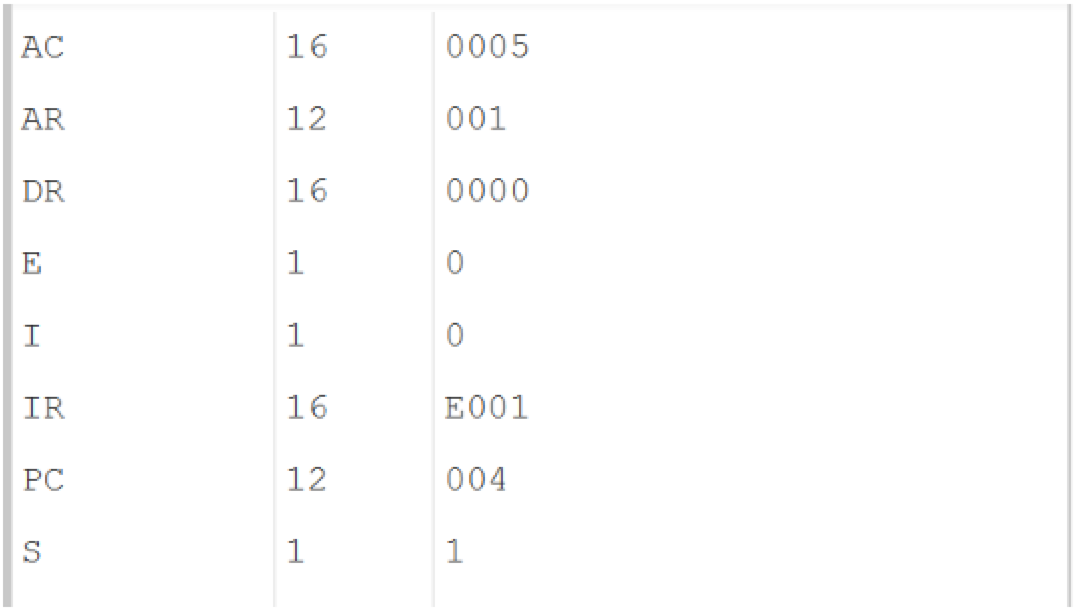
S	1	1



009	0006	009	0007

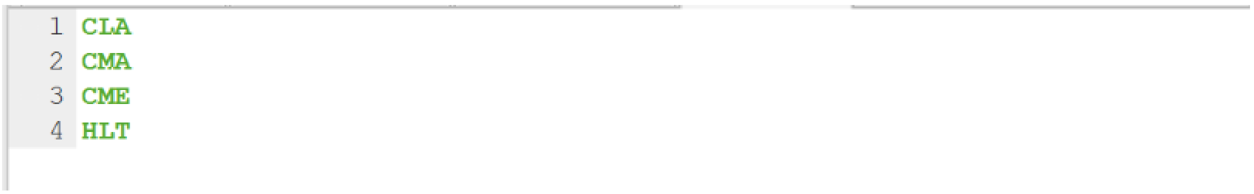


AC	16	0004
AR	12	001
DR	16	0004
E	1	0
I	1	0
IR	16	E001
PC	12	005
S	1	1

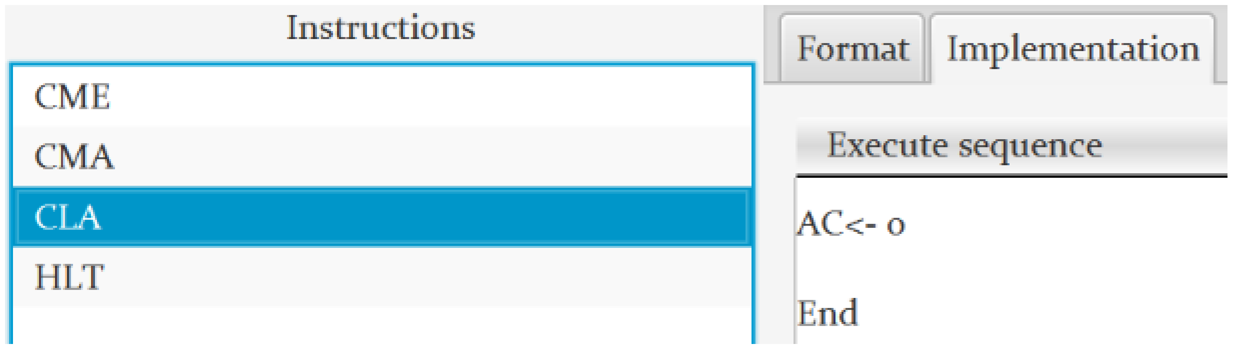


AC	16	0005
AR	12	001
DR	16	0000
E	1	0
I	1	0
IR	16	E001
PC	12	004
S	1	1

Q7. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers indecimal after the execution: i. CLA ii. CMA iii. CME iv. HLT

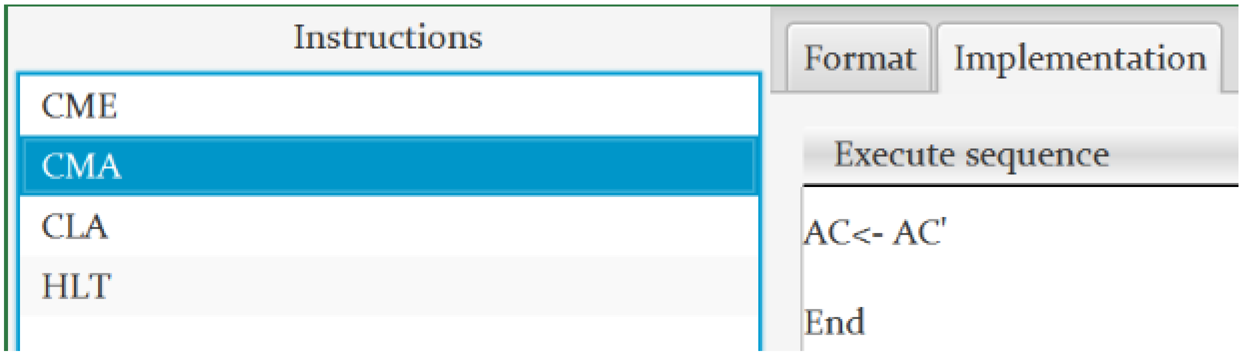


```
CLA
CMA
CME
HLT
```



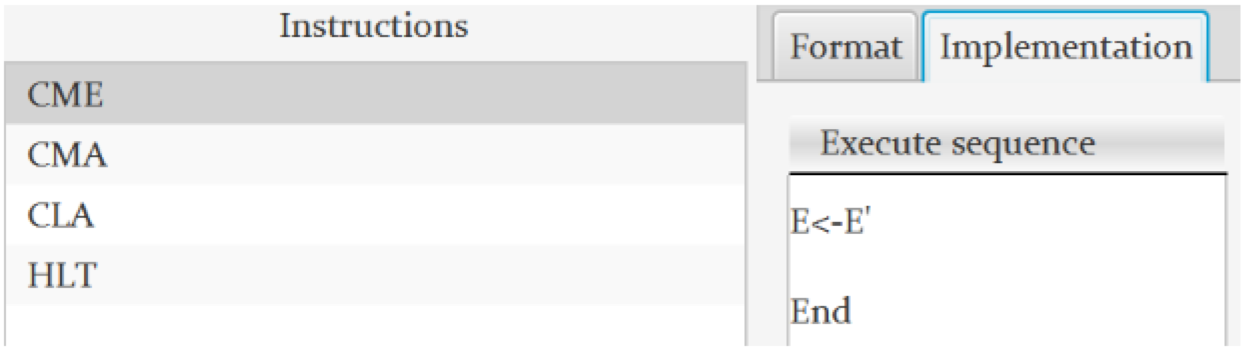
AC<- o

End



AC<- AC'

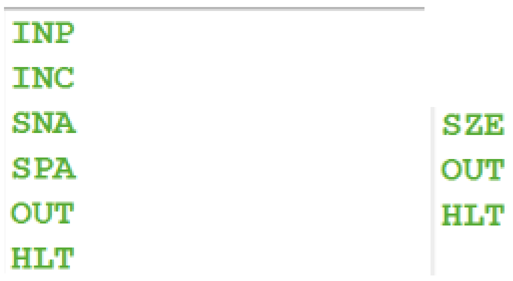
End



E<-E'

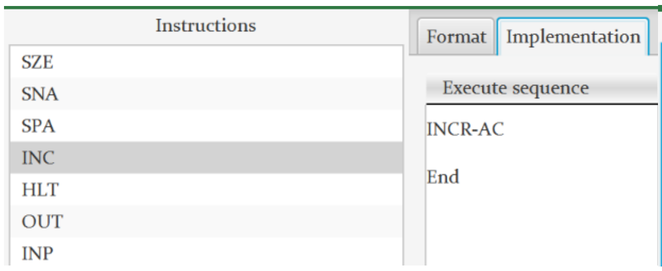
End

Q8. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. INC ii. SPA iii. SNA iv. SZE



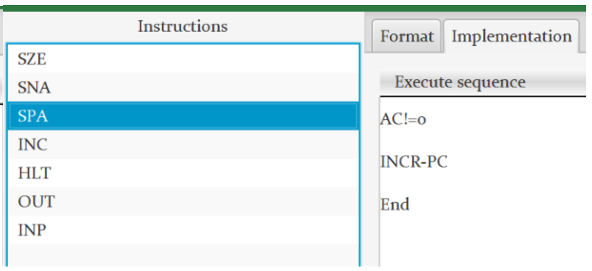
```
INP
INC
SNA
SPA
OUT
HLT
```

```
SZE
OUT
HLT
```



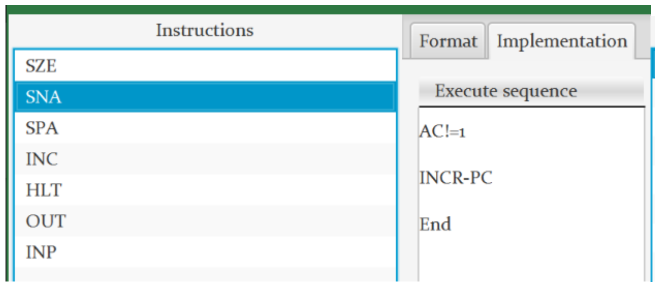
Execute sequence

```
INCR-AC
End
```



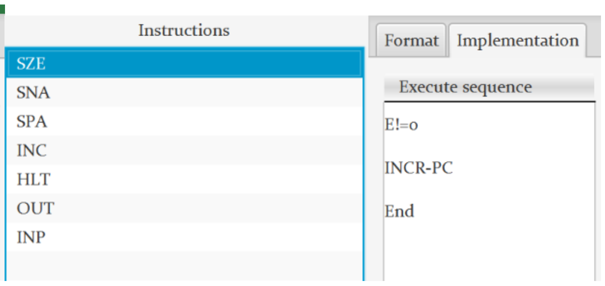
Execute sequence

```
AC!=o
INCR-PC
End
```



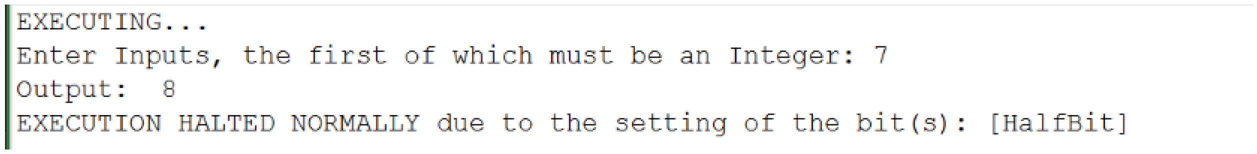
Execute sequence

```
AC!=1
INCR-PC
End
```

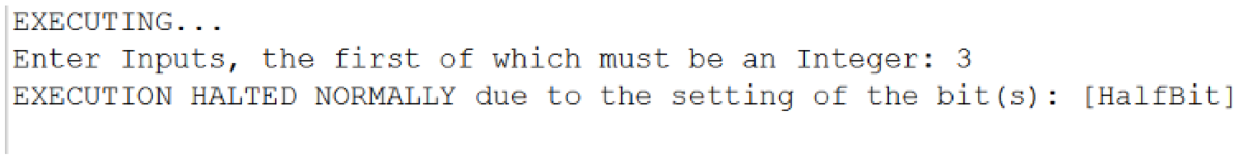


Execute sequence

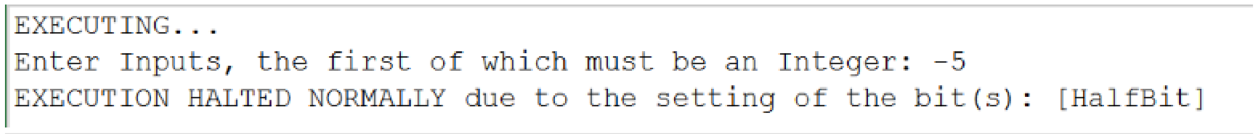
```
E!=o
INCR-PC
End
```



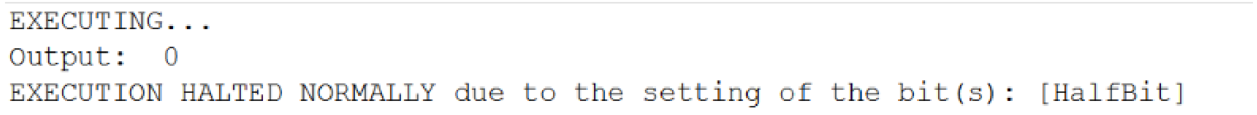
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 7
Output:  8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```



```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 3
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

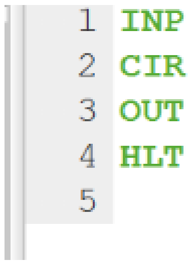


```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -5
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

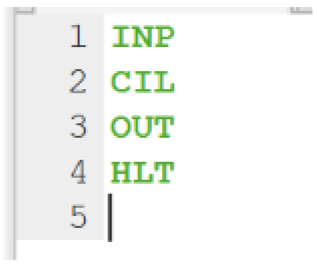


```
EXECUTING...
Output:  0
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

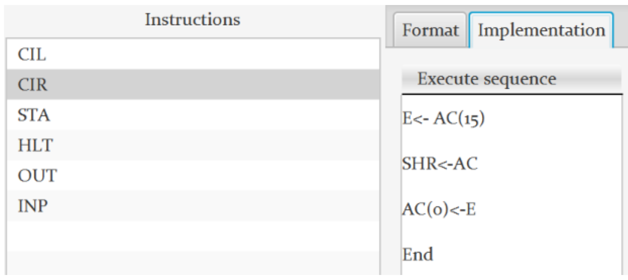
Q9. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. CIR ii. CIL



```
INP
CIR
OUT
HLT
```



```
INP
CIL
OUT
HLT
```

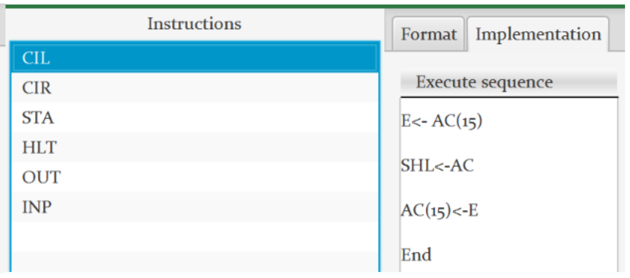


Instructions

CIL
CIR
STA
HLT
OUT
INP

Execute sequence

E<- AC(15)
SHR<-AC
AC(0)<-E
End

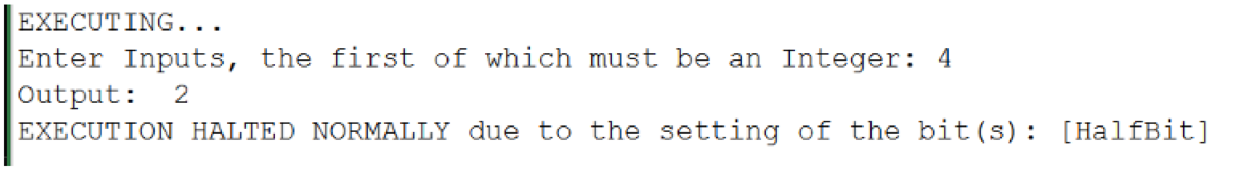


Instructions

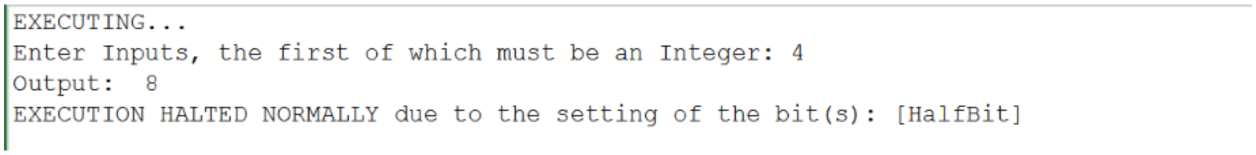
CIL
CIR
STA
HLT
OUT
INP

Execute sequence

E<- AC(15)
SHL<-AC
AC(15)<-E
End

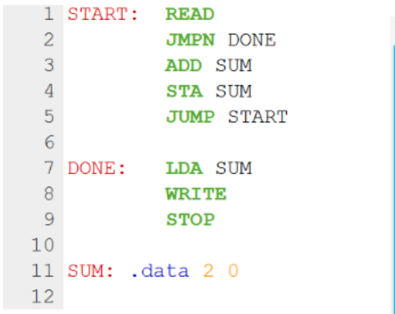


```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

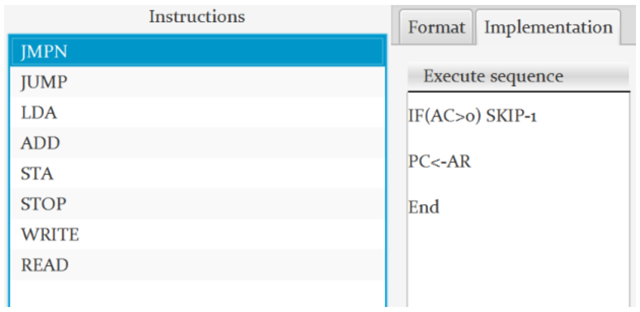


```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

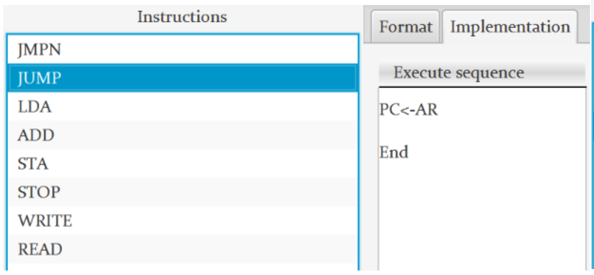
Q10. Write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).



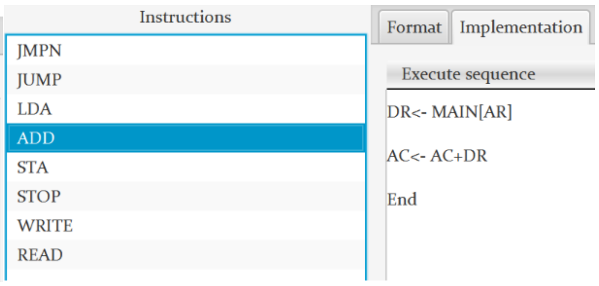
```
START:  READ
        JMPN DONE
        ADD SUM
        STA SUM
        JUMP START

DONE:   LDA SUM
        WRITE
        STOP

SUM: .data 2 0

```



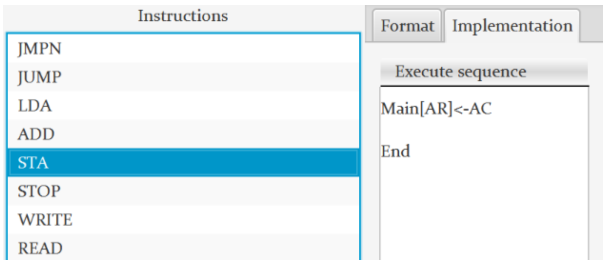
```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            IF(AC>o) SKIP-1
LDA             PC<-AR
ADD             End
STA
STOP
WRITE
READ
```



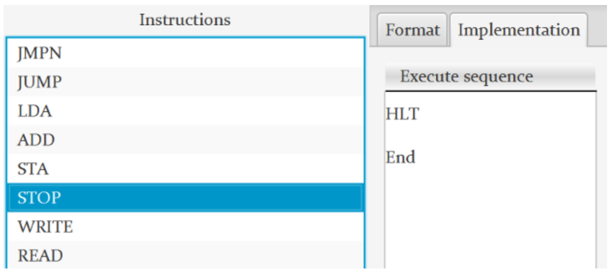
```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            PC<-AR
LDA             End
ADD
STA
STOP
WRITE
READ
```



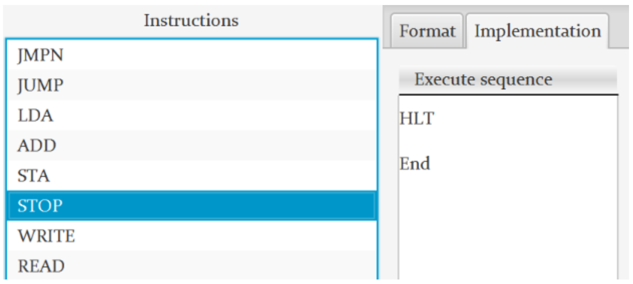
```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            DR<- MAIN[AR]
LDA             AC<- AC+DR
ADD             End
STA
STOP
WRITE
READ
```



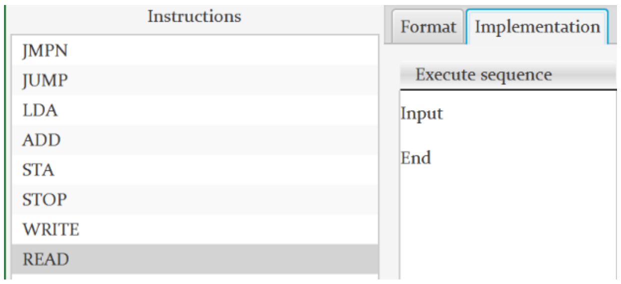
```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            Main[AR]<-AC
LDA             End
ADD
STA
STOP
WRITE
READ
```



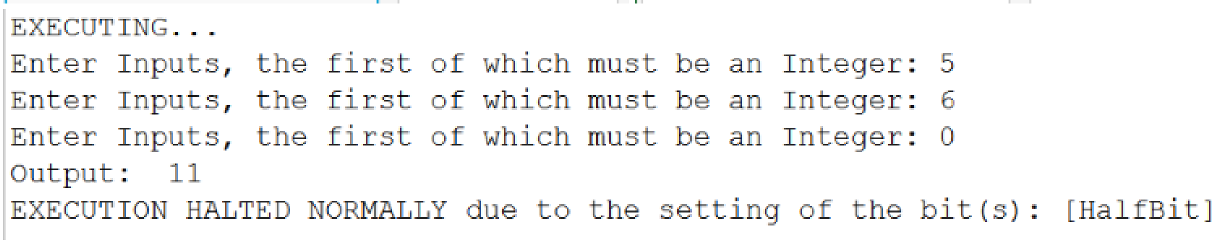
```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            HLT
LDA             End
ADD
STA
STOP
WRITE
READ
```



```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            HLT
LDA             End
ADD
STA
STOP
WRITE
READ
```

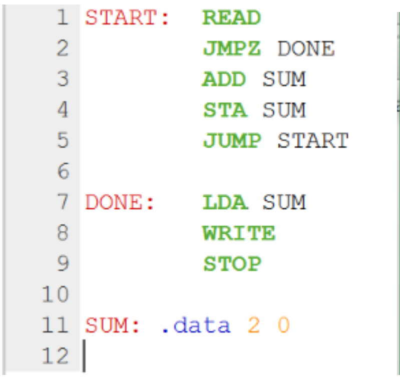


```
Instructions    Format  Implementation
JMPN            Execute sequence
JUMP            Input
LDA             End
ADD
STA
STOP
WRITE
READ
```

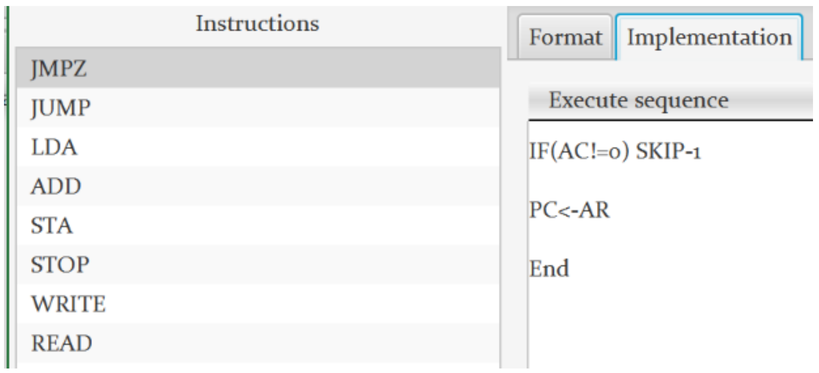


```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 5
Enter Inputs, the first of which must be an Integer: 6
Enter Inputs, the first of which must be an Integer: 0
Output:  11
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```

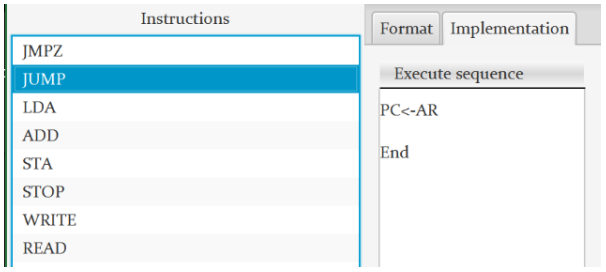
Q12. Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum



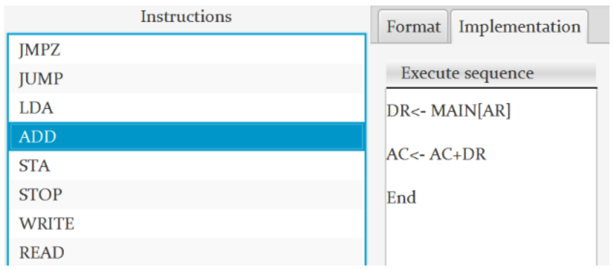
```
START:    READ
          JMPZ  DONE
          ADD  SUM
          STA  SUM
          JUMP  START

DONE:     LDA  SUM
          WRITE
          STOP

SUM: .data  2  0

```



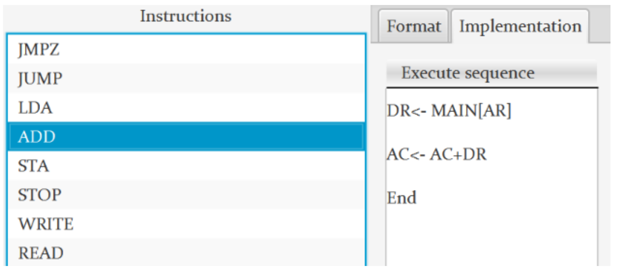
```
IF(AC!=o) SKIP-1

PC<-AR

End
```



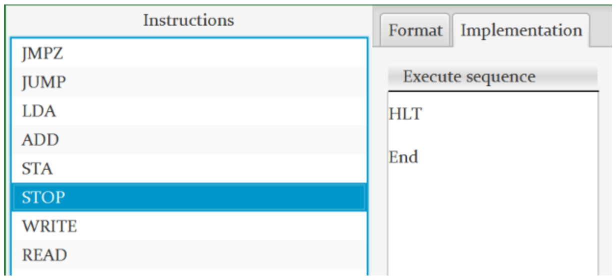
```
PC<-AR

End
```



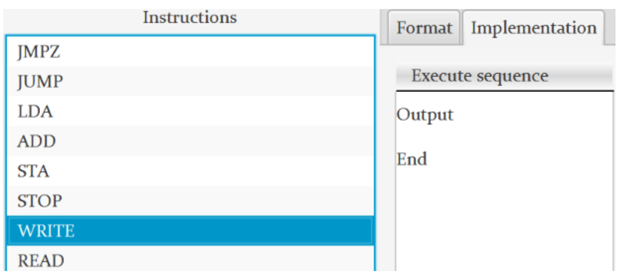
```
DR<- MAIN[AR]

AC<- AC+DR

End
```



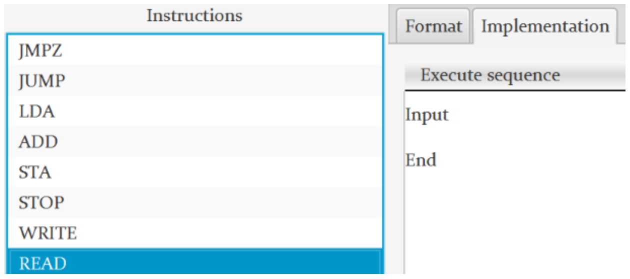
```
DR<- MAIN[AR]

AC<- AC+DR

End
```



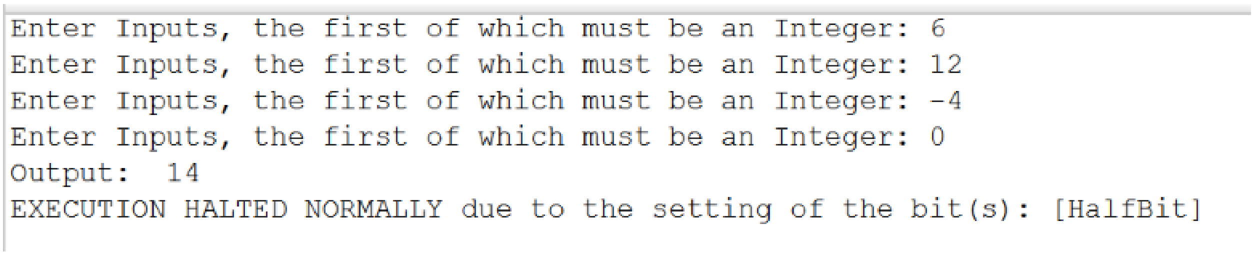
```
HLT

End
```



```
Output

End
```



```
Input

End
```



```
Enter Inputs, the first of which must be an Integer: 6
Enter Inputs, the first of which must be an Integer: 12
Enter Inputs, the first of which must be an Integer: -4
Enter Inputs, the first of which must be an Integer: 0
Output:  14
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HalfBit]
```