

# ***SMART FRIDGE***

*COA Lab Project (2016-2017)*

*The LNMIIIT, Jaipur*



*Sakshi Sachar (15UCS117)*

*Samkit Jain (15UCS119)*

*Shipra Gupta (15UCS127)*

*Shorya Agarwal (15UCS132)*

# INTRODUCTION



- Intelligent appliances with multimedia capability have been emerging into our daily life.
- The Smart Refrigerator module is designed to convert any existing refrigerator into an intelligent cost effective appliance using sensors .
- The idea of connecting home appliances to the smart home environment has been seen as the future and is highly regarded as the next big thing.





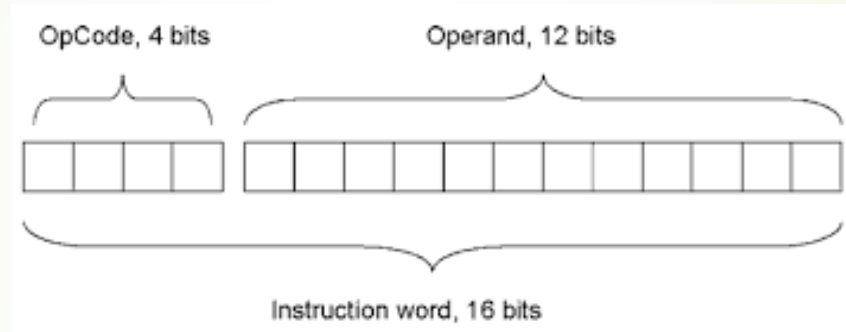
# FUNCTIONS

- Grocery tracker : Will track the grocery load in the basket and will set alert if less than required.
- Food freshness tracker : Scan the expiration date of items and send alerts.
- Water filling status : will contain the sensor to detect how much water to dispense into your containers.
- Instant cooling zone : Whenever you place an item will lower its temperature in 1 minute and set notification.
- Energy Monitor : Will display that how much power has been consumed in a day when the compressor starts working or when the door opens and the lights get on.
- Refrigerator cum freezer : Will specify how to use this zone.

- 
- 
- Autofill : Will automatically fill the container to its full capacity
  - Precise fill : Will fill the specified quantity of water.
  - Multi-temperature zone : Specify the temperature of each zone.
  - Compressor : Will keep a check of the volume of gas.

# MEMORY MODEL

- Since the ISA we are designing is for a 16-bit machine therefore word length becomes 16-bit or 2 bytes.



- We are using byte addressing and *ALIGNED* memory model because memories operate efficiently because of alignment.



# REGISTERS

32 General Purpose Registers which are used for storing values of temporary variables or memory addresses

R00	00000	R10	01010
R01	00001	R11	01011
R02	00010	R12	01100
R03	00011	.	
R04	00100	.	
R05	00101	.	
R06	00110	R31	11111
R07	00111		
R08	01000		
R09	01001		

## 3 Special Purpose Registers

PC – Program Counter

IR – Instruction Register

FR – Flag Register

PC 1101

IR 1100

FR 1111

The FR has following condition code bits which are set when the result is:

N – Negative

P – Even parity

Z - zero

V – overflow

C – Carry

A – Auxillary carry

# INSTRUCTION FORMAT

Two – address instructions

Opcode	Register	Register
6 bit	5 bit	5 bit

INSTRUCTION FORMAT	DESCRIPTION
MOV R1,R2	Copies content of R1 into R2
CMP R1,R2	Compares value of R1 and R2
ADD R1,R2	Add contents of R1 ,R2 and store in R2
SUB R1,R2	Subtract contents of R1 ,R2 and store in R2
MUL R1,R2	Multiply contents of R1 ,R2 and store in R2
DIV R1,R2	Divison contents of R1 ,R2 and store in R2
AND R1,R2	Logical AND
OR R1,R2	Logical OR


Opcode	Register	Immediate
3 bits	5 bits	8 bits

INSTRUCTION FORMAT	DESCRIPTION
MOVIR01 ,#value	<i>This can be used to move constant values in registers Ex : Registers</i>
CMPIR01 ,#value	<i>Compare</i>
ADDI R01 ,#value	<i>Add</i>
SUBI R01 ,#value	<i>Subtract</i>
MULI R01 ,#value	<i>Multiplication</i>
DIVIR01 ,#value	<i>Divide</i>




# One Address Instruction

Opcode	Logical Address
8 bits	8 bits



<i><b>INSTRUCTIONS</b></i>	<i><b>ADDRESSES</b></i>
JMP \$LO	Jump unconditionally to memory address
JEQ \$LO	Jump if equal
JNQ \$LO	Jump if not equal
JNZ \$LO	Jump if zero
JGT \$LO	Jump if greater than
JLT \$LO	Jump if less than



Opcode	Register
11 bits	5 bits

Instruction	Description
LSHIFT R01	Left Shift the value in register R01
RSHIFT R01	Right Shift the value in register R01
NOT R01	Invert the bits
IN R01	Take input in register R01
OUT R01	Take output from register R01

Opcode	
16 bits	
Instruction	Description
HLT	Halt the program
RET	Return



# INSTRUCTION DESIGN

- *Reduced Instruction Set Computer(RISC) is used as it can be implemented easily as they are simple instructions and can be executed in one clock cycle.*
- *Fixed length instructions : Which are easy to decode. Length of each instruction is one word i.e. 16 bits.*

## Two address instructions

-with immediate addressing

ADDI 000 *ri*

SUBI 001 *ri*

MULI 010 *ri*

DIVI 011 *ri*

CMPI 100 *ri*

MOVI 101 *ri*

-with register – register addressing

ADD 110000 *rr*

SUB 110001 *rr*

MUL 110010 *rr*

DIV 110011 *rr*

AND 110100 *rr*

OR 110101 *rr*

CMP 110110 *rr*

MOV 110111 *rr*

## One address instructions

-with Label/Logical address

JNQ 111000 a

JEQ 111001 a

JLT 111010 a

JGT 111011 a

JMP 111100 a

- with register addressing

IN 11110100000 r

OUT 11110100001 r

NOT 11110100010 r

LSHIFT 11110100011 r

RSHIFT 11110100100 r

## Zero address instruction

RET 1111010001000000 z

MSF 1111010001000001 z

HLT 1111010001000010 z



# INSTRUCTION TYPES

- *Data Transfer –*

- *Register to Register: MOV1*
- *Memory to Register: LDB*
- *Register to Memory : STB*
- *Constant into Register: MOV2*

- *Dyadic –*

- *MOV1 , MOV2 , ADD , SUBT , MUL ,DIV , CMP ,ADD , OR*


- *Comparison and Conditional branching –*

- *JMP , JNQ , JEQ ,JNZ ,JGT ,JLT , CMP*





# ***DATA TYPES***

- *Unsigned Integers : Range(0 to 65535)*
  - *ASCII Code : 7 bit characters*
  - *Bitmap : 16 bit word can hold 16 Boolean values*
- 

# ADDRESSING MODES

*Immediate Addressing Mode –*

Opcode	Immediate	Register
MOV1	#4	R00

*Register Addressing Mode –*

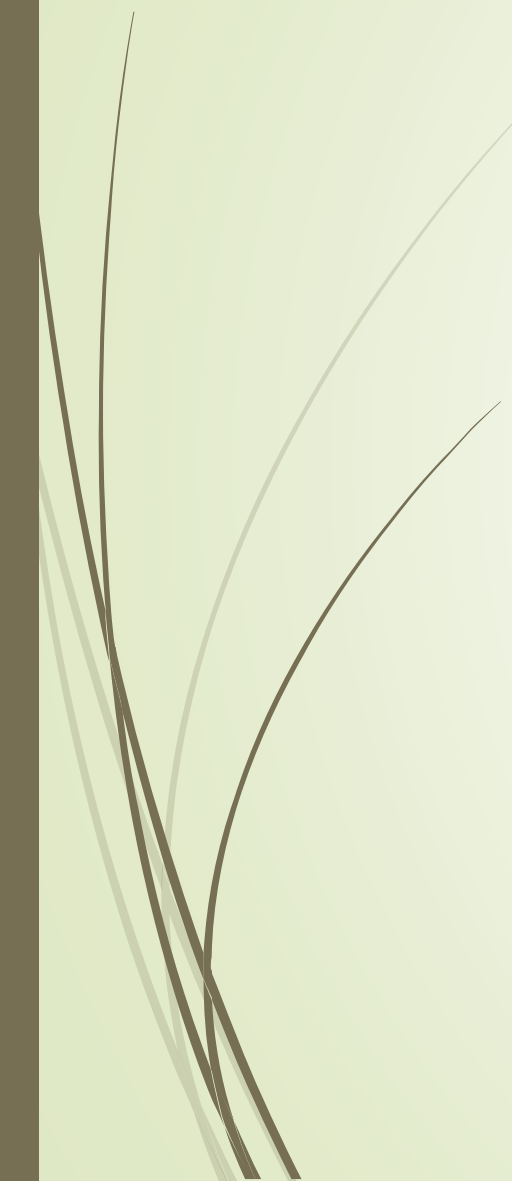
Opcode	Register	Register
MOV1	R00	R01

*Register Indirect Mode –*

Opcode	Register	Register
LDB	R1	@R2

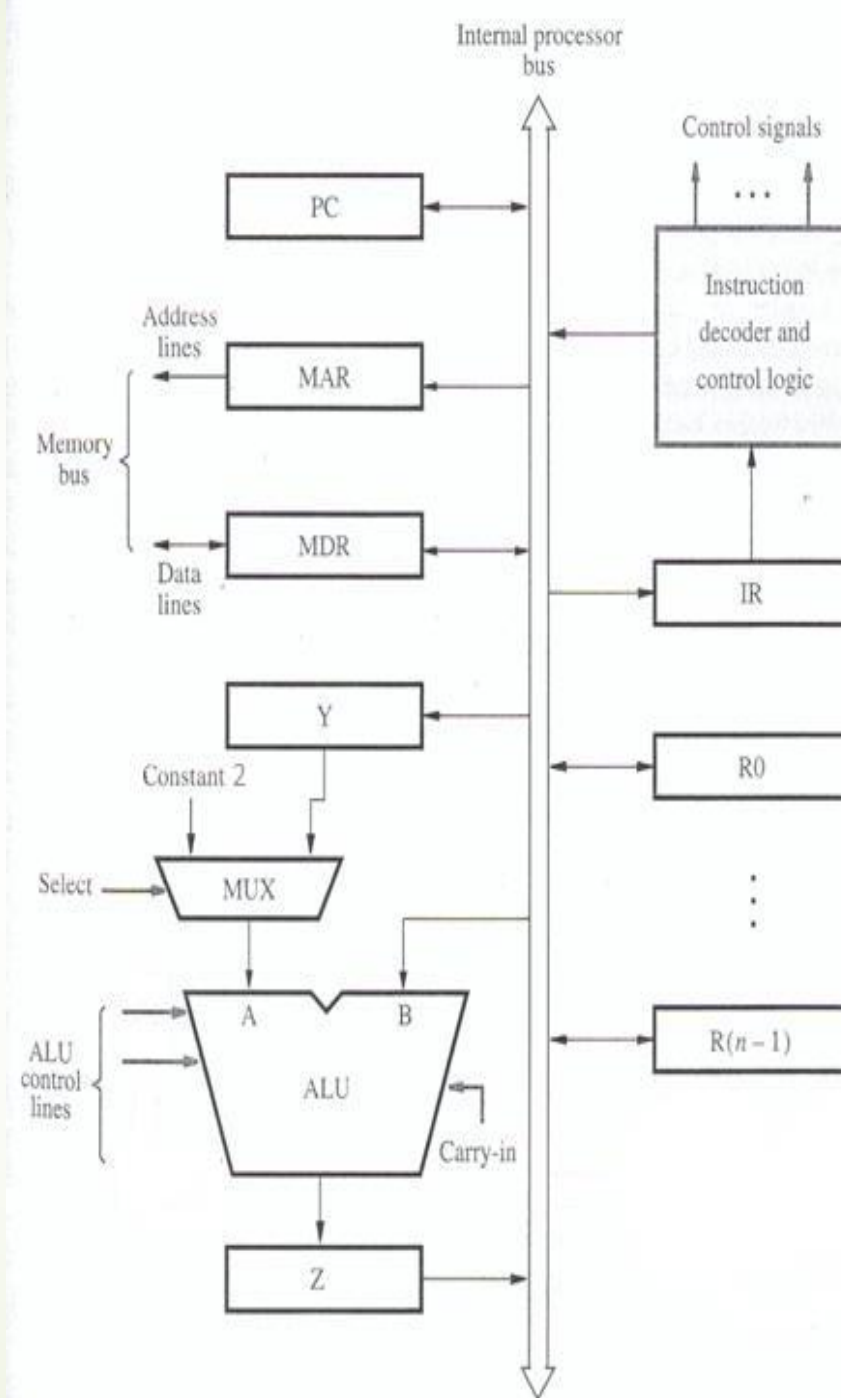


# FLOW OF CONTROL HANDLING

- Generally, the flow of control is **sequential** but when labels are included, branching is used as flow of control is transferred to logical address where label lies. It can be done using instructions like JMP in our ISA.
- 

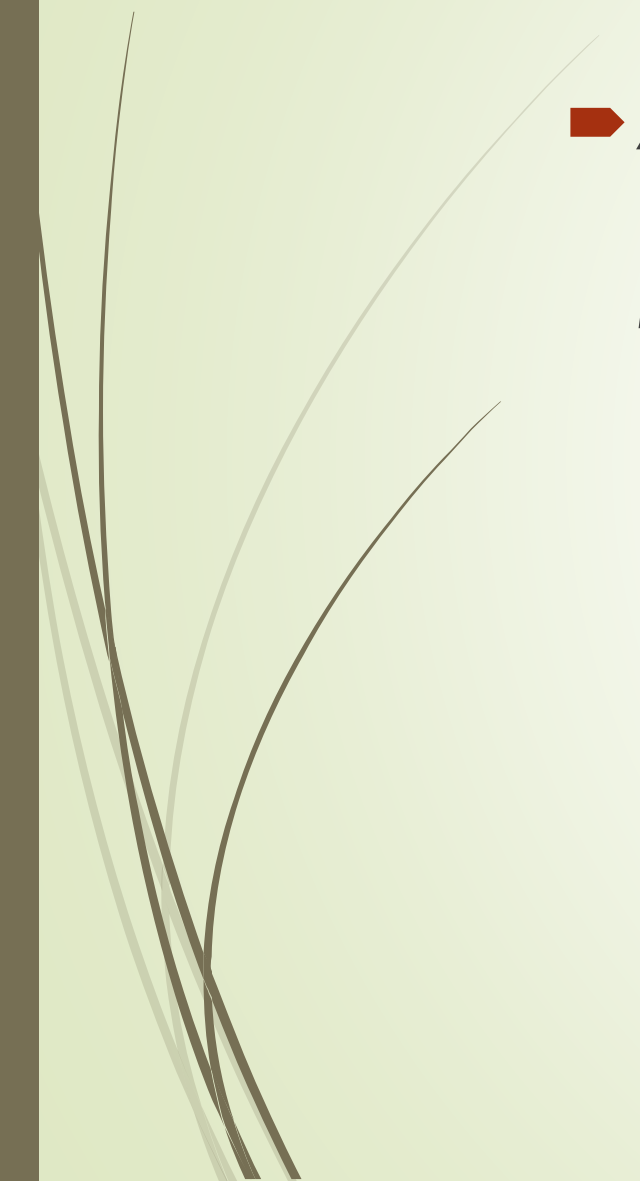
# DATAPATH – Single Bus Architecture

- Datapath is a collection of functional units, such as arithmetic logic units or multipliers, that perform data processing operations, registers, and buses. Along with the control unit it composes the central processing unit (CPU).

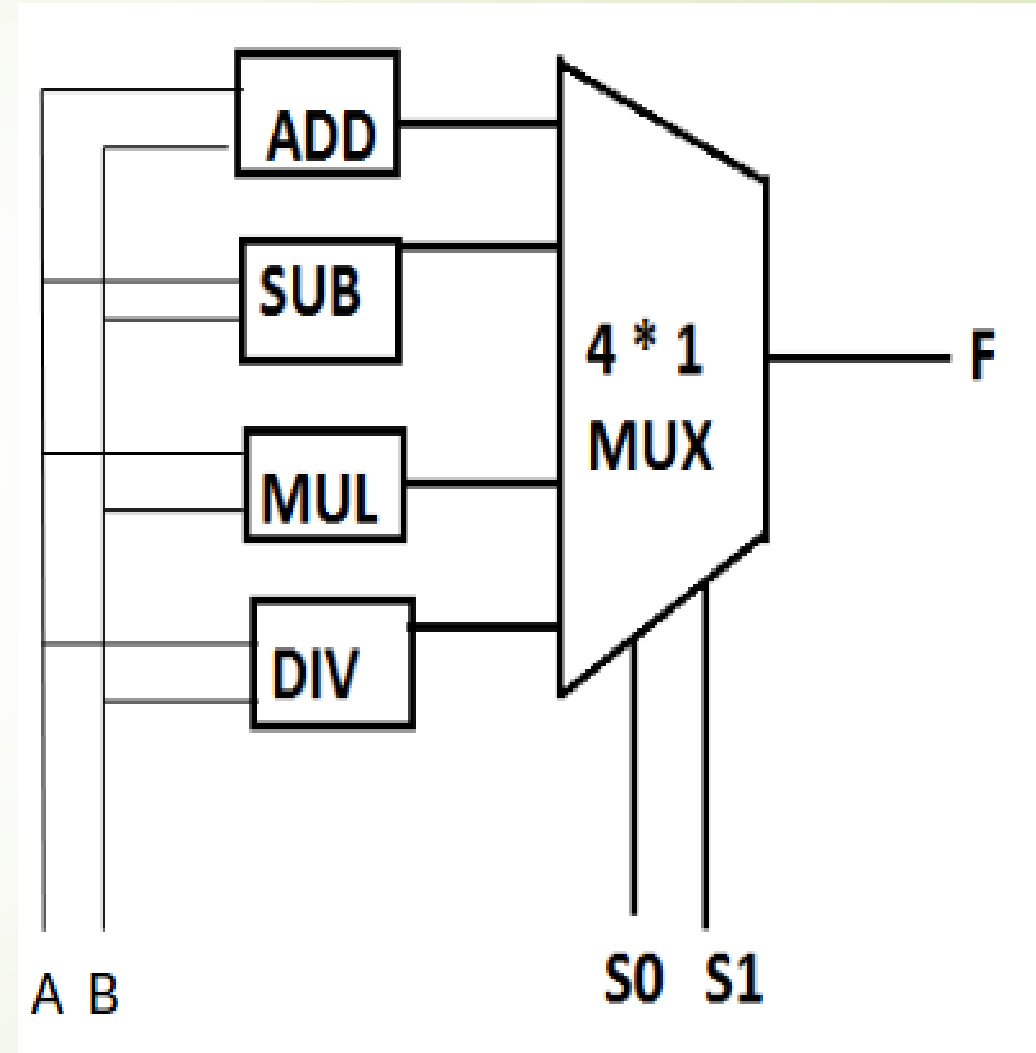




# ARITHMETIC LOGIC UNIT

- An arithmetic logic unit (**ALU**) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer.
- 

OPERATION	S0	S1
ADD	0	0
SUB	0	1
MUL	1	0
DIV	1	1





# MICROINSTRUCTIONS

## ➤ FETCH CYCLE

1. PCout , MARin ,Read , select 2 , ADD , Zin
2. WMFC , Zout, Yin, PCin
3. MDRin-ext , MDRout , IRin
4. Branching to appropriate microroutine.

FETCH CYCLE

MicroInst <sup>n</sup>	PCIN	PCOUT	MARIN	MAROUT	MDRIN-E	READ	WMFC	YIN	ZIN	IRIN	MDROUT	ADD	SELECT2	ZOUT
1.	0	1	1	0	0	1	0	0	1	0	0	1	1	0
2.	1	0	0	0	0	0	1	1	0	0	0	0	0	1
3.	0	0	0	0	1	0	0	0	0	1	1	0	0	0

## ADD R1 R2

1. Fetch cycle
2. R1out, Yin ,  
select Y
3. R2out, ADD ,  
Zin
4. Zout, R1in, END

ADD

MicroInst	R1out	R2out	R1IN	R2IN	YIN	SYOUT/ SELECT Y	ADD	ZIN	ZOUT	END
1.	1	0	0	0	1	1	1	0	0	0
2.	0	1	0	0	0	0	1	1	0	0
3.	0	0	1	0	0	0	0	0	1	0



## SUB R1 R2

1. Fetch cycle
2. R1out, Yin ,  
select Y
3. R2out, SUB , Zin
4. Zout, R1in, END

SUB R1, R2

MicroInst	R1 <sub>out</sub>	R2 <sub>out</sub>	R1 <sub>IN</sub>	R2 <sub>IN</sub>	Y <sub>IN</sub>	SNOUT/ SELECT Y	SUB	Z <sub>IN</sub>	Z <sub>OUT</sub>	END
1.	1	0	0	0	1	1	1	0	0	0
2.	0	1	0	0	0	0	0	1	1	0
3.	0	0	1	0	0	0	0	0	1	0

## MUL R1 R2

1. Fetch cycle
2. R1out, Yin ,  
select Y
3. R2out, MUL ,  
Zin
4. Zout, R1in, END

MUL R1, R2

MicroInst	R1out	R2out	R1IN	R2IN	YIN	SYOUT/LOOK	SELECT Y	MUL	ZIN	ZOUT	END
1.	1	0	0	0	1	1	1	0	0	0	0
2.	0	1	0	0	0	0	0	1	1	0	0
3.	0	0	1	0	0	0	0	0	0	1	0

## DIV R1 R2

1. Fetch cycle
2. R1out, Yin ,  
select Y
3. R2out, DIV , Zin
4. Zout, R1in, END

DIV R1, R2

MicroInst	R1 <sub>out</sub>	R2 <sub>out</sub>	R1 <sub>IN</sub>	R2 <sub>IN</sub>	Y <sub>IN</sub>	Y <sub>OUT</sub>	SELECT Y	DIV	Z <sub>IN</sub>	Z <sub>OUT</sub>	END
1.	1	0	0	0	1	1	1	0	0	0	0
2.	0	1	0	0	0	0	0	1	1	0	0
3.	0	0	1	0	0	0	0	0	0	1	0



## CMP R1 R2

1. Instruction Fetch  
R1out, Yin, select Y,  
Yout
2. R2out, SUB, Zin
3. If  $Z < 0$  then  $N = 1$ ,  
END
4. If  $Z = 0$  then Zero  
flag = 1, END
5. END

CMP R1, R2

MicroInst	R1out	R2out	R1IN	R2IN	YIN	YOUT	SELECT Y	SUB	ZIN	ZOUT	END
1.	1	0	0	0	1	1	1	0	0	0	0
2.	0	1	0	0	0	0	0	1	1	0	0
3.	0	0	0	0	0	0	0	0	0	0	1

appropriate flag will be set according to condition and then signal for END is generated.



## JMP L1

1. Instruction Fetch
2. IRout ,  
OFFSETFIELDout,  
select Y, ADD, Zin
3. Zout, Pcin ,Yin,  
END

JMP LABEL

MicroInst <sup>n</sup>	IRout	SELECT Y	ADD	ZIN	ZOUT	PCIN	YIN	OFFSET OUT	END
1.	1	1	1	1	0	0	0	1	0
2.	0	0	0	0	1	1	1	0	1

## JNQ L1

1. Instruction Fetch
2. IRout,  
OFFSETFIELDout,  
select Y, ADD, Zin
3. If Zero flag(Z)=1  
then END
4. Zout, Pcin, Yin,  
END

JNQ LABEL

Micro Inet <sup>n</sup>	IRout	SELECT Y	ADD	ZIN	ZOUT	PCIN	YIN	OFFSET OUT	END
1.	1	1	1	1	0	0	0	1	0
2.	0	0	0	0	0	0	0	0	1
3.	0	0	0	0	1	1	1	0	1

→ If zero flag = 1

## JEQ L1

1. Instruction Fetch
2. IRout ,  
OFFSETFIELDout,  
select Y, ADD, Zin
3. If Zero flag (Z)=0  
then END
4. Zout, Pcin ,Yin,  
END

JEQ LABEL

MicroInst <sup>n</sup>	IR <sub>OUT</sub>	SELECT Y	ADD	Z <sub>IN</sub>	Z <sub>OUT</sub>	PC <sub>IN</sub>	Y <sub>IN</sub>	OFFSET <sub>OUT</sub>	END
1.	1	1	1	1	0	0	0	1	0
2.	0	0	0	0	0	0	0	0	1
3.	0	0	0	0	1	1	1	0	1

→ If zero flag = 0



## JLT L1

1. Instruction Fetch
2. IRout ,  
OFFSETFIELDout, select  
Y, ADD , Zin
3. If N=0 then END
4. Zout , PCin, Yin, END

JLT LABEL

Micro Instr <sup>n</sup>	IRout	SELECT Y	ADD	ZIN	ZOUT	PCIN	YIN	OFFSET OUT	END
1.	1	1	1	1	0	0	0	1	0
2.	0	0	0	0	0	0	0	0	1
3.	0	0	0	0	1	1	1	0	1

N=0 → if

## JGT L1

1. Instruction Fetch
2. IRout ,  
OFFSETFIELDout, select  
Y, ADD , Zin
3. If N=1 or Zero flag=1  
then END
4. Zout , PCin, Yin, END

JGT LABEL

MicroInst <sup>n</sup>	IRout	SELECT Y	ADD	ZIN	ZOUT	PCIN	YIN	OFFSET OUT	END
1.	1	1	1	1	0	0	0	1	0
2.	0	0	0	0	0	0	0	0	1
3.	0	0	0	0	1	1	1	0	1

N=1  
OR  
zero flag=1

1. PCout , MARin , Read , select 2 , ADD , Zin
2. WMFC , Zout, Yin, PCin
3. MDRin-ext , MDRout , IRin
4. R1in , R2out, END

1. PCout , MARin , Read , select 2 , ADD , Zin
2. WMFC , Zout, Yin, PCin
3. MDRin-ext , MDRout , IRin
4. R1in , R2out, END

[illegible]



# ADDI R1 , Im

# 1. Instruction Fetch

## 2. R1 out, Yin, select Y

### 3. DCout, ADD, Zin

## 4. Zout, R1in, END

R1 <sub>OUT</sub>	Y <sub>IN</sub>	SELECTY	Y <sub>OUT</sub>	DC <sub>OUT</sub>	ADD	Z <sub>IN</sub>	Z <sub>OUT</sub>	R1 <sub>IN</sub>	END
1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1

# SUBI R1 , Im

# 1. Instruction Fetch

## 2. R1 out, Yin, select Y

### 3. DCout, SUB, Zin

## 4. Zout, R1in, END

R1 <sub>OUT</sub>	Y <sub>IN</sub>	SELECTY	Y <sub>OUT</sub>	DC <sub>OUT</sub>	SUB	Z <sub>IN</sub>	Z <sub>OUT</sub>	R1 <sub>IN</sub>	END
1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1





# DIVI R1 , Im

# 1. Instruction Fetch

## 2. R1 out, Yin, select Y

### 3. DCout, DIV, Zin

## 4. Zout, R1in, END

DIV Reg, Im

R1 <sub>OUT</sub>	Y <sub>IN</sub>	SELECTY	Y <sub>OUT</sub>	DC <sub>OUT</sub>	DIV	Z <sub>IN</sub>	Z <sub>OUT</sub>	R1 <sub>IN</sub>	END
1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1

## CMPI R1, Im

1. Instruction Fetch
2. R1out, Yin, Yout, select Y
3. DCout, SUB, Zin
4. If Z=0 then Zero flag=1, END
5. If Z<0 then N=1, END
6. END

CMPI Reg, Im

MicroInst	R1out	DCout	R1IN	R2IN	YIN	Yout	SELECT Y	SUB	ZIN	ZOUT	END
1.	1	0	0	0	1	1	1	0	0	0	0
2.	0	1	0	0	0	0	0	1	1	0	0
3.	0	0	0	0	0	0	0	0	0	0	1

appropriate flag will be set according to condition and then signal for END is generated.





## MSF

1. PCout, MARin, Read, Zin, select 2, ADD
2. PCin, WMFC, Yin, Zout
3. MDRin-ext, IRin, MDRout
4. PCout, Zin, select 2, ADD
5. Zout, Yin, MSFin, END

MSF																
Micro Instrul	PCIN	PCOUT	MARIN	MAROUT	MDRIN-E	READ	WMFC	YIN	ZIN	IRIN	MDROUT	ADD	SELECT2	ZOUT	MSFIN	END
1.	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
2.	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
3.	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0
4.	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0
5.	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1







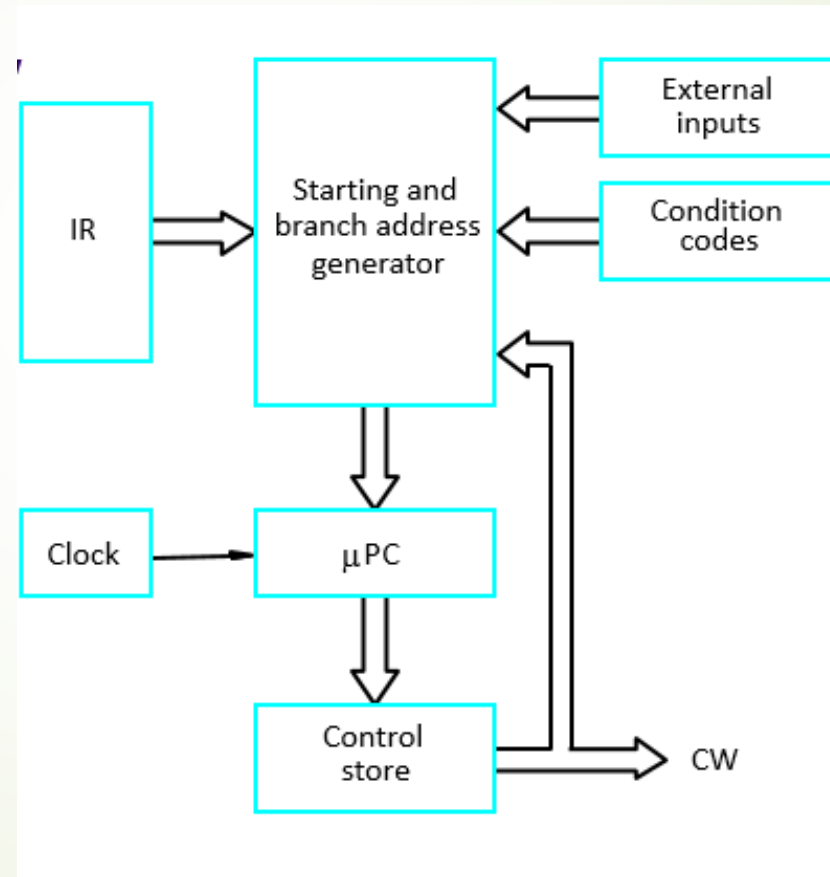
## **IN R1**

1. Instruction Fetch
2. Interrupt signal for input ,R1in, END

## **HLT**

1. Instruction Fetch
2. HLT, END

# CONTROL UNIT for MICROPROGRAMMED CONTROLLED





***THANK YOU !!***