Gajendra Processor

Lab Report

submitted by

P . Yashwanth Sai (CS22B002), V . Yaswanth Sai (CS22B043)

Under the supervision
of

Dr.Ayon Chakraborty

Computer Science and Engineering

Indian Institute of Technology Madras

Jul - Nov 2023

Contents

List of Figures

1 General CPU Register

1.1 Diagram

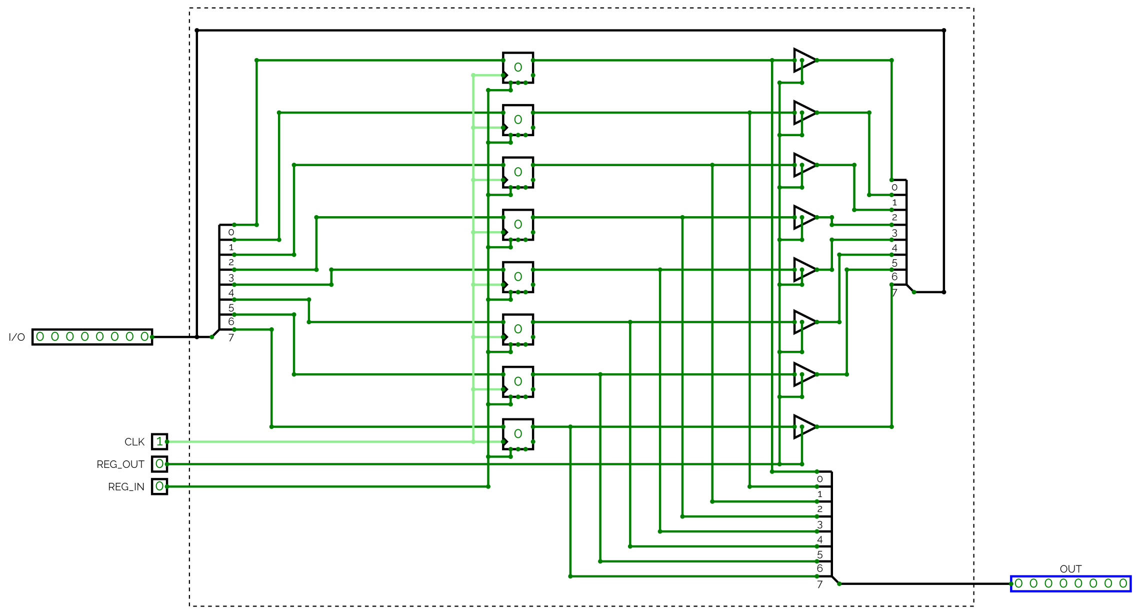


Figure 1: 8-bit Register

1.2 I/O Control

It takes 8-bit input from the common bus. Which is stored in it. The 8-bit value stored in it is given as output. REG_in and REGout are used as the control inputs for controlling the input and output.

1.3 Working

If REG_in is 1 the input is read from the common bus and when REG_out is 1 the output is written to the common bus.

2 Program Counter

2.1 Diagram

The ROM consists of both the instructions and data for the program that has to be run , The program counter points to the address of the instruction in ROM , that has to be processed

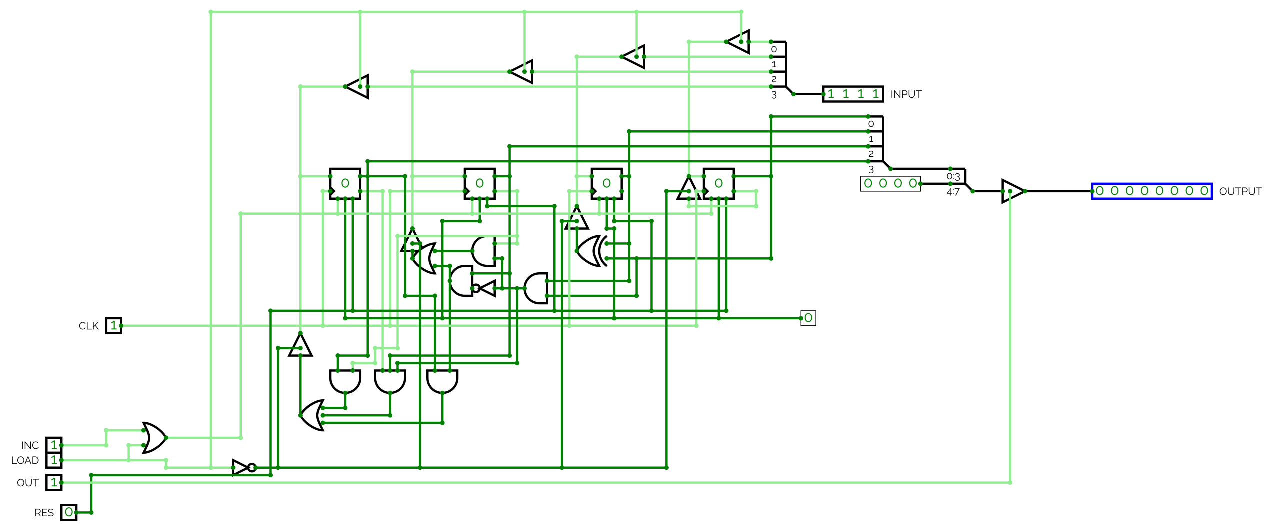


Figure 2: Program Counter

next . At the starting of the program the program counter is reset to 0000 , It then sends the address that is stored in it (i.e 0000 in this case) to the Memory Address Register , The program counter is then incremented by 1 (i.e to 0001 in this case) , After the first instruction is executed then the program counter sends the current address stored in it (i.e 0001 in this case) , this cycle keeps on repeating until the program ends.

The following are the functionalities that are present in the Program Counter.

2.2 PC_INC

If PC _INC = 1 then the value of the address stored in the Program Counter is incremented by 1 in each positive clock triggering .

2.3 PC_OUT

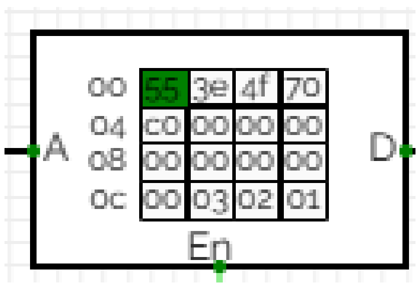
If PC_OUT = 1 then the Program Counter sends the address stored in it into the Least Significant Bits (0-3) of the **common bus** of the **Gajendra Processor** through **output** .

2.4 PC _LOAD

If PC_LOAD = 1 then the Program Counter takes the Least Significant Bits (0-3) of the **common bus** into it through the **input** and stores it as the next instruction .

3 MEMORY

3.1 Diagram



00	55	3e	4f	70
04	c0	00	00	00
08	00	00	00	00
0c	00	03	02	01

Figure 3: Memory

3.2 Diagram

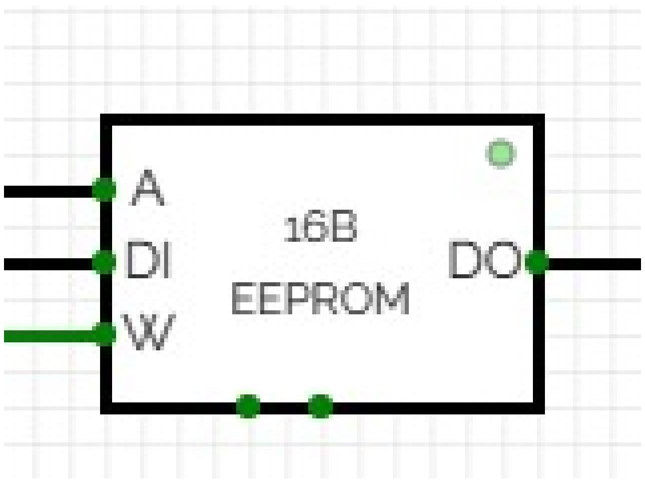


Figure 4: EEPROM

Memory is a ROM which contains both instruction and data required for a program. The following are the functionalities that are present in the Memory. We are using EEPROM here for STA instruction .

3.3 MEM_OUT

If MEM_OUT = 1 then the memory releases a 8-bit output into the common bus.

4 MEMORY ADDRESS REGISTER

During a computer run , the 4-bit address in the Programming Counter is latched into the MAR .Then the MAR applies this 4-bit address to the Memory where a read operation is performed. The following are the functionalities that are present in the MAR .

4.1 MAR _IN

If MAR _IN = 1 , then the MAR takes the 4-bit address (Least Significant in common bus) from the Program Counter through common bus.

5 INSTRUCTION REGISTER

5.1 Diagram

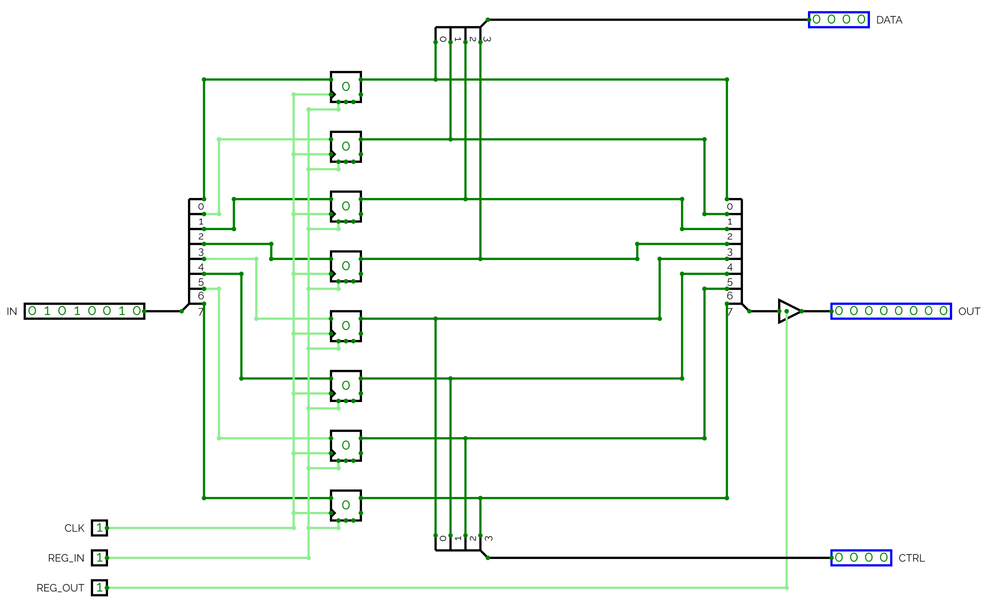


Figure 5: Instruction Register

To fetch an instruction from the memory the computer does a memory read operation . This places the contents of the addressed memory location on the common bus. The instruction register loads this content into it on the next positive clock edge .
The contents of the instruction register are split into two nibbles of 4-bits each (least 4 significant bits , most 4-significant bits) , The most 4-significant bits are sent into **Controller** and it will also send the 8-bit content into the **Instruction decoder** .
The following are the functionalities that are present in the Instruction register .

5.2 INS_REG _IN

If INS_REG _IN = 1 , then the INS register takes the 8-bit content as input from the common bus .

5.3 INS_REG _OUT

If INS_REG _OUT = 1 , then INS register loads its 8-bit content into Instruction decoder which further enter into the common bus .

5.4 CTRL

The Most Significant 4-bits(4-7) automatically goes to the **Controller** and **INS Decoder** as instruction from INS Register.

6 INSTRUCTION DECODER

6.1 Diagram

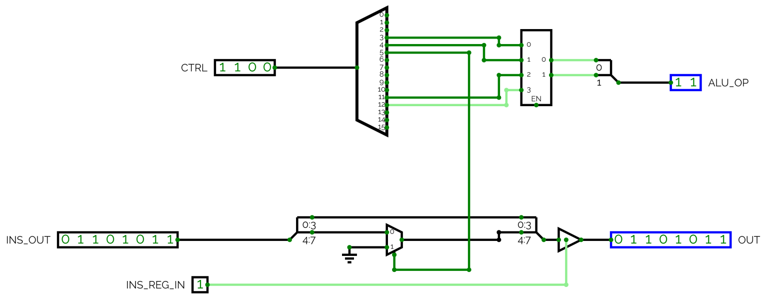


Figure 6: Instruction Decoder

It helps the ALU by giving the instruction whether to add or subtract , by using the most significant 4-bits (i.e CTRL) . It also helps us with the instruction **LDI** by making the 4 - most significant bits into 0000 , so that the remaining 4-bits gets stored in our **accumulator (i.e register A).**
The following are the functionalities that are present in the Instruction Decoder .

6.2 INS _DEC _OUT

If INS _DEC _OUT = 1 , then it takes the 8-bit content from OUT of INS register .

6.3 ALU_OUT

The INS_DEC generates a 2-bit input for the **ALU**.
It generates the following outputs for the corresponding instructions .

$$00 \implies Addition$$

$$01 \implies Subtraction$$

$$10 \implies LShift$$

$$11 \implies RShift$$

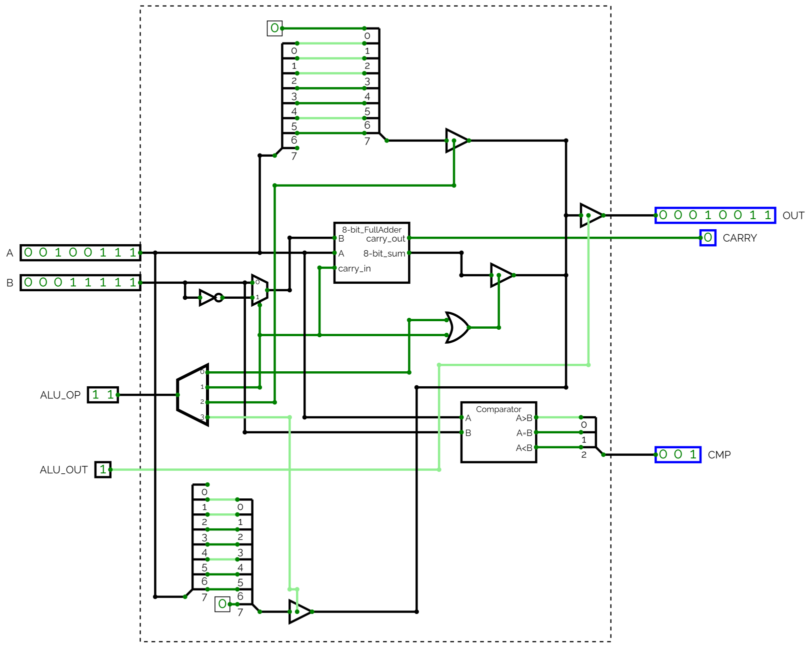


Figure 7: Gajendra ALU

7 Arithmetic and Logic Unit(ALU)

The ALU performs the arithmetic operations that are executed by the computer. The operations that are supported are Addition(ADD) , Subtraction(SUB) and SHIFT . It stores the final value in it and it also returns the final 8-bit into Accumulator or REG _A .

7.1 ADD

The ADD operation is used to add two 8-bit numbers using a 8-bit full adder which adds without carry .

7.2 SUB

The SUB operation is used to subtract two 8-bit numbers using a 8-bit full adder with an input from the INS_DEC which triggers sub operation.

7.3 SHIFT

The LSHIFT operation performs a bitwise left shift on the value in register A whereas the RSHIFT operation performs a bitwise right shift on the value in register A.

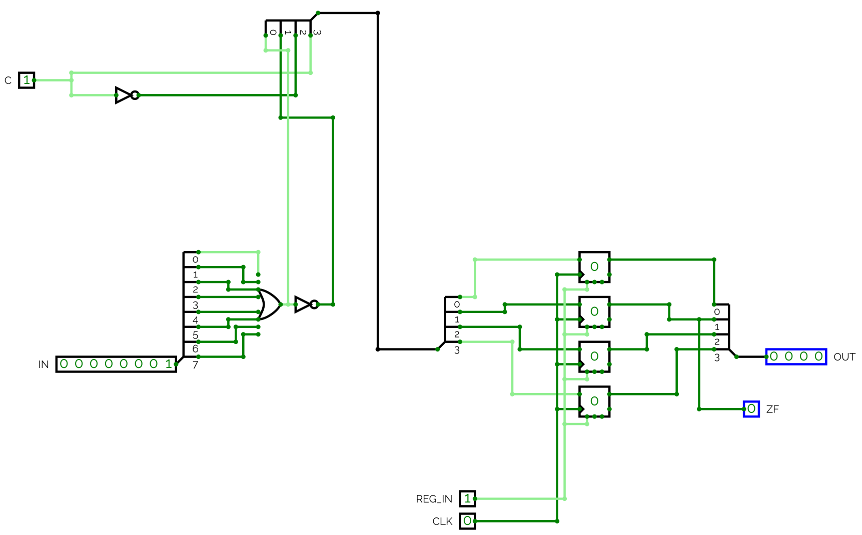


Figure 8: Status Register

8 Status Register

8.1 Diagram

8.2 I/O

It takes 2 inputs ALU _OUT and Carry from ALU and gives a 4-bit output in which the **the most significant bit is C(1 if carry = 1)** and the next bit gives **NC**$(\overline{C})$ and the next bit gives **Z(1 if(ALU _OUT = 1)** and the least significant gives **NZ** $\overline{Z}$.

8.3 Uses

It is used to implement JUMP and JUMP if zero conditions .

9 Control Processor

9.1 Diagram

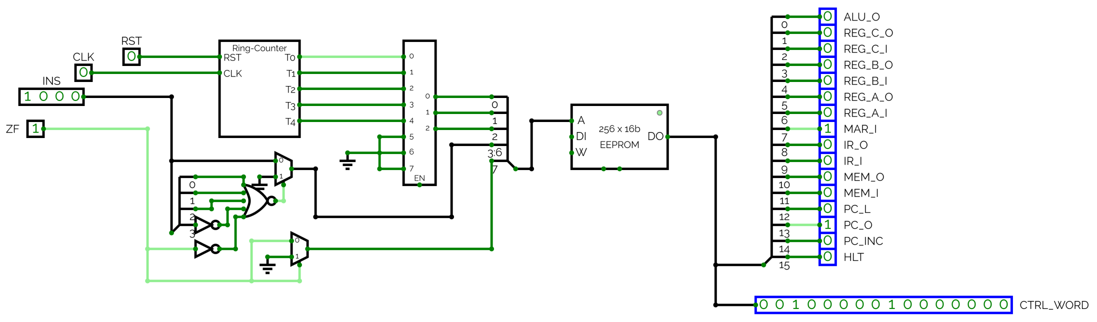


Figure 9: Control Processor

9.2 I/O

The control processor takes 4 bit instruction as input and we use a ring counter of length 5 indicating the 5 T-states. The EEPROM in the control processor takes 8-bit address in which the first bit is inherently zero and next 4 bits are the instruction and the last 3 bits are the T-state that is to be implemented. The EEPROM gives 16-bit output which is the control word which is also the output of the Control Processor.

9.3 Working

This control word controls all the other elements like REG_in,REG_out,PC_load, PC_out, PC_inc, flag, MEM_in, MEM_out, MAR_in, INS_out, INS_out, ALU_out The control processor is the most important component of our processor as it controls all the other components . When Ins is 1000 (JNZ) and ZF = 1, this instruction changes to NOP (0000) and the MSB of address entering into EEPROM will also be 0 (i.e the ZF will enter A as 0). In this way other half of the EEPROM need not be wasted and optimized.

9.4 Ring Counter

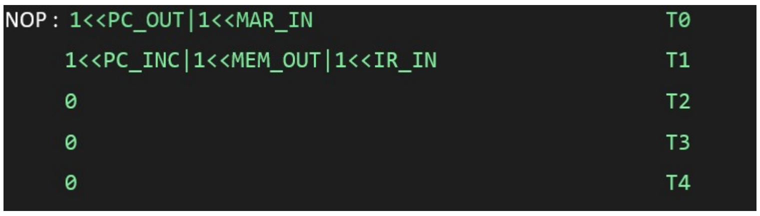
Every instruction has some micro instructions called T-states that are to be executed in order one at a time , these control words are generated by accessing the memory locations in the EEPROM , the T-states are governed by the ring counter .

9.5 EEPROM Programming

The EEPROM is loaded in such a way that every instruction has 8 memory locations in the EEPROM for its micro instructions. The each memory locations contains the control word of that particular micro instruction.

10 Microinstructions and Controller Logic Design

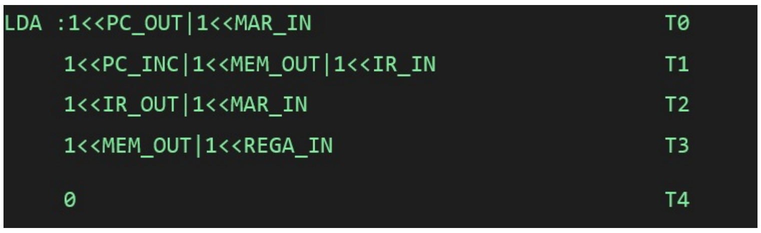
10.1 NOP :



```
NOP: 1<<PC_OUT|1<<MAR_IN                         T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                T1
    0                                            T2
    0                                            T3
    0                                            T4
```

Figure 10: Micro Instruction for NOP

10.2 LDA :

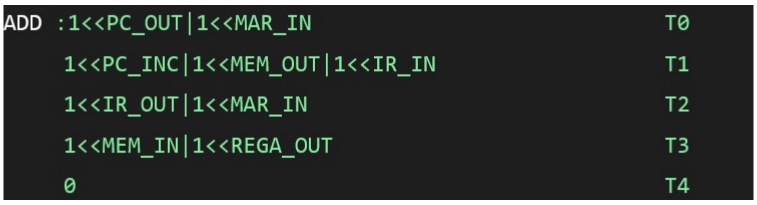


```
LDA :1<<PC_OUT|1<<MAR_IN                         T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                T1
    1<<IR_OUT|1<<MAR_IN                          T2
    1<<MEM_OUT|1<<REGA_IN                        T3

    0                                            T4
```

Figure 11: Micro instruction for LDA

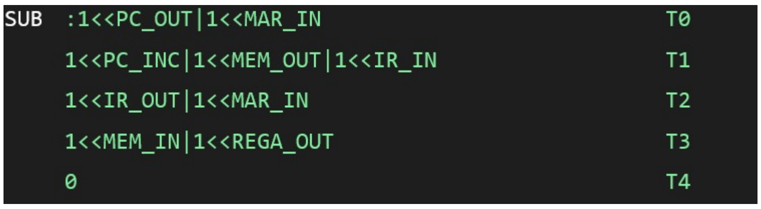
10.3 ADD :



```
ADD :1<<PC_OUT|1<<MAR_IN                         T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                T1
    1<<IR_OUT|1<<MAR_IN                          T2
    1<<MEM_IN|1<<REGA_OUT                        T3
    0                                            T4
```

Figure 12: Micro instruction for ADD

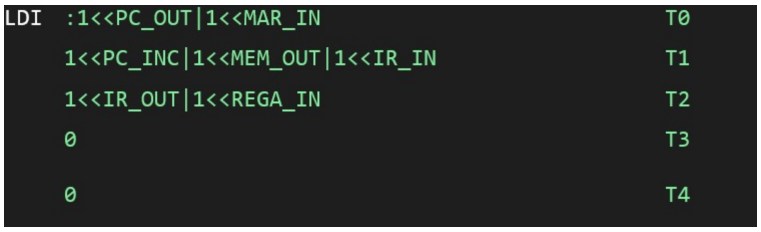
10.4 SUB :



```
SUB  :1<<PC_OUT|1<<MAR_IN                       T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN              T1
     1<<IR_OUT|1<<MAR_IN                        T2
     1<<MEM_IN|1<<REGA_OUT                      T3
     0                                          T4
```

Figure 13: Micro instruction for SUB

10.5 LDI :

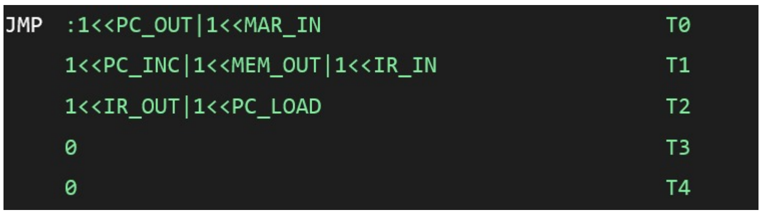


```
LDI  :1<<PC_OUT|1<<MAR_IN                       T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN              T1
     1<<IR_OUT|1<<REGA_IN                       T2
     0                                          T3

     0                                          T4
```

Figure 14: Micro instruction for LDI

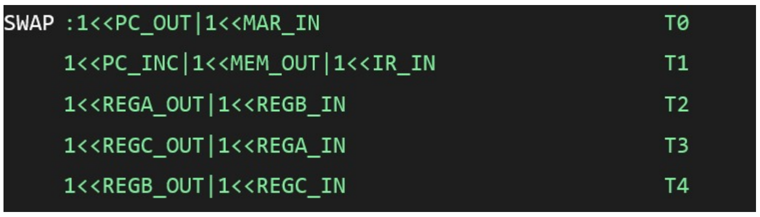
10.6 JMP :



```
JMP  :1<<PC_OUT|1<<MAR_IN                       T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN              T1
     1<<IR_OUT|1<<PC_LOAD                       T2
     0                                          T3
     0                                          T4
```

Figure 15: Micro instruction for JMP

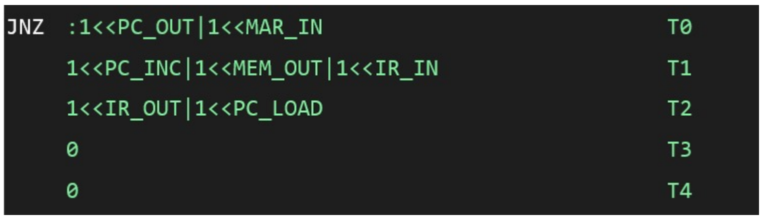
10.7 SWAP :



```
SWAP :1<<PC_OUT|1<<MAR_IN                                   T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                          T1
     1<<REGA_OUT|1<<REGB_IN                                 T2
     1<<REGC_OUT|1<<REGA_IN                                 T3
     1<<REGB_OUT|1<<REGC_IN                                 T4
```

Figure 16: Micro instruction for SWAP

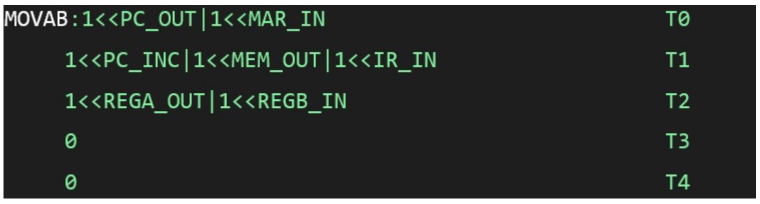
10.8 JNZ :



```
JNZ  :1<<PC_OUT|1<<MAR_IN                                   T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                          T1
     1<<IR_OUT|1<<PC_LOAD                                   T2
     0                                                      T3
     0                                                      T4
```

Figure 17: Micro instruction for JNZ

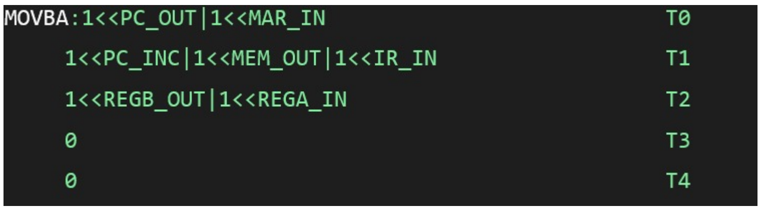
10.9 MOVAB :



```
MOVAB:1<<PC_OUT|1<<MAR_IN                                   T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                          T1
     1<<REGA_OUT|1<<REGB_IN                                 T2
     0                                                      T3
     0                                                      T4
```

Figure 18: Micro instruction for MOVAB

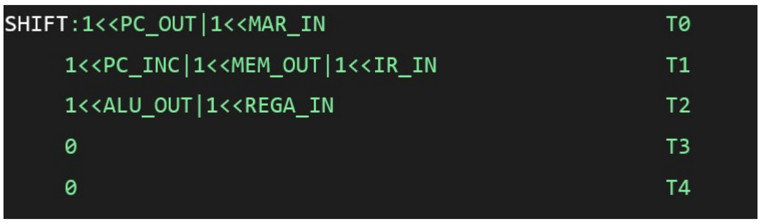
10.10 MOVBA :



```
MOVBA:1<<PC_OUT|1<<MAR_IN                              T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                      T1
    1<<REGB_OUT|1<<REGA_IN                             T2
    0                                                  T3
    0                                                  T4
```

Figure 19: Micro instruction for MOVBA

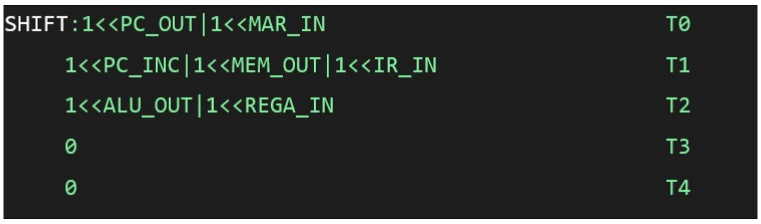
10.11 LSHIFT :



```
SHIFT:1<<PC_OUT|1<<MAR_IN                              T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                      T1
    1<<ALU_OUT|1<<REGA_IN                              T2
    0                                                  T3
    0                                                  T4
```

Figure 20: Micro instruction for LSHIFT

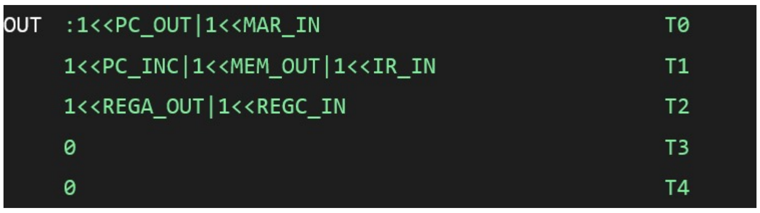
10.12 RSHIFT :



```
SHIFT:1<<PC_OUT|1<<MAR_IN                              T0
    1<<PC_INC|1<<MEM_OUT|1<<IR_IN                      T1
    1<<ALU_OUT|1<<REGA_IN                              T2
    0                                                  T3
    0                                                  T4
```

Figure 21: Micro instruction for RSHIFT

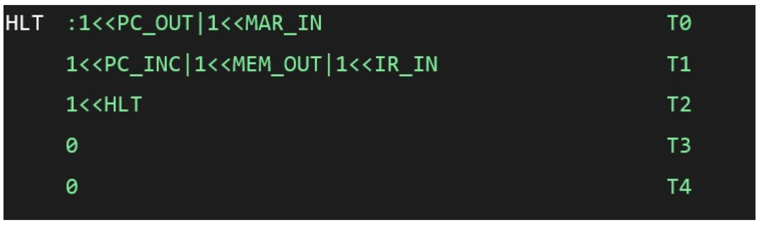
10.13 OUT :



```
OUT  :1<<PC_OUT|1<<MAR_IN                          T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                 T1
     1<<REGA_OUT|1<<REGC_IN                        T2
     0                                             T3
     0                                             T4
```

Figure 22: Micro instruction for OUT

10.14 HLT :

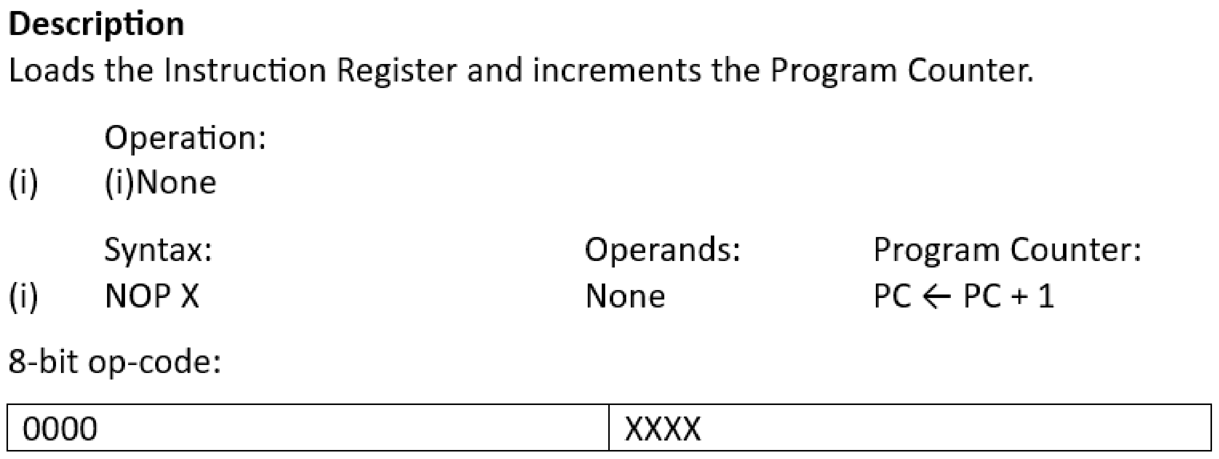


```
HLT  :1<<PC_OUT|1<<MAR_IN                          T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                 T1
     1<<HLT                                        T2
     0                                             T3
     0                                             T4
```

Figure 23: Micro instruction for HLT

11 Instruction Set

11.1 NOP :



Description

Loads the Instruction Register and increments the Program Counter.

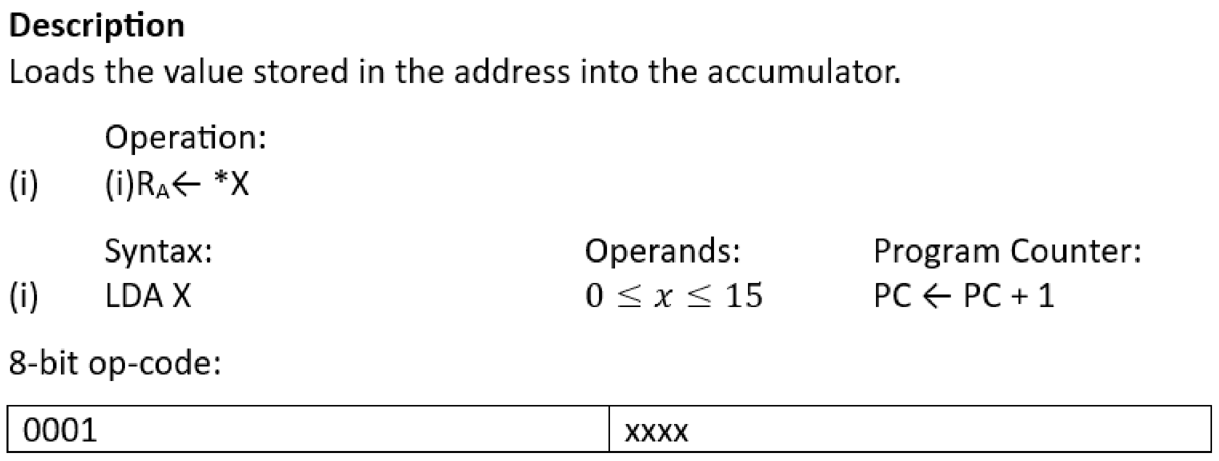
	Operation:		
(i)	(i)None		
	Syntax:	Operands:	Program Counter:
(i)	NOP X	None	PC ← PC + 1

8-bit op-code:

0000	XXXX

Figure 24: Instruction Set for NOP

11.2 LDA :



Description

Loads the value stored in the address into the accumulator.

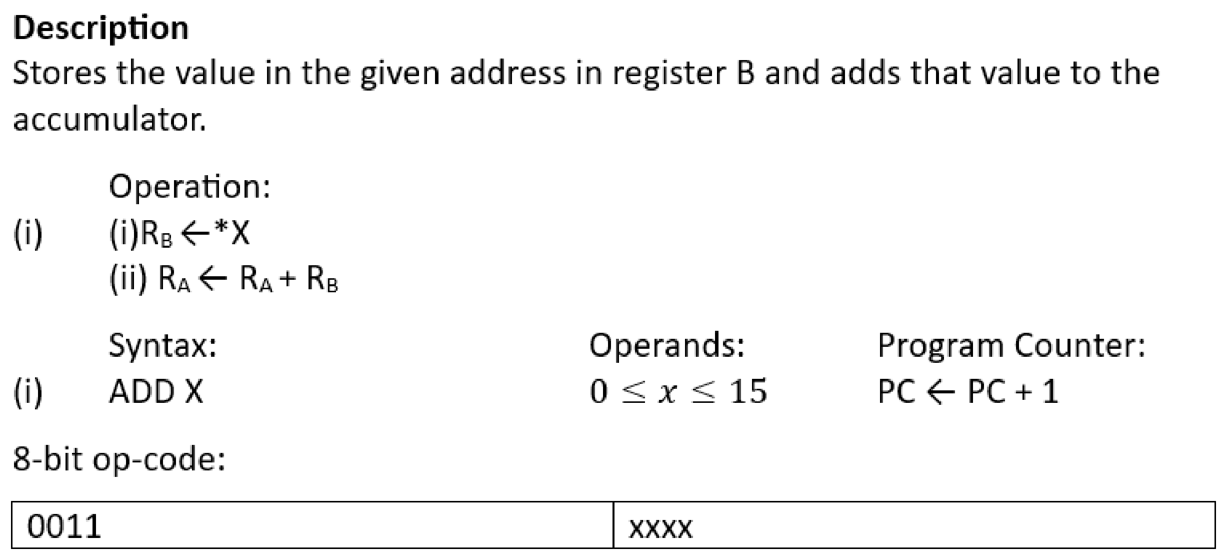
	Operation:		
(i)	(i)R_A← *X		
	Syntax:	Operands:	Program Counter:
(i)	LDA X	$0 \leq x \leq 15$	PC ← PC + 1

8-bit op-code:

0001	xxxx

Figure 25: Instruction Set for LDA

11.3 ADD :



Description
Stores the value in the given address in register B and adds that value to the accumulator.

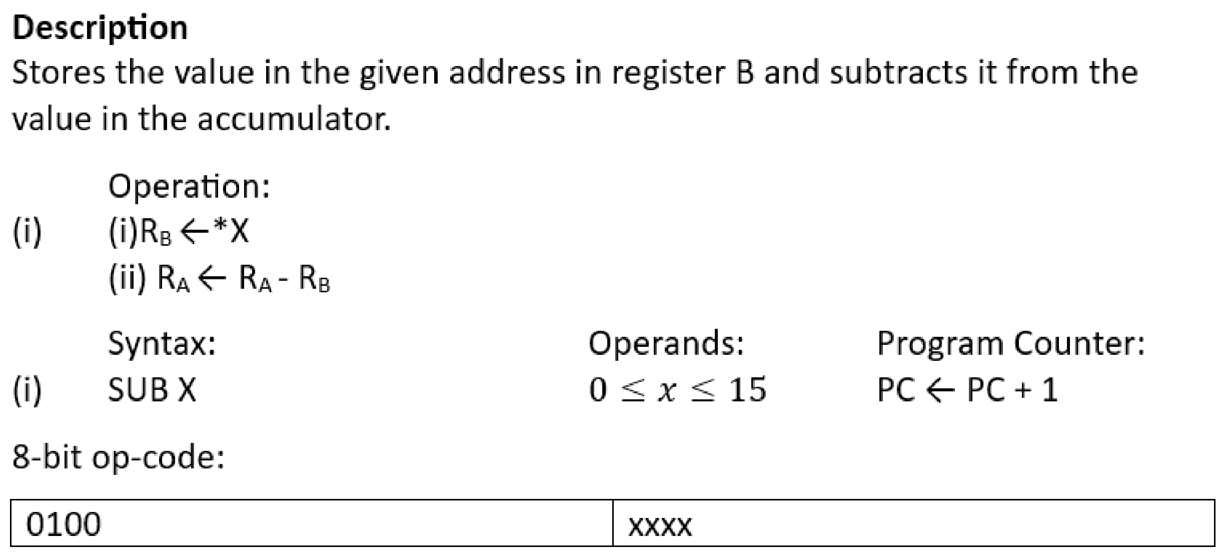
	Operation:		
(i)	(i)$R_B \leftarrow *X$		
	(ii) $R_A \leftarrow R_A + R_B$		
	Syntax:	Operands:	Program Counter:
(i)	ADD X	$0 \leq x \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0011	xxxx

Figure 26: Instruction Set for ADD

11.4 SUB :



Description
Stores the value in the given address in register B and subtracts it from the value in the accumulator.

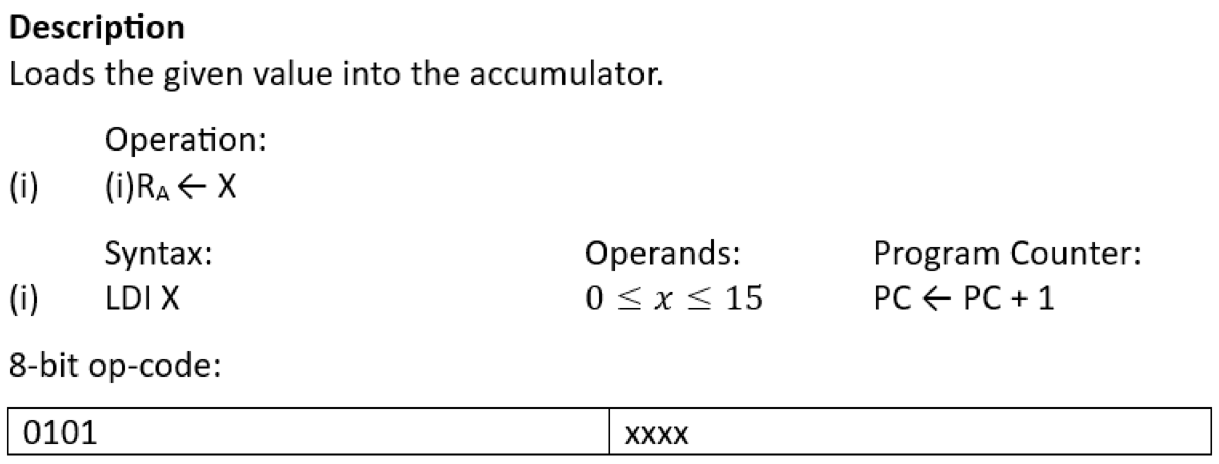
	Operation:		
(i)	(i)$R_B \leftarrow *X$		
	(ii) $R_A \leftarrow R_A - R_B$		
	Syntax:	Operands:	Program Counter:
(i)	SUB X	$0 \leq x \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0100	xxxx

Figure 27: Instruction Set for SUB

11.5 LDI :



Description

Loads the given value into the accumulator.

	Operation:
(i)	(i)$R_A \leftarrow X$

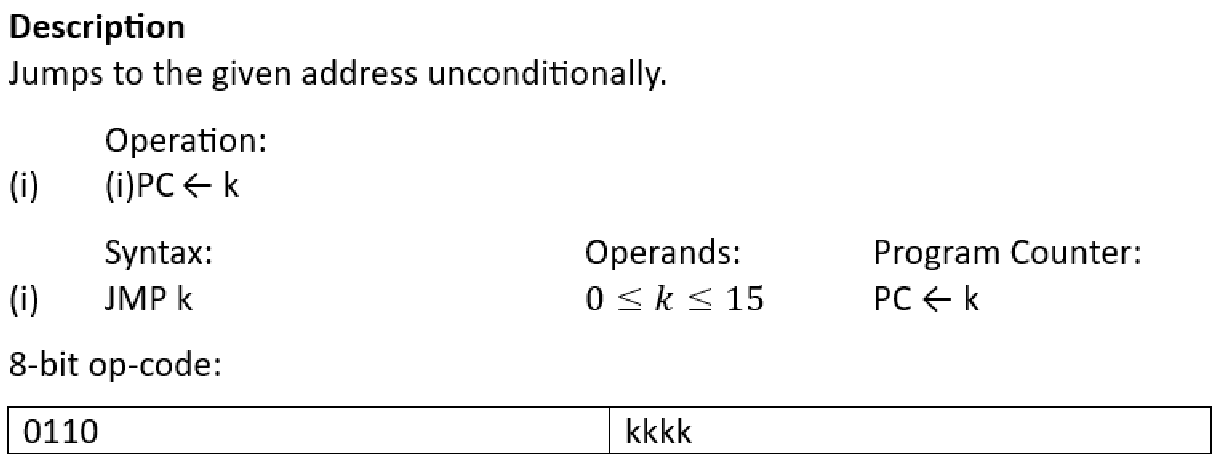
	Syntax:	Operands:	Program Counter:
(i)	LDI X	$0 \leq x \leq 15$	PC ← PC + 1

8-bit op-code:

0101	xxxx

Figure 28: Instruction Set for LDI

11.6 JMP :



Description

Jumps to the given address unconditionally.

	Operation:
(i)	(i)PC ← k

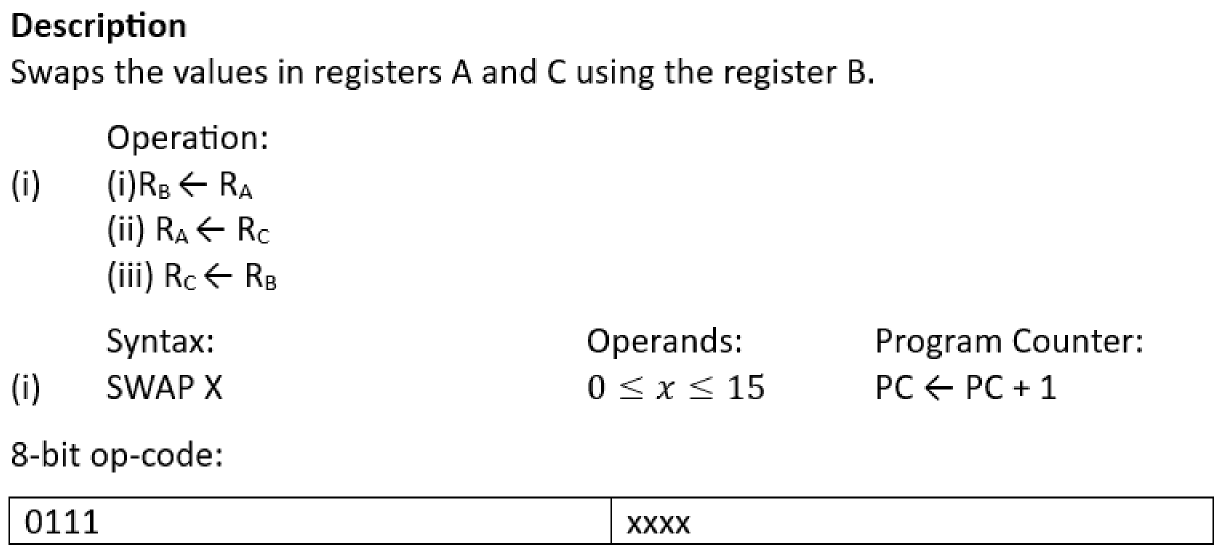
	Syntax:	Operands:	Program Counter:
(i)	JMP k	$0 \leq k \leq 15$	PC ← k

8-bit op-code:

0110	kkkk

Figure 29: Instruction Set for JMP

11.7 SWAP :



Description

Swaps the values in registers A and C using the register B.

	Operation:
(i)	(i)$R_B \leftarrow R_A$ (ii) $R_A \leftarrow R_C$ (iii) $R_C \leftarrow R_B$

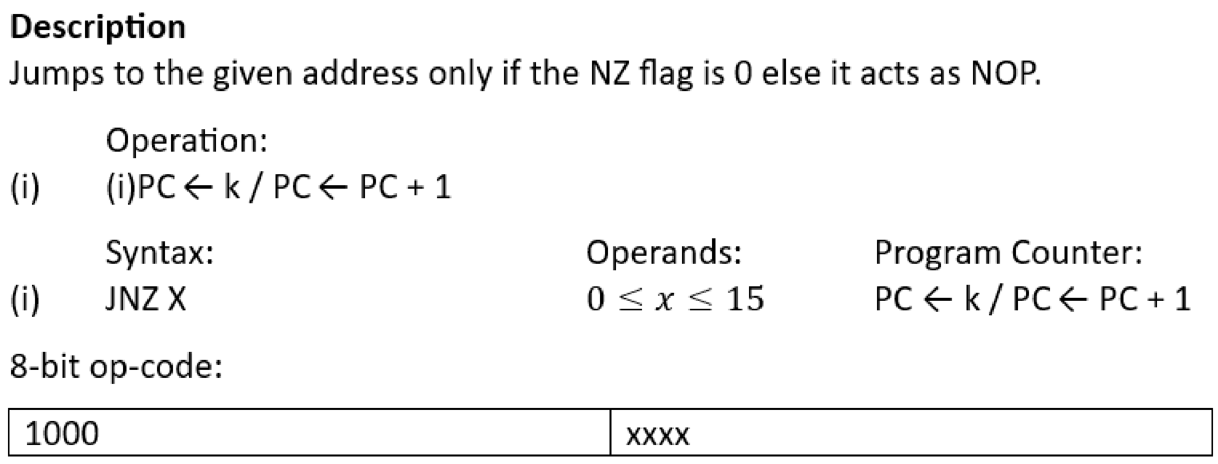
	Syntax:	Operands:	Program Counter:
(i)	SWAP X	$0 \le x \le 15$	PC ← PC + 1

8-bit op-code:

0111	xxxx

Figure 30: Instruction for SWAP

11.8 JNZ :



Description

Jumps to the given address only if the NZ flag is 0 else it acts as NOP.

	Operation:
(i)	(i)PC ← k / PC ← PC + 1

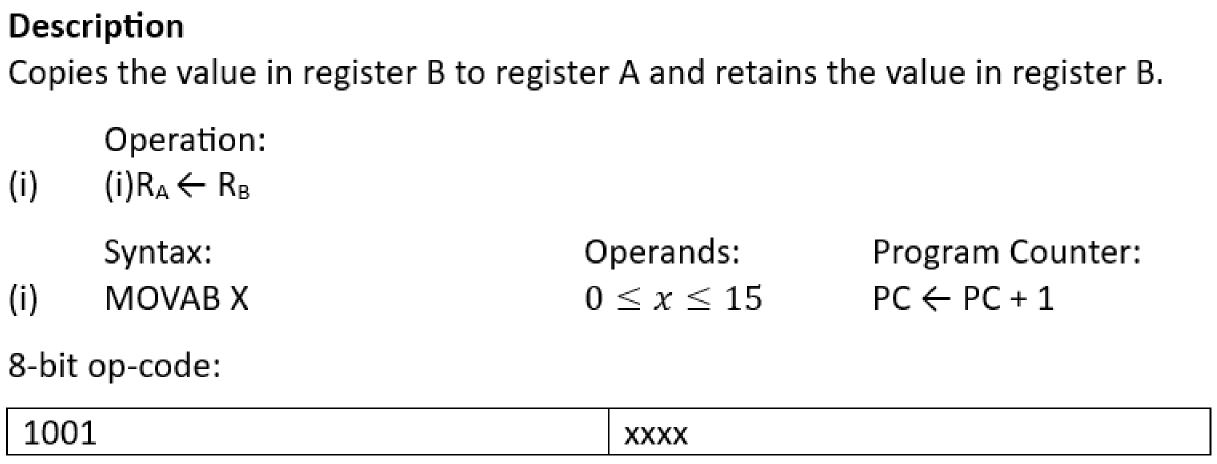
	Syntax:	Operands:	Program Counter:
(i)	JNZ X	$0 \le x \le 15$	PC ← k / PC ← PC + 1

8-bit op-code:

1000	xxxx

Figure 31: Instruction Set for JNZ

11.9 MOVAB :



Description

Copies the value in register B to register A and retains the value in register B.

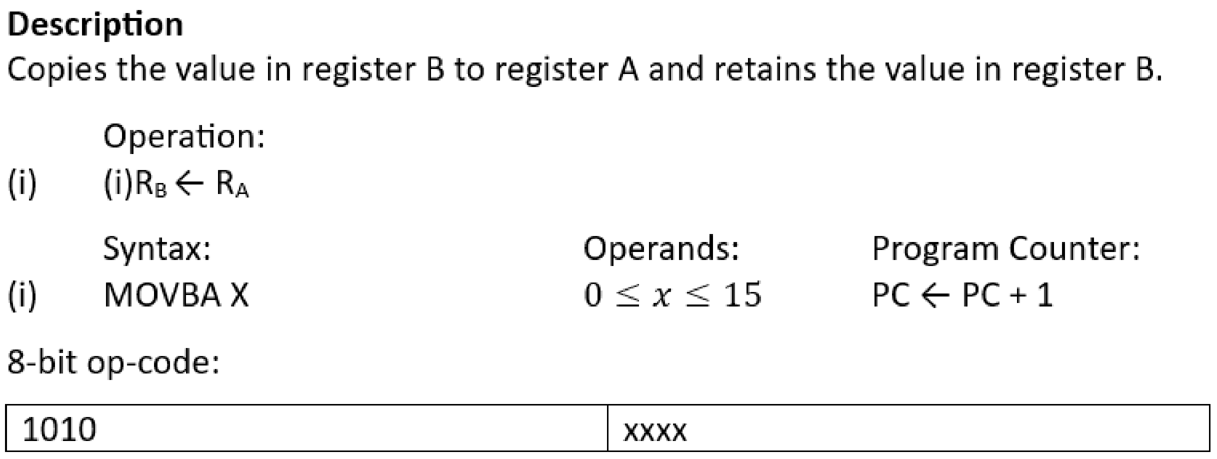
	Operation:		
(i)	(i)$R_A \leftarrow R_B$		
	Syntax:	Operands:	Program Counter:
(i)	MOVAB X	$0 \leq x \leq 15$	PC $\leftarrow$ PC + 1

8-bit op-code:

1001	xxxx

Figure 32: Instruction Set for MOVAB

11.10 MOVBA :



Description

Copies the value in register B to register A and retains the value in register B.

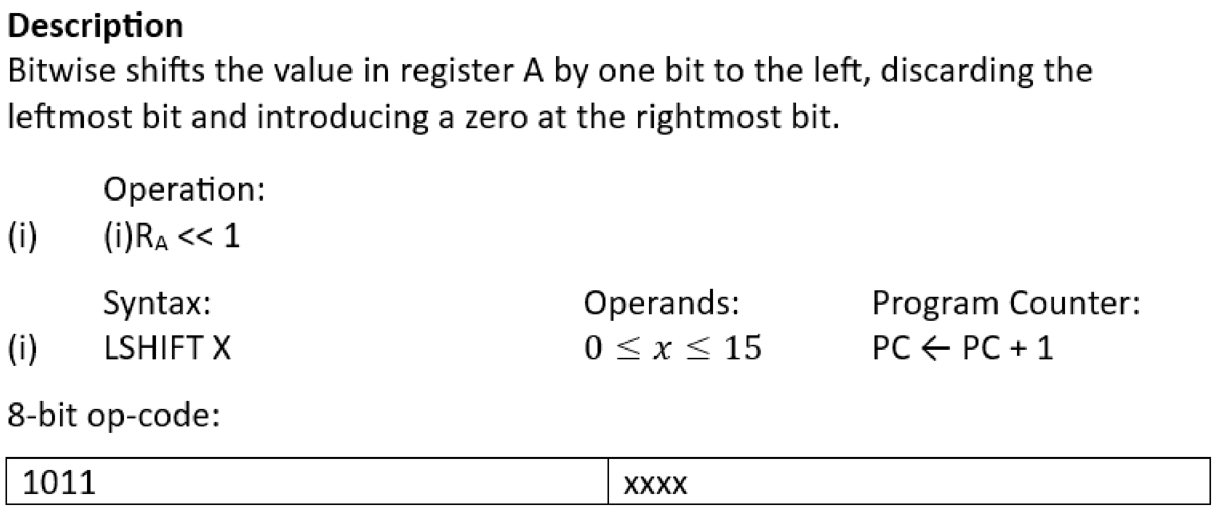
	Operation:		
(i)	(i)$R_B \leftarrow R_A$		
	Syntax:	Operands:	Program Counter:
(i)	MOVBA X	$0 \leq x \leq 15$	PC $\leftarrow$ PC + 1

8-bit op-code:

1010	xxxx

Figure 33: Instruction Set for MOVBA

11.11 LSHIFT :



Description
Bitwise shifts the value in register A by one bit to the left, discarding the leftmost bit and introducing a zero at the rightmost bit.

	Operation:
(i)	(i)R_A << 1

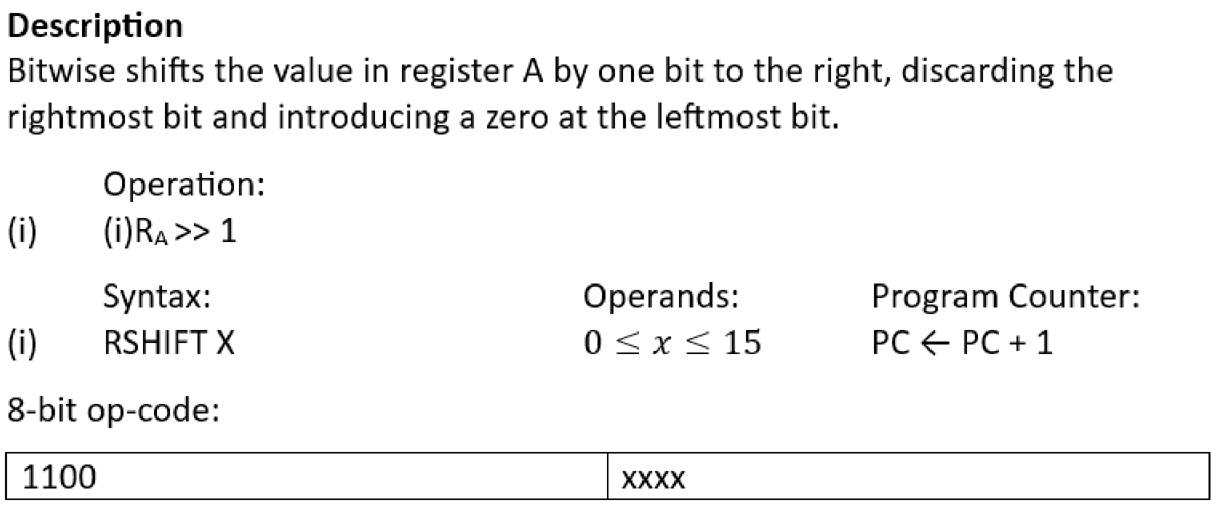
	Syntax:	Operands:	Program Counter:
(i)	LSHIFT X	$0 \leq x \leq 15$	PC ← PC + 1

8-bit op-code:

1011	xxxx

Figure 34: Instruction Set for LSHIFT

11.12 RSHIFT :



Description
Bitwise shifts the value in register A by one bit to the right, discarding the rightmost bit and introducing a zero at the leftmost bit.

	Operation:
(i)	(i)R_A >> 1

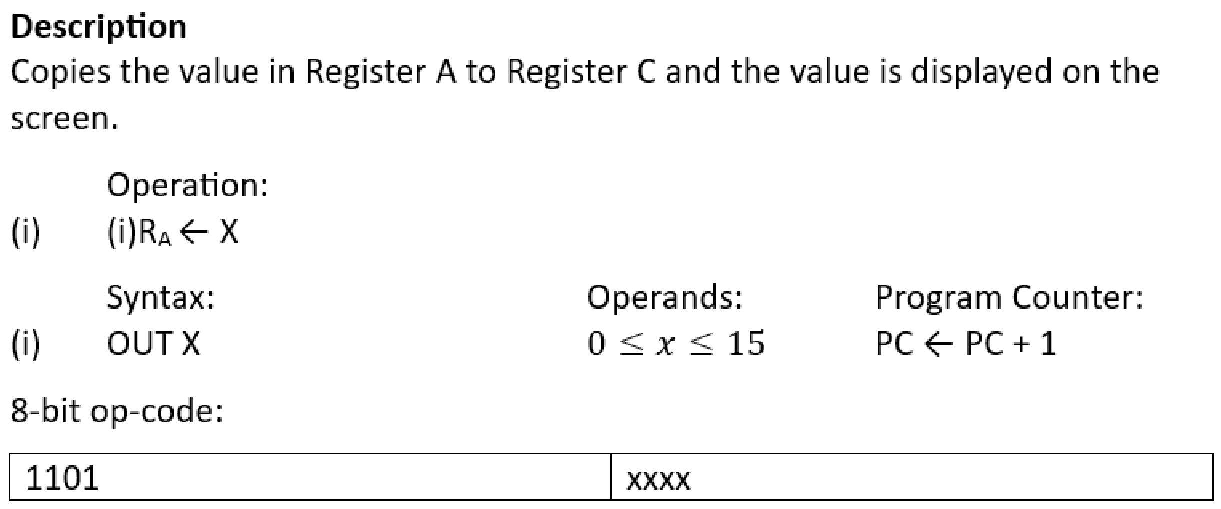
	Syntax:	Operands:	Program Counter:
(i)	RSHIFT X	$0 \leq x \leq 15$	PC ← PC + 1

8-bit op-code:

1100	xxxx

Figure 35: Instruction Set for RSHIFT

11.13 OUT :



Description

Copies the value in Register A to Register C and the value is displayed on the screen.

Operation:

(i) (i)$R_A \leftarrow X$

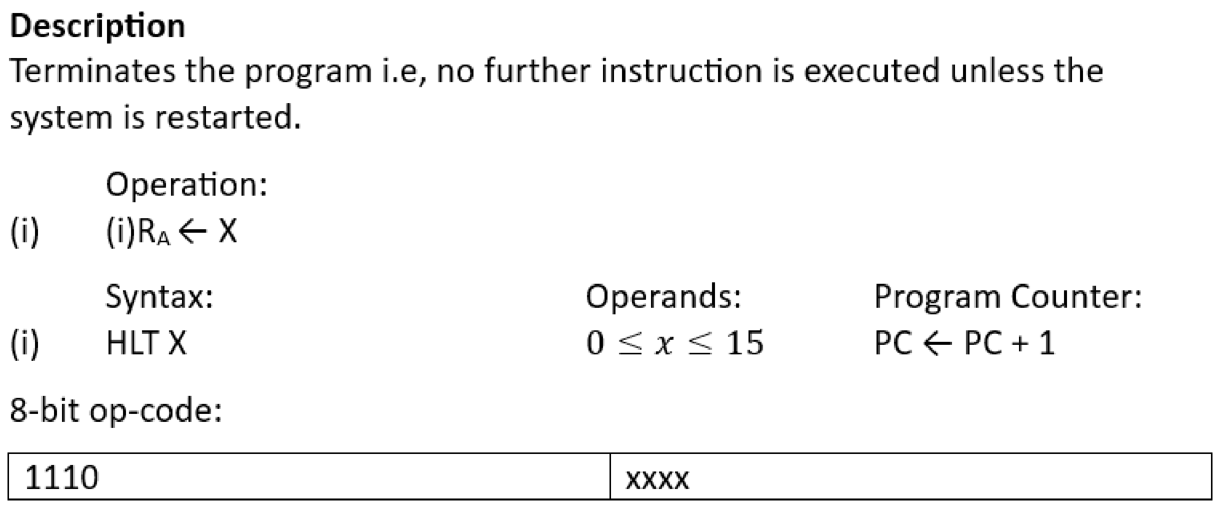
	Syntax:	Operands:	Program Counter:
(i)	OUT X	$0 \le x \le 15$	PC ← PC + 1

8-bit op-code:

1101	xxxx

Figure 36: Instruction Set for OUT

11.14 HLT :



Description

Terminates the program i.e, no further instruction is executed unless the system is restarted.

Operation:

(i) (i)$R_A \leftarrow X$

	Syntax:	Operands:	Program Counter:
(i)	HLT X	$0 \le x \le 15$	PC ← PC + 1

8-bit op-code:

1110	xxxx

Figure 37: Instruction Set for HLT

12 Sample programs

12.1 For Addition and Subtraction

Load for EEPROM for the Addition and Subtraction Program : (0x1c,0x3f,0x4d,0xd0,0xe0,0,0,0,0,0,0,0,0x0

LOAD c
ADD f
SUB d
OUT
HALT

0xd : 2
0xc : 3
0xf : 5

12.2 For L-shift

Load for EEPROM for the L-shift Program :(0x56,0xb0,0xd0,0xe0,0,0,0,0,0,0,0,0,0,0,0,0)

LDI 6
LSHIFT
OUT
HALT

12.3 For Multiplication

This example is for 3*5 .(CUSTOM values can be given in the mentioned places).

Load for EEPROM for the multiplication of 3 and 5 : (0x1a,0x3c,0x2e,0x1d,0x4b,0x2d,0x1e,0x81,0xd0,0
0x03,0x05 (i.e c, d can be changed)

LDA a
ADD c
STA e
LDA d
SUB b
STA d
LDA e
JNZ 1 (To 2nd step(i.e ADD c))
OUT
HALT

0xA : 0
0xb : 1
0xc : 3
0xd : updated value of the cycle ¡= 5 .
0xe : value of final value after n cycles in loop (i.e 3*n) is stored .