Instruction Set

1. MRI Instruction

Format of 12-bit MRI Instruction:

I (1-Bit) (Direct or Indirect)	3-Bit MRI Opcode	8-Bit Address

Instruction Table:

MRI Instruction	Opcode	
	Direct (I=0)	Indirect (I=1)
LDA XX	0XX	8XX
STA XX	1XX	9XX
ADM XX	2XX	AXX
CALL XX	3XX	BXX
JMP XX	4XX	CXX
JC XX	5XX	DXX
JZ XX	6XX	EXX

‘XX’ - 8-Bit Address

2. non-MRI Instruction

Format of 12-bit non-MRI Instruction:

I (1-Bit) (Register or I/O based)	111	4-Bit Non-MRI Opcode	4-Bit Data/Don't care

Instruction Table:

Type	Non-MRI Instruction	Opcode
Arithmetic and Logical	ADD B	70_
	SUB B	71_
	CLA	72_
	CMA	73_
Register based Data Transfer	MOV A,B	74_
	MOV B,A	75_
Stack related	PUSH A	76_
	POP	77_
	RET	78_
* Immediate Data Transfer	MVI A,data	79X
	MVI F,data	7AX
Shift and Rotate related	LSC A,data	7BX
	RSC A,data	7CX
	RRX A,data	7DX
	ASR A,data	7EX
Halt	HLT	7F_
** I/O related (I/O=1)	IN S	F0X
	OUT S	F1X
	SKI S	F2X
	SKO S	F3X

'X' - 4-Bit Data
' _ ' - can be any value from 0 to F

MICRO INSTRUCTIONS

For MRI Instructions

LDA

Instruction to load register A with the value at the given address(direct)

Direct : instruction_format **(0XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L) , MAR(L,E)          // MAR <- MDR(7-0), IR <- MDR(8-11)
T3:  Idle
T4:  Idle
T5:  A(L) , RAM(E), Reset              // A <- M[MAR]
```

Indirect: instruction_format **(8XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:   MDR(E) , IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <- MDR(11-8)
T3:  MDR(L) , RAM(E)                   // MDR <- M[MAR]
T4:  MAR(L,E), MDR(E)                  // MAR <- MDR(7-0)
T5:  A(L) , RAM(E), Reset              // A <- M[MAR]
```

STA

Instruction to store the value from register A to the given memory location.

Direct : instruction_format **(1XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L) , MAR(L,E)          // MAR <- MDR(7-0), IR <- MDR(8-11)
```

T3: Idle
T4: Idle
T5: RAM(L) , A(E) , Reset   // M[MAR] <- A

Indirect: instruction_format **(9XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:   MDR(E) ,IR(L) , MAR(L,E)   // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E)   // MDR <- M[MAR]
T4: MAR(L,E), MDR(E)   // MAR <- MDR(7-0)
T5: RAM(L) , A(E) , Reset   // M[MAR] <- A

ADM

Instruction to add AC to memory operand.

Direct **:** instruction_format **(2XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L)   // MAR <- MDR(7-0), IR <- MDR(8-11)
T3: Idle
T4: Idle
T5: MDR(L) , RAM(E)   // MDR <- M[MAR]
T6: A(E,L) , MDR(E) , ALU(000) , Reset   // A <- A + MDR

Indirect: instruction_format **(AXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:   MDR(E) ,IR(L) , MAR(L,E)   // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E)   // MDR <- M[MAR]
T4: MAR(L,E),MDR(E)   // MAR <- MDR(7-0)
T5: MDR(L) , RAM(E)   // MDR <- M[MAR]
T6: A(E,L) , MDR(E) , ALU(000) , Reset   // A <- A + MDR

CALL

The CALL microoperation stores the return address in the stack and loads the PC with the given address.

Direct **:** instruction_format **(3XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E), IR(L), MAR(L,E)          // MAR <- MDR(7-0), IR <- MDR(11-8)
T3:  Idle
T4:  Idle
T5:  SP(D)
T6:  SP(E), MAR(E,L)                  // MAR <- SP
T7:  RAM(L), PC(E)                    // M[MAR] <- PC
T8:  PC(L), MDR(E), Reset             // PC <- MDR(7-0)
```

Indirect: instruction_format **(BXX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I), RAM(E), MDR(L)
T2:  MDR(E), IR(L), MAR(L,E)          // MAR <- MDR(7-0), IR <- MDR(11-8)
T3:  MDR(L), RAM(E)                   // MDR <- M[MAR]
T4:  Idle
T5:  SP(D)
T6:  SP(E), MAR(E,L)                  // MAR <- SP
T7:  RAM(L), PC(E)                    // M[MAR] <- PC
T8:  PC(L), MDR(E)                    // PC <- MDR(7-0)
```

JMP

The program sequence is transferred to the memory location specified by the particular level given in the operand.

Direct **:** instruction_format **(4XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
```

```
T2:  MDR(E) ,IR(L), MAR(L,E)                  // IR <- MDR(8-11)
T3:  Idle
T4:  Idle
T5:  PC(L) , MDR(E), Reset                    // PC <- MDR(7-0)
```

Indirect: instruction_format **(CXX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:   MDR(E) , IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <- MDR(11-8)
T3:  MDR(L) , RAM(E)                   // MDR <- M[MAR]
T4:  Idle
T5:  PC(L) , MDR(E), Reset             // PC <- MDR(7-0)
```

JC

The program sequence is transferred to a particular level or a 12-bit address if C=1 (or carry is 1)

Direct : instruction_format **(5XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L), MAR(L,E)           // IR <- MDR(8-11)
T3:  Idle
T4:  Idle
T5:  PC(L) , MDR(E), Reset             // if C=1, PC <- MDR(7-0)
                                          else Reset
```

Indirect: instruction_format **(DXX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:   MDR(E) , IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <-MDR(11-8)
T3:  MDR(L) , RAM(E)                   // MDR <- M[MAR]
T4:  Idle
T5:  PC(L) , MDR(E), Reset             // if C=1, PC <- MDR(7-0)
                                          else Reset
```

JZ

The program sequence is transferred to a particular level or a 12-bit address if Z=1 (or zero flag is 1)

Direct **:** instruction_format **(6XX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <- MDR(8-11)
T3:  Idle
T4:  Idle
T5:  PC(L) , MDR(E), Reset          // if Z=1, PC <- MDR(7-0)
                                       else Reset
```

Indirect: instruction_format **(EXX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) , IR(L) , MAR(L,E)      // MAR <- MDR(7-0), IR <- MDR(11-8)
T3:  MDR(L) , RAM(E)                // MDR <- M[MAR]
T4:  Idle
T5:  PC(L) , MDR(E), Reset          // if Z=1, PC <- MDR(7-0)
                                       else Reset
```

For non-MRI Instructions

ADD B **:** instruction_format **(70_)**

Instruction to add values present in RAM/Registers.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , B(E) , ALU(000) , Reset

SUB B **:** instruction_format **(71_)**

Instruction to subtract values present in RAM/Registers.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , B(E) , ALU(001) , Reset

CLA **:** instruction_format **(72_)**

This instruction specifies to clear the accumulator.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle

T5: A(E,L) , ALU(002) , Reset

CMA : instruction_format (73_)

This instruction specifies to complement the value present in the accumulator.

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                    // IR <- MDR(4-7)
T3:  Idle
T4:  Idle
T5:  A(E,L) , ALU(003) , Reset
```

MOV A,B : instruction_format (74_)

This instruction specifies to move the data present in register B to register A.

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                    // IR <- MDR(4-7)
T3:  Idle
T4:  Idle
T5:  B(E), A(L) , Reset               //  A <- B
```

MOV B,A : instruction_format (75_)

This instruction specifies to move the data present in register A to register B.

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                    // IR <- MDR(4-7)
T3:  Idle
T4:  Idle
```

T5: B(L), A(E) , Reset   // B <- A

PUSH **:** instruction_format **(76_)**

This instruction is used to push the data from A to the address pointed by SP in RAM.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(D)
T6: SP(E) , MAR(E,L)   // MAR <- SP
T7: RAM(L) , A(E) , Reset   // M[MAR] <- A

POP **:** instruction_format **(77_)**

This instruction is used to pop the data from the address pointed by SP in RAM.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(E) , MAR(E,L)   // MAR <- SP
T6:   RAM(E) , A(L)   // A <- M[MAR]
T7: SP(I) , Reset

RET **:** instruction_format **(78_)**
Retrieves PC from the address pointed by SP in RAM.
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(E) , MAR(E,L)   // MAR <- SP

T6:   RAM(E) , PC(L)   // PC <- M[MAR]
T7: SP(I) , Reset

MVI A , data   :   instruction_format **(79X)**

This instruction specifies to move the data immediately in A.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), A(L) , Reset   // A <- MDR(0-3)

MVI F , data   :   instruction_format **(7AX)**

This instruction specifies to move the data immediately in F.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), F(L) , Reset   // F <- MDR(0-3)

LSC A , data   :   instruction_format **(7BX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L)   // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), BS(L)   // BS <- MDR(0-3)
T6:   A(E,L), BS(E), Reset

RSC A , data **:** instruction_format **(7CX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), BS(L)                  // BS <- MDR(0-3)
T6:   A(E,L), BS(E), Reset
```

RRX A , data **:** instruction_format **(7DX)**

Input carry, A and tmp value which has one time to rotate to the barrel shifter and note the answer in the accumulator

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), BS(L)                  // BS <- MDR(0-3)
T6:   A(E,L), BS(E), Reset
```

ASR A , data **:** instruction_format **(7EX)**

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), BS(L)                  // BS <- MDR(0-3)
T6:   A(E,L), BS(E), Reset
```

HLT : ** instruction_format **(7F_)

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                   // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  disable SG , Reset              // SG - sequence generator
```

IN S : ** instruction_format **(F0X)

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                   // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), IS(L)                   // IS <- MDR(0-3)
T6:   A(L), IN(E), FGI(0) , Reset    // A <- IN, FGI <- 0,
                                     (IN Port selected by IS )
```

OUT S : ** instruction_format **(F1X)

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                   // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), OS(L)                   // OS <- MDR(0-3)
T6:   A(E), OUT(L), FGO(0), Reset    // OUT <- A, FGO <- 0,
                                     ( OUT Port selected by OS )
```

SKI S : ** instruction_format **(F2X)

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), IS(L)                  // IS <- MDR(0-3)
T6:   PC(I), Reset          // if FGI=1,PC<-PC+1, ( FGI is selected by IS )
                               else Reset
```

SKO S : ** instruction_format **(F3X)

```
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2:  MDR(E) ,IR(L)                  // IR <-MDR(4-7)
T3:  Idle
T4:  Idle
T5:  MDR(E), OS(L)                  // OS <- MDR(0-3)
T6:   PC(I), Reset         // if FGO=1,PC<-PC+1,( FGO is selected by OS)
                              else Reset
```