Microprocessor

Week ①

Integrated circuit

Small scale Integration (S.S.I) - A few logic gates.

Medium scale Integrated circuit (MSI) - A complete integrated circuit counter

Large scale Integrated circuit (LSI) - more than 10,000 individual transistors on ~~silicon~~ single silicon chip.

Mode of operation

Sequence of operation → computer can perform a number of basic operations called machine instructions. which the user selects and orders in a way which solves a particular problem. This sequential list of operation is referred to as a program

A digital computer utilises the very high speed of execution of each machine instruction usually a few micro seconds by having the required sequence ~~code~~ of instructions (or) Program stored within the computer itself. This is known as the stored Program concept and is the fundamental difference between a basic calculator and computer system.

Input Temperature Output Turn on heating element.

Decimal system

10^4	10^3	10^2	10^1	10^0	
4	9	5	3	6	= 49536

$4\times10^4 + 9\times10^3 + 5\times10^2 + 3\times10^1 + 6\times10^0 = 49536$

Binary system

2^4	2^3	2^2	2^1	2^0	
1	0	1	1	1	$= 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$
					$= 23$

Hex decimal number

4 bit binary pattern	Hexa decimal Symbol
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

0110 1101 = 6D (Hex)
6 D

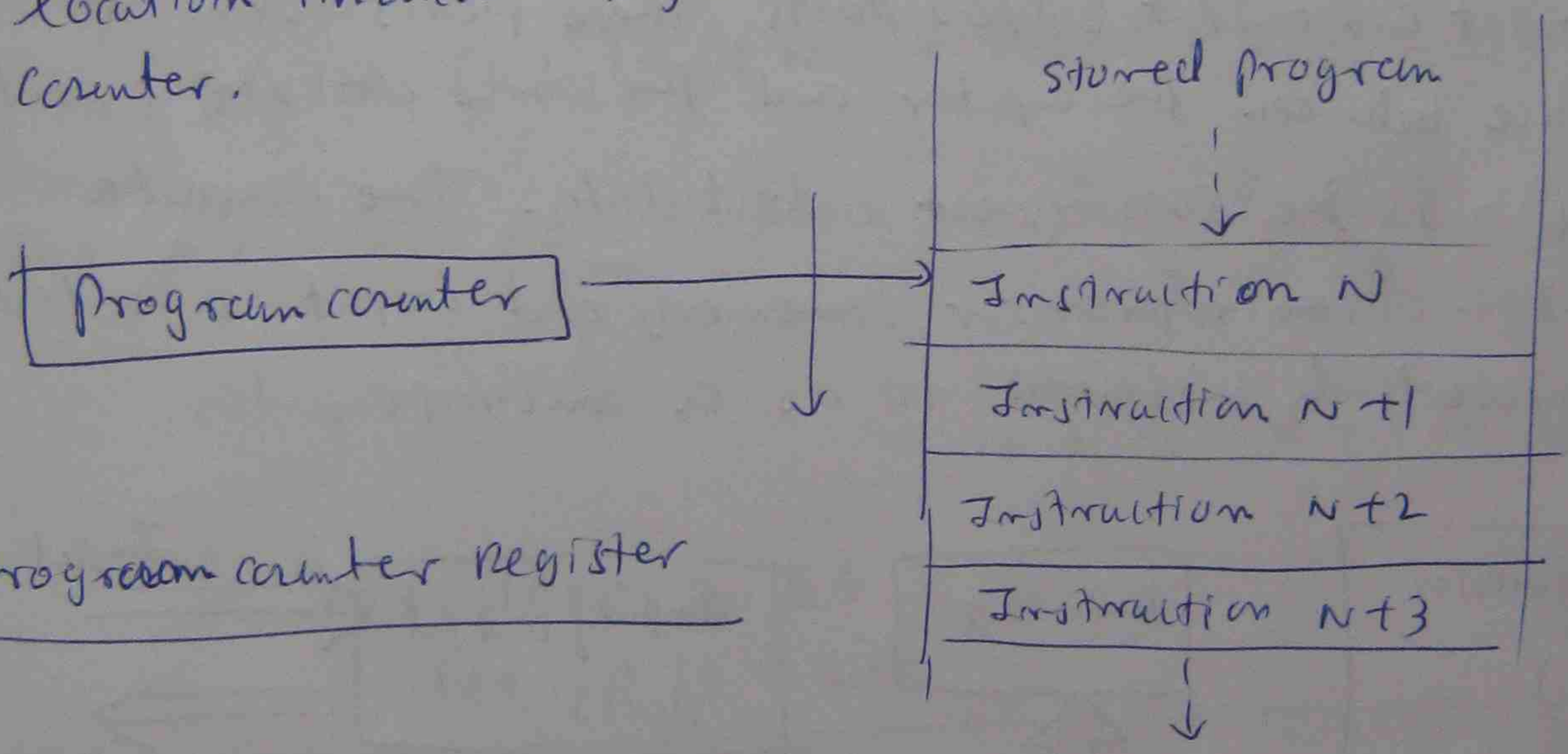
1111 0010 = F2 (Hex)
F 2

1011 0100 1000 1110 = B48E (Hex)
B 4 8 E

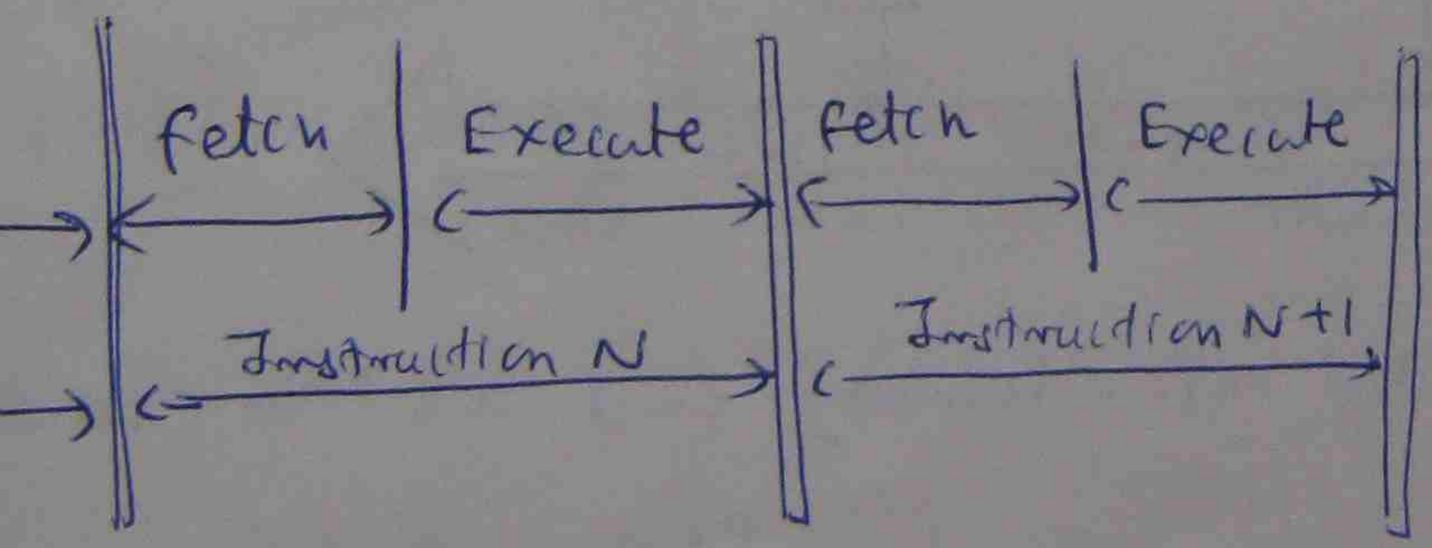
Program counter

In order to remember which program instruction is to be executed next, the microprocessor contains a register (or temporary information storage location) called the Program counter (PC), the contents of which points to the next sequential instruction to be fetched and executed.

Thus during a typical instruction cycle, the next instruction to be executed is read from the memory location indicated by the contents of the Program counter.



Program counter Register



Problem

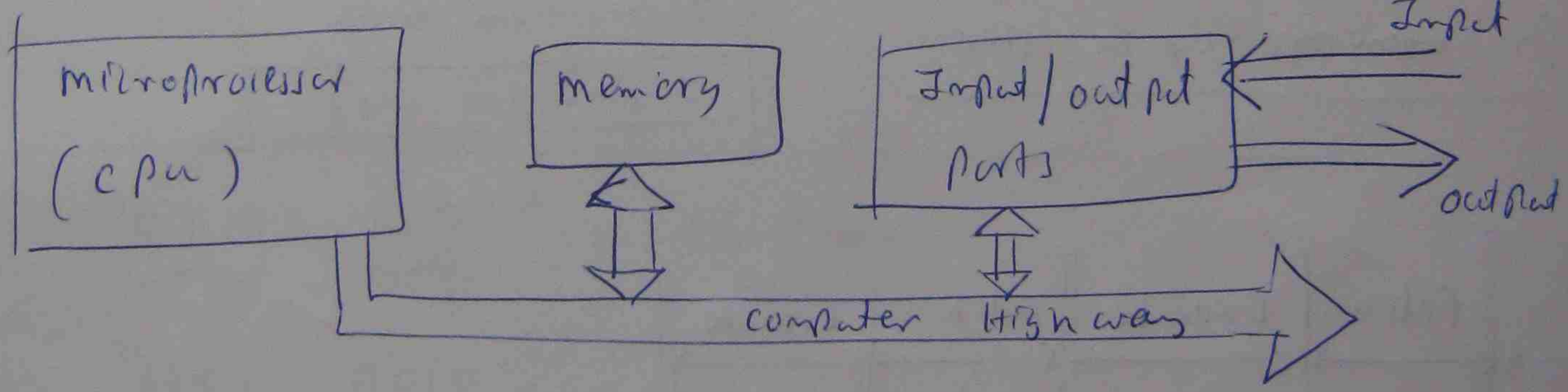
(1) Convert the following decimal numbers into their equivalent binary numbers.

(a) 35, (b) 67 (c) 224

Basic structure and operation

A digital computer executes a list of basic machine instructions (the program) which have been selected and ordered by the user to solve a particular task.
In order to exploit ~~the~~ the intrinsic high speed of execution of each machine instruction, the program is stored within the computer.

A basic digital computer is comprised of a memory which is primarily used to hold (or) store the program, and some input and output (I/O) ports. These ports form the interface between the computer and the source of input data and the subsequent output data. The complete ~~output~~ combination of microprocessor, memory and input & output ports is collectively referred to as a microcomputer.



→ complete program is loaded into memory and is then executed.

microprocessor
- → Phase(1)
  The next instruction is fetched from memory (Fetch cycle)
- → Phase(2)
  The next instruction is fetched from memory; then, in the second phase (or) execution cycle. The microprocessor executes (or performs) the action specified by the instruction

Pb①

35

2	35	1
2	17	1
2	8	0
2	4	0
2	2	0
	1	

Exercise

1000 11 (9)

To make 8 bit put 0, 0

00 1000 11

67 = 01000011

224 = 11100000

Pb② Convert the following binary numbers into their equivalent decimal numbers.

(a) 10 111 (b) 10 10101 (c) 110 110 11

(a) 10 111 = 000 10 111 = $0\times2^7 + 0\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$

= 16 + 4 + 2 + 1 = 23

Similarly 10 10101 = 85

110 110 11 = 219

Pb③ convert the following decimal numbers into their equivalent hexadecimal number.

(a) 27 (b) 96 (c) 3334

27 =

2	27	1
2	13	1
2	6	0
2	3	1
	1	

110 11 ⇒ 000 | 10 11

= 1B

(b) 96 =>

2	96 – 0
2	48 – 0
2	24 – 0
2	12 – 0
2	6 – 0
2	3 – 1
	1

(6)

1100000 = 0110, 0000

6 0

60 (Hex)

(c) 3334 => 110, 100000110 = D06 (Hex)

D 0 6

Pb(4) convert the following hexa decimal numbers into their equivalent decimal numbers.

(a) D3 (b) 2F (c) 2D9E

(a) D3 = 1101, 0011 = $1\times2^7+1\times2^6+0\times2^5+1\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0 = 211$

(b) 2F = 0010, 1111 = $0\times2^7+0\times2^6+1\times2^5+0\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0 = 47$

(c) 2D9E = 0010, 1101, 1001, 1110

$= 0\times2^{15}+0\times2^{14}+1\times2^{13}+0\times2^{12}+1\times2^{11}+1\times2^{10}+0\times2^9+1\times2^8+1\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+0\times2^0$

$= 11678$

D027

Microprocessor

Week (2)

Microcomputer Architecture

Microprocessor (CPU)

Memory → Hold the stored program.

Input & output ports → Interface the micro computer to the various input & output devices controlled by it.

Microprocessor

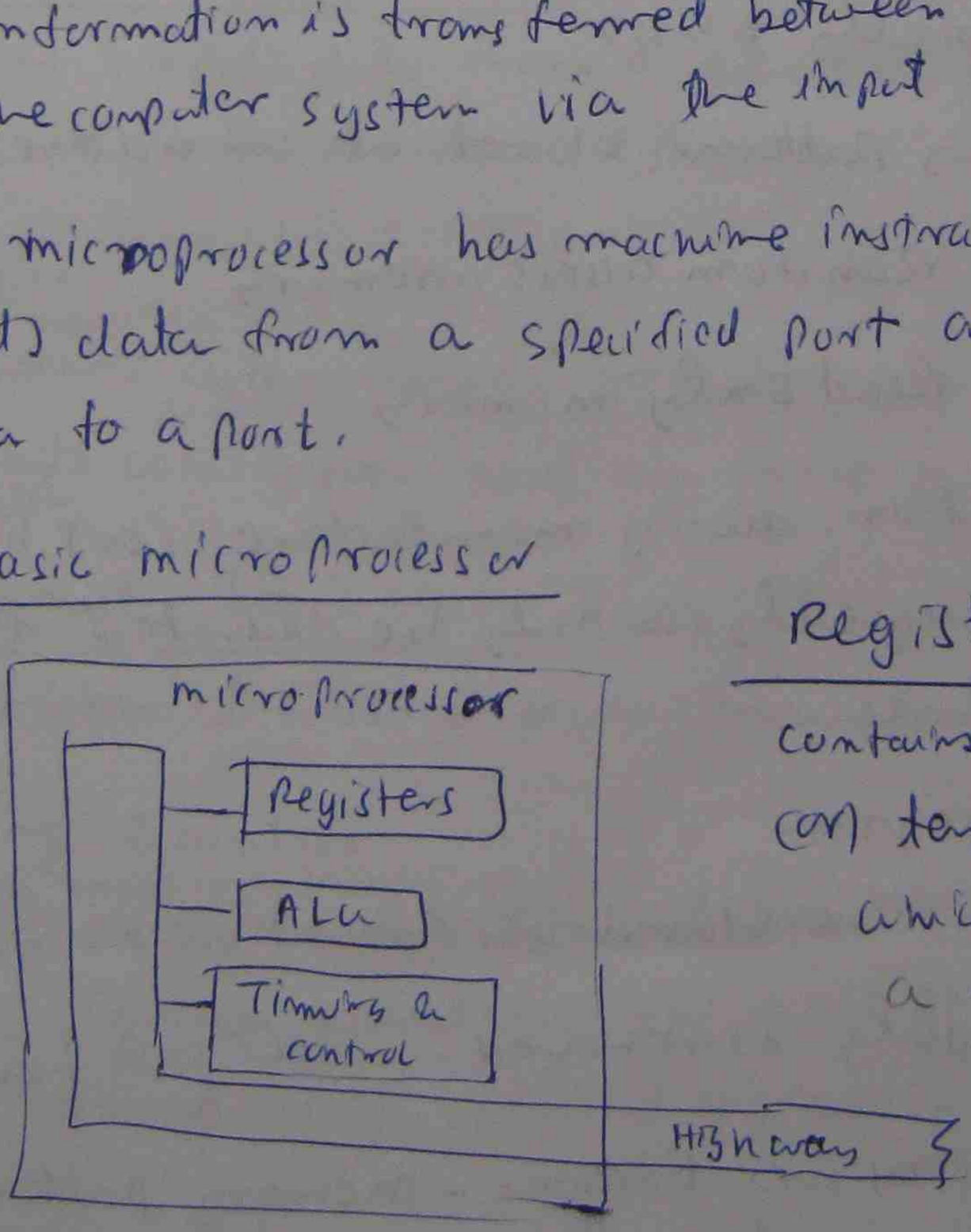
Individual data byte manipulation instructions (add, subtract)

memory transfer instructions (read data byte from memory
write data byte to memory)

Information is transferred between external devices and the computer system via the input & output ports.

The microprocessor has machine instructions to both read (input) data from a specified port and to write (output) data to a port.

Basic microprocessor



Register

contains a number of registers (or) temporary storage elements which can hold or store a single byte or word

(ALU) Arithmetic Logic Unit - Performs the actual data manipulation.

Timing & control section - co-ordinates the internal operation of the ALU and registers so that the desired action specified by an instruction is performed.

The microprocessor communicates with the memory, both to obtain the individual instructions which make up the program and to access & store data and to transfer data to and from input and output using a highway (or) bus.

Memory

Locations ⟷ address. - Each location contains a binary pattern with a number of bits corresponding to the word length of the computer (typically 8 bits).

Content - The binary pattern stored at an address.

Memory → RAm - Random access memory
→ Rom - Read only memory.

Rom - Fixed information during manufacture (or) by the user and consequently can only be operated in a read only mode.
- Hold fixed program
- Non volatile. The stored information is not lost when power supply is removed. low cost / and

Erasable Programmable Roms (or) EPRom - memory pattern can be changed by the user in a controlled manner.

Week 4 (2)

Microcomputer Architecture

~~#5~~ (7)

Functional Units of a microcomputer

microprocessor — cpu
The memory — It is used to hold the stored program.
Input/output Ports — To interface the micro computer to the various input and output devices controlled by it.

Microprocessor

- Execute a number of basic machine instructions.
- Data byte manipulation, instruction (add, subtract)
- memory transfer instruction (Read data byte, transfer between external devices and the computer system via the input and output ports,)
- machine instructions

  Read (input) data from a specified port
  write (output) data to a port.

Register

- contains a number of registers or temporary storage elements which can hold or store a single byte or word.

Arithmetic Logic Unit (ALU)

- Perform the actual data manipulation operations.

Timing & control

- co-ordinate the internal operation of the microprocessor and controls operation of the ALU and registers so that the desired action specified by an instruction is performed.

(16) (2)

Microprocessor communication

- The microprocessor communicates with the memory both to obtain the individual instructions which make up the program and to access and store data.
- To transfer data to and from input and output ports using a highway (or) bus.

Erase

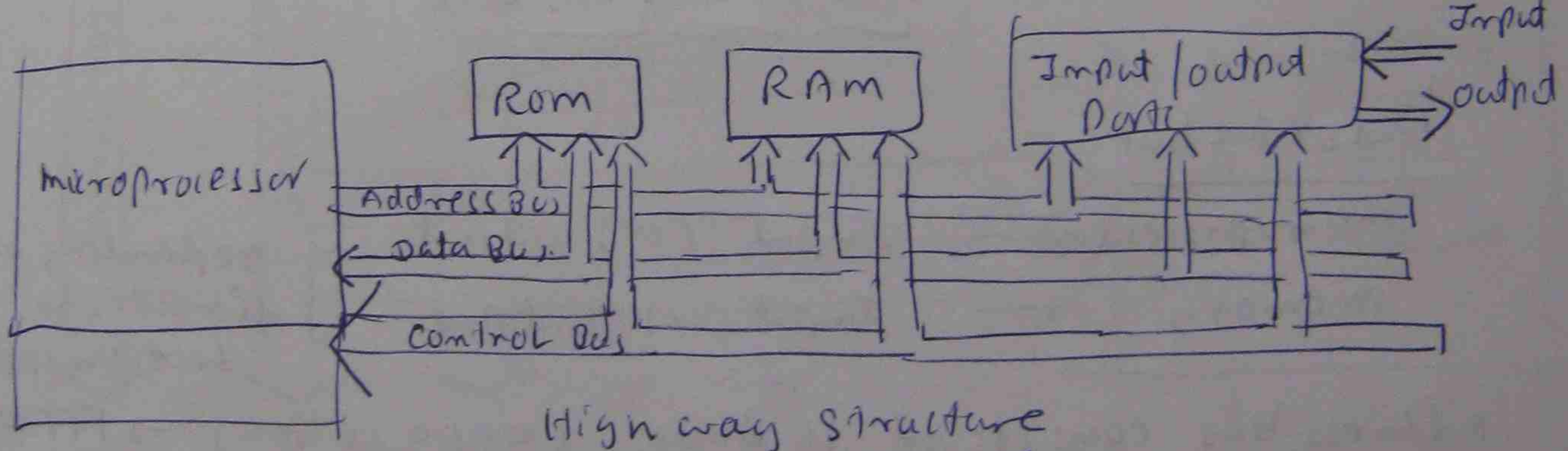
- Exposure to intensive altra violet light
- Applying voltage to ~~chip~~ specific pins on the integrated circuit.

Highway Structure

Data bus - carry the data associated with a memory or input/output transfer & typically 8 bit wide

Address bus - To specify the memory location (or) Input output port involved in a transfer

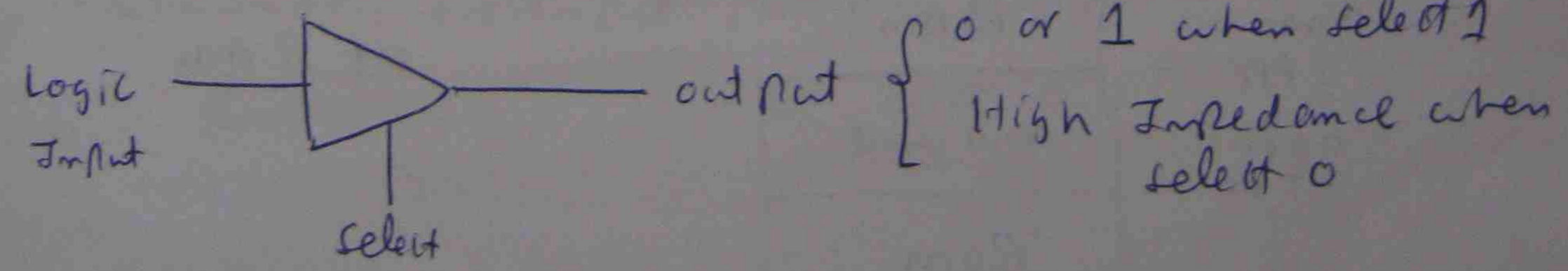
control bus - made up of the various control lines generated by the microprocessor and other system components to synchronise transfer.



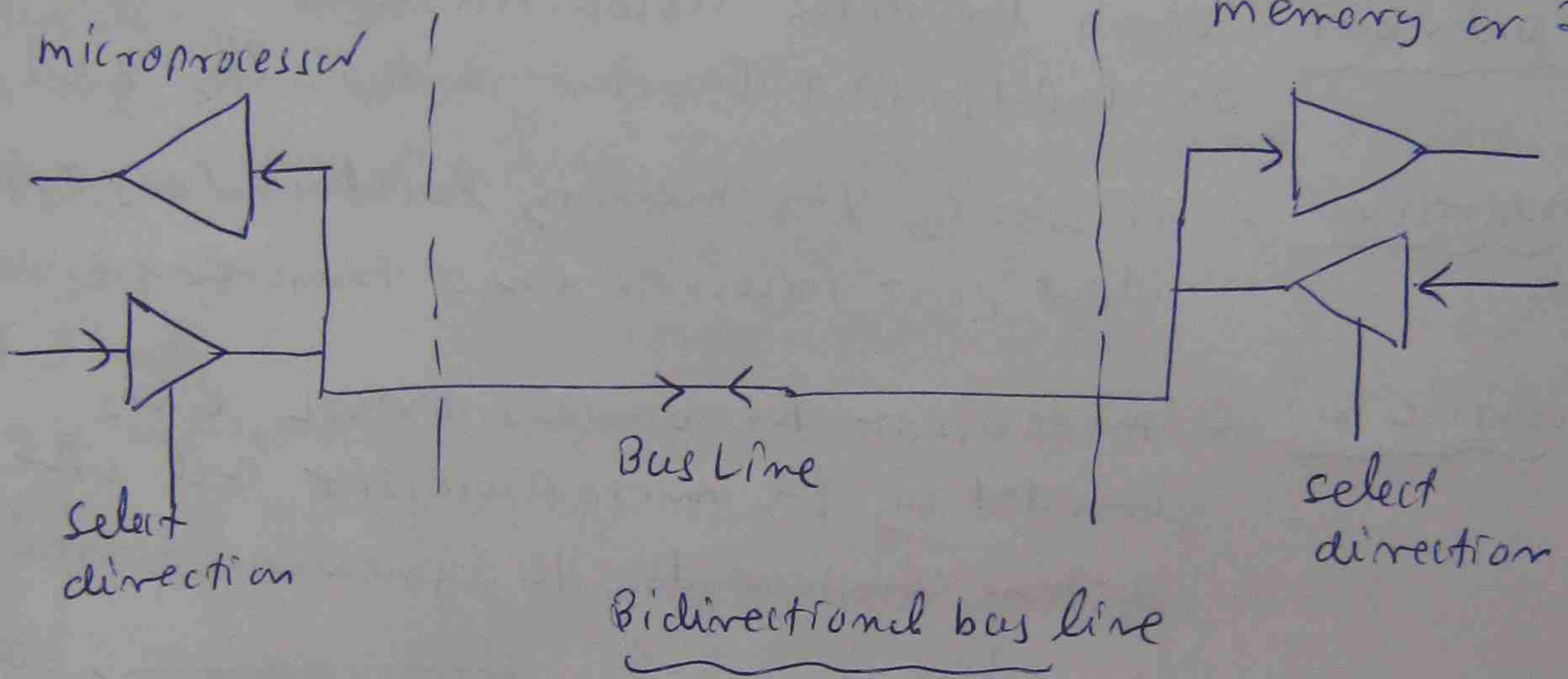
Highway Structure

Data bus - Bi directional data flow

The processor can write data on to the bus lines to be read by a memory device (or) it can read data from the bus presented by such a device.



It becomes possible to make a single pin a logic input and output by incorporating within the microprocessor logic output gates, a third output state in addition to normal 0 and 1 signals. This third state is a high impedance condition where the output is effectively switched off.

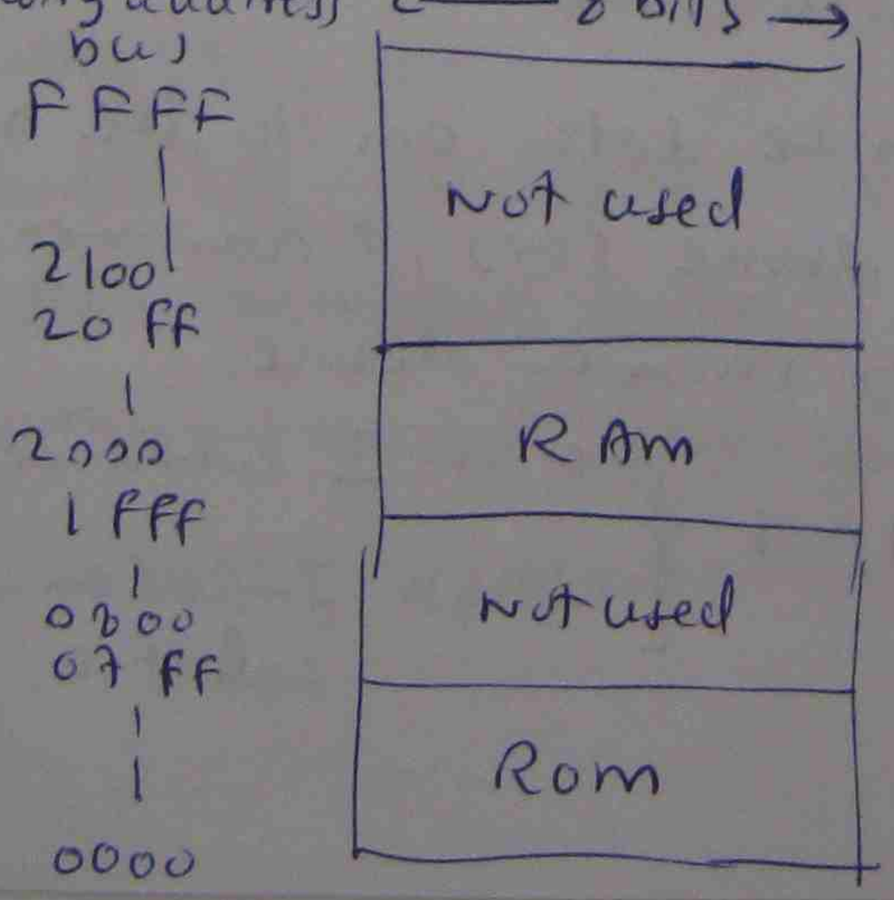


End of the bus

microprocessor — Input (or) output
memory — Input (or) output
} depending on direction control

Address bus consists of 16 lines. (0000)(Hex) → FFFF (Hex)

memory map



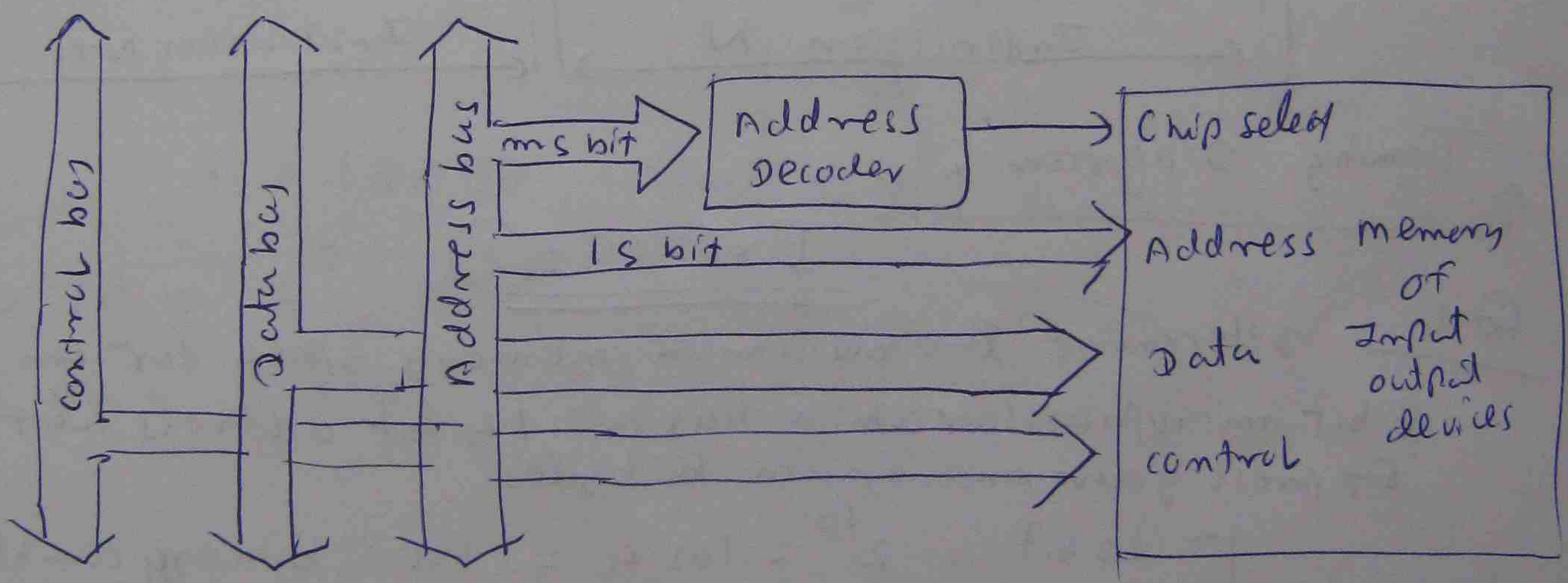
2K => 0000 → 07FF bytes of Rom

256 (2000 → 20FF) bytes of RAM

Address decoding

Since there are a number of devices connected to the computer highways - Rom and Ram chips, input/output devices etc - it is necessary to ensure that only the device intended for the data transfer responds when a request is made by the microprocessor.

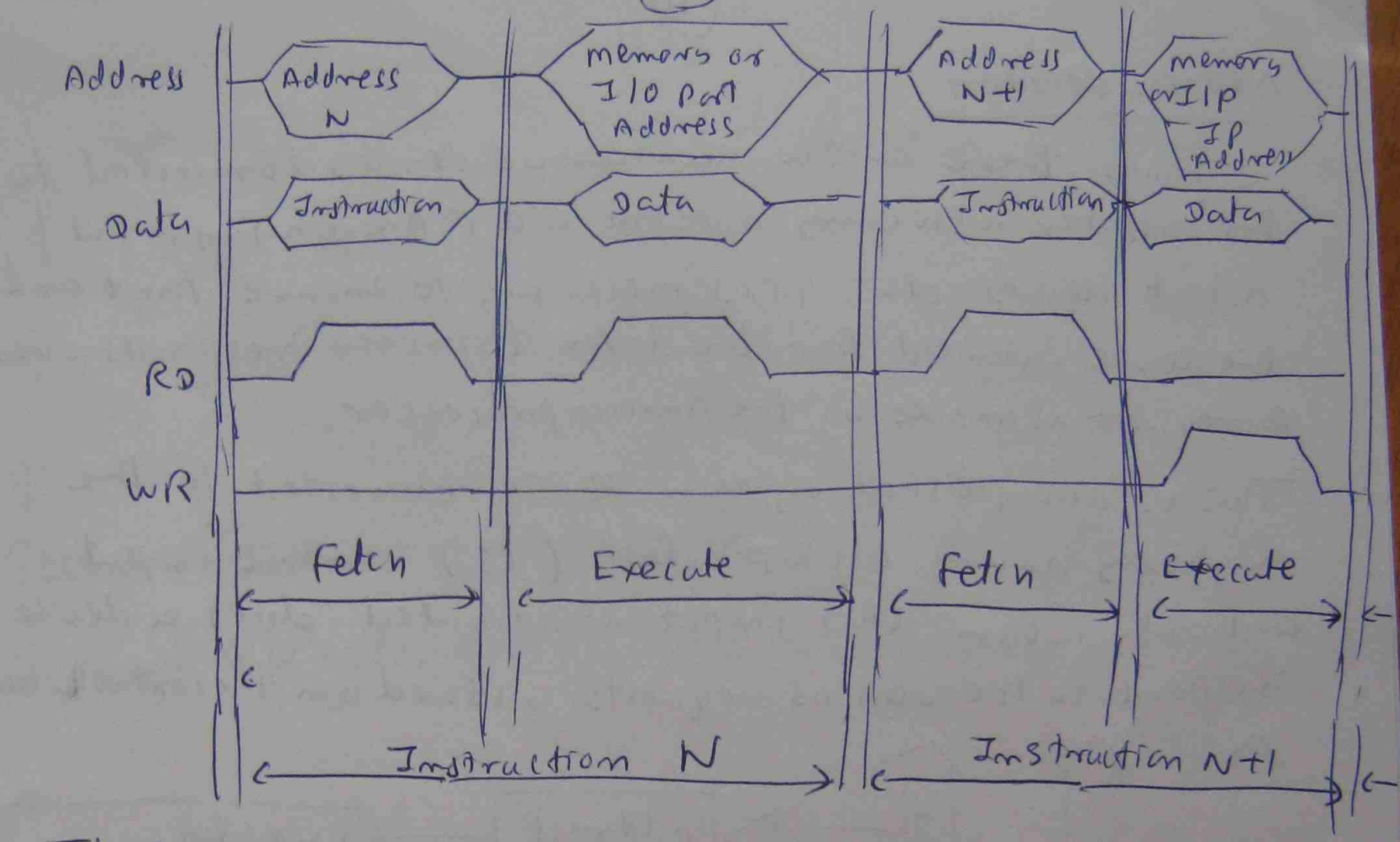
This is accomplished by each device connected to the highway having a chip select (CS) control input, and only when this input is activated does a device respond to the various requests issued on the control bus



Bus control

The control bus incorporates the timing signals which are generated by the microprocessor to synchronise information transfers between the microprocessor & a memory or input/output port

Read - RD, write WR



Timing Diagram

EXERCISE (2)

Q1 Determine the maximum memory space for an 8 bit microprocessor which has a 14 bit address word Express your answer in K bytes.

14 bit → 10 bit = 2^{10} = 1024 = 1K binary combinations
or,
4 bit = 2^4 = 16 binary combinations
(0000 → 1111)

2 | 16
2 | 8
4

∴ 14 bit = 1K × 16 = 16K bytes of memory

Q2 A microcomputer system requires 4K bytes of Rom and 256 bytes of RAM. Determine the start and end addresses of each memory block if the two memories are to occupy continuous blocks of memory starting at address (0000) hex. Express your answer in hex notation.

1K = 2^{10} = 1024 locations. 10 bit

4K = 2^{12} = 4096 locations 12 bit

256 = 2^8 locations 8 bit

Rom = $2^{10} \simeq$ 1000 ~~= 0000 ⟶ 1000 Rom~~

1000 × 4 = 4 Kbytes Rom = 0000 ← Zero byte

↓

1000

∴ Rom 0000 ↓

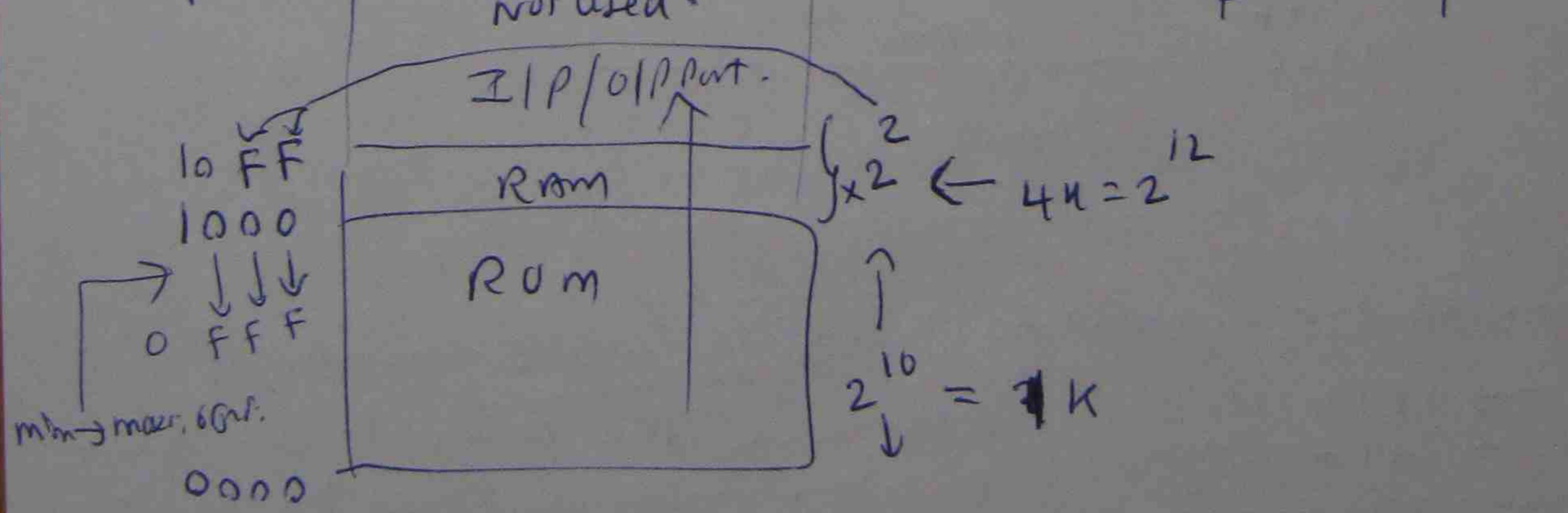
4 × 1000 4 Kbyte. Base line ⟶ 4 × 1000

$1K = 2^{10}$

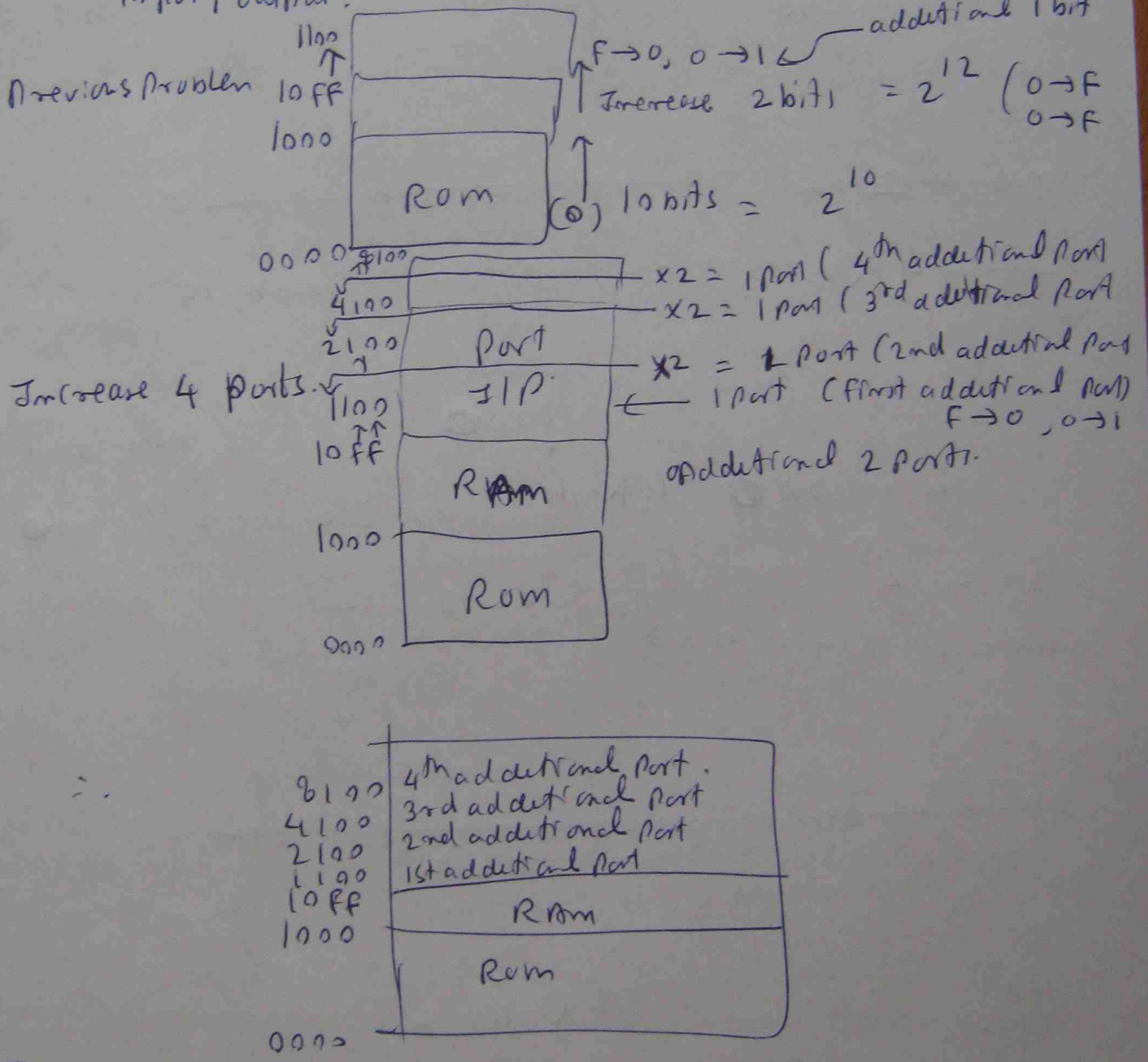
$4K = 4 \times 1K = 4 \times 2^{10} = 2^2 \times 2^{10} = 2^{12}$

8 bit = 0000 0000 ⟶ 1111 1111

FFFF 8 bit ← 256 byte F F



Q3 If the microcomputer in above question requires four additional input / output ports. Define suitable addresses for the ports assuming memory mapped Input / output.



∴

Q4 A micro computer system has the following memory map

0000 → 0FFF ROM
2000 → 21FF RAM
4000 → 400F I/O

Determine the amount of ROM & RAM memory and the number of I/O ports in the system.

4bit → 1111

0000 → 0FFF ← 1111 = 4bit, 1111 = 4bit

2000 → 21FF ← 1111 4bit, 1111 = 4bit, 0001 = 1bit

Total = 12 bit

$2^{12} \simeq 4k$

9 bit = 2^9 = 512 byte byte

4000 → 400F

1111 = 4bit

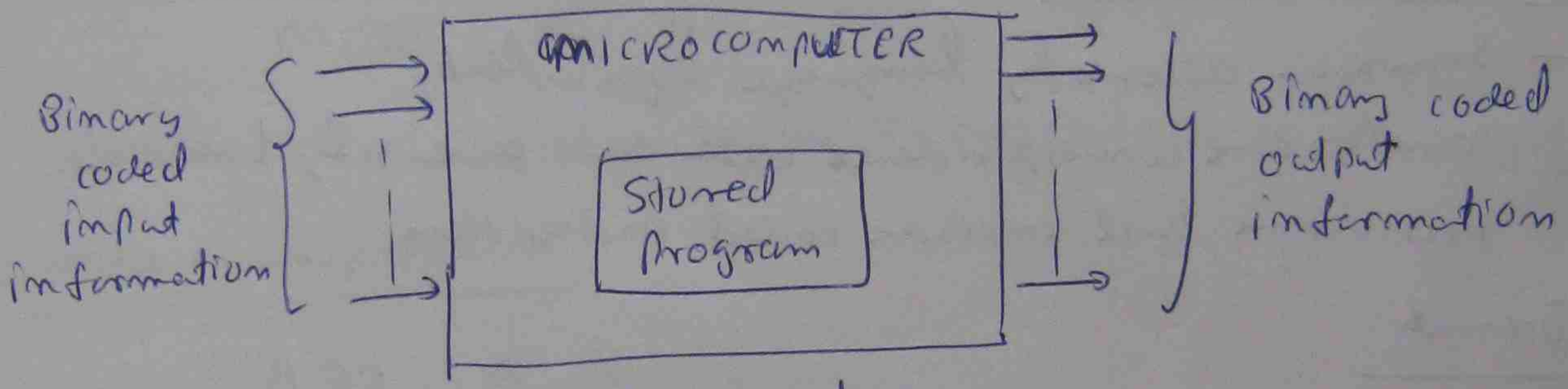
2^4 = 16 port

Week (3)

Introduction to Programming and data transfer

microcomputer

- Read binary coded information from its input ports
- manipulate this information according to a program stored within its memory
- subsequently produce output information at its output ports



※ It is necessary to become familiar with the different types of machine instructions which a typical microprocessor executes and to investigate their effect on the total system.

Intel 8085

A	
B	C
D	E
H	L
F	Im
Stack Pointer SP	
Program counter PC	

A = 8 bit arithmetic register (accumulator)

BCDE - 4 bit general purpose registers.

F = 8 bit flag register controlled by ALU

Im - 8 bit interrupt control register

HL - two 8 bit registers used to form a 16 bit memory pointer.

SP – Stack Pointer register.
16 bit memory address which displays points to the top of a system stack.

PC – Program counter register which contains a 16 bit memory address which points to the next instruction to be executed.

Assembly Language

– Symbolic assembly language equivalent.
– one to one correspondence between assembly language instruction and machine coded instruction.

Format

LABEL : OPERATION MNEMONIC , OPERANDS COMMENTS.

LABEL – Operational symbolic address for the instruction

OPERATION MNEMONIC – Tell the programmer the specific operation to be performed

OPERANDS – A value on which this operation is to be carried out
(OR)
The memory locations where the value can be found.

COMMENTS – optional comments.
To facilitate understanding & enhance the readability of the complete program
– Does not influence the machine code resulting from the assembly instruction

Classification of instructions

- Data transfer
- Data manipulation
- Transfer of control
- Input / output
- machine control

1 Data Transfer

MOV A, B

Results in B register being transfered to A register

2 Data manipulation

ADD A, B

A register (accumulator) containing the sum of its previous contents and the contents of the B register.

3 Transfer of control

- unconditional
- Subroutine call
- conditional (flag dependent) Jump instructions.
- Return instruction.

Jmp LABEL1

- microprocessor breaks its normal mode of sequential instruction execution.
- Jump unconditionally to symbolic address LABEL1 for next instruction to be executed.

4 Input / output

move data between the various input / output ports of the system and an internal processor register - usually A - register.

OUT 05

The content of the A register being transferred to output port 05 (hex).

5 machine control

Instructions in the machine control group affect the state (or) mode of operation of the processor itself.

Interrupt, enable, disable, processor halt, no operation instructions.

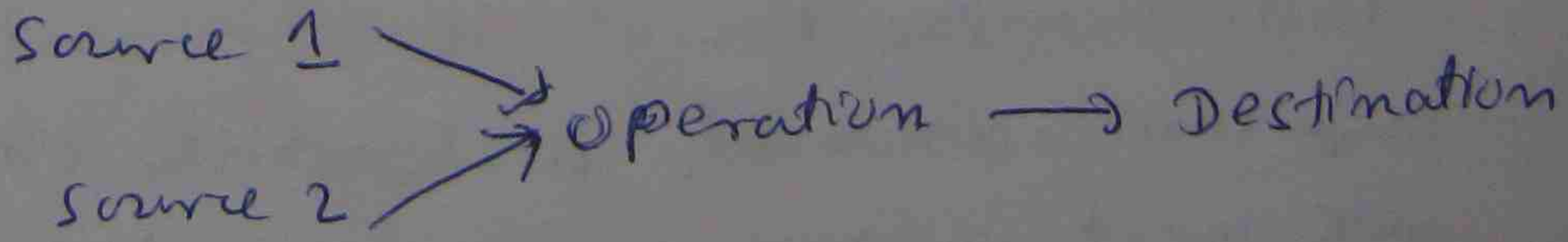
Operand addressing mode

machine instruction

2 addresses

Specify the location of the values to be manipulated (source addresses)

The third to specify the location where the result is to be stored (destination address)



Instruction addressing mode

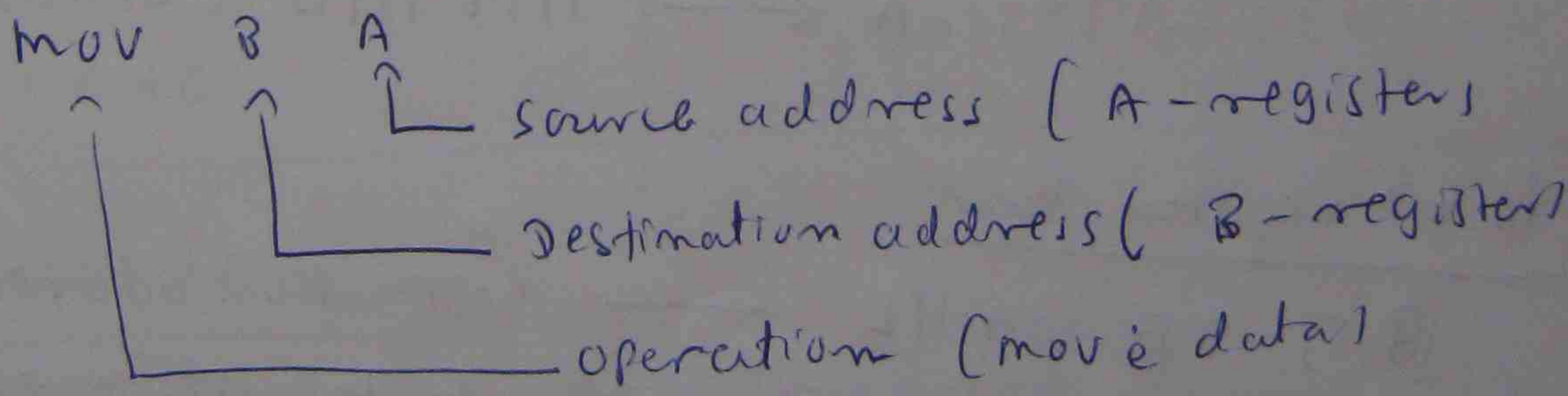
4 main types of addressing modes

Register Addressing
Immediate Addressing
} These are used primarily for data transfer and manipulation instructions which involve only the internal processor registers.

Direct (Extended) Addressing
Register Indirect Addressing
} These are used primarily for data transfer and manipulation instructions which involve the system memory.

Register Addressing

- move data between the internal processor registers.
- The instruction source and destination addresses specify which of these registers are involved in the transfer.



This results in the contents of the A register being transferred to the B-register. The contents of the A-register remain unchanged. (B) ← (A)

Data transfer involving 16 bit register pairs

XCHG

(DE) ⟷ (HL)

The contents of register pair DE being exchanged with the contents of register pair HL.

Immediate Addressing

The source address does not specify a register (or) memory location but instead the actual source data is contained with in the instruction itself. and is therefore immediately available.

MVI A FE(hex)

- FE(hex) — source data
- A — Destination address
- MVI — operation.

The data value FE(hex) being transferred to the A register

(A) ← FE(hex)

(or)

A ← 1111 1110 (binary)

16 bit register pair

B, D, H or SP —— Destination address.

These instructions require two byte of immediate data

LXI H, 8020 { 16 bit register pair HL being loaded with immediate data 8020 (hex)

(H)(L) ← 8020 (hex)

LXI D, E627 { (D)(E) ← E627 (hex)

REGISTER DATA TRANSFER

Pb(1)

1 – The Program loads a value into A register using immediate addressing

– Then loads this value into two further registers

2 B and C using ~~two~~ register addressing.

– Finally Register Pairs HL and DE are loaded

3 a/b with 802D & E627 using immediate addressing. ~~Then their~~

4 – Then their contents are exchanged using register addressing.

Assembly Instructions.

MNEMONIC	OP1	OP2	comments
1 MVI	A	FE	(A) ← FE (hex)
2 MOV	B	A	(B) ← (A)
2 MOV	C	B	(C) ← B
3a LXI	H	802D	(H)(L) ← 802D (hex)
3b LXI	D	E627	(D)(E) ← E627
4 XCHG			(D)(E) ⟷ (H)(L)

Direct Addressing

An operand may be either read from (or) written to a memory location, the address of which is specified in the instruction itself.

Load

Ex LDA 20EA (OR) (A) ← (20EA)

Register A loaded with content 20EA (hex)

~~A loaded~~ (Loaded) ← (content)

Store

Ex STA 20f2 (OR) (20f2) ← (A)

Register A being stored in memory location

20 f 2 (hex)

(Memory location) ← (Register)

Hold

Simultaneously

LOAD

Ex

LHLD 20A2 (OR) (L) ← (20A2)
(H) ← (20A3)

Register

content

~~Register~~ Register Pair, Pair ~~I~~

LOAD HOLD

Register 2
~~Register~~ content
~~pair 2~~

Register Pair H and L is frequently used to hold a combined 16 bit memory address.

This address for it address increase 1 for reg: L

~~address~~

Ex Store simultaneously

SHLD 20AF ← Store

20A~~F~~2

Register

content

~~20AF~~ content of Register ~~(H)~~ is stored in 20A~~F~~2

(L) is stored in 20A3

(L) ← (20A2)
(H) ← (20A3)

Ex SHLD 20AF
(20AF) ← (L)
(20B0) ← (H)

DIRECT ADDRESSING (23)

Pb(2)

- Immediate and direct addressing of register A at FF and EE

- ~~Data 20A2 and another consecutive memory location~~

  Data in A is stored at address 20A2 and another consecutive memory location.

- Load register pair HL with these data

- Store the same data loaded at HL into a two consequent memory locations.

Immediate Address(L) Store(L) Immediate address Store(2)

	Assembly Instructions		Actions
1(a)	MVI A, FF		(A) ← FF(hex)
2(a)	~~MVI~~ STA 20A2		(20A2) ← (A)
1(b)	MVI A, EE		(A) ← EE(hex)
2(b)	STA 20A3		(20A3) ← (A)
3	LHLD 20A2	Load	(L) ← (20A2) FF (hex) (H) ← (20A3) EE (hex)
4	SHLD 20A4	Store	~~(L) (H) (20~~ (20A4) ← (L) (20A5) ← (H)

Register Indirect Addressing

Direct Addressing	Indirect addressing
	more efficient method
– Only the A register may be used to store or load a value to and from memory	– Data may be transferred between any of the Processor registers and the system memory
– If a value were to be stored in a memory location, the B register	– Operand is either read from (or) written to memory location
(1) Transfer contentents from B to A	– Instruction does not contain actual memory address
(2) Then store operation is performed	– Operand is either read from (or) written to the memory location, the address of which is currently stored in the register pair HL

Ex mov A M

A register being loaded with the contents of the memory location whose address is specified in register H and L (A) ← (H)(L)

Ex mvI M, FF

The value FF(hex) being stored in the memory location whose address is in register H & L ((H)(L)) ← FF(hex)

Indirect Addressing

Ph (3)

1 The memory address of A which contains data value AA is 2040 and it is loaded into register H and L by indirect addressing mode.

LXI H

2 Then a value is moved to memory location using register indirect addressing.

MVI M, AA

3 The value is loaded into two further registers B & C using register indirect addressing

MOV B, M

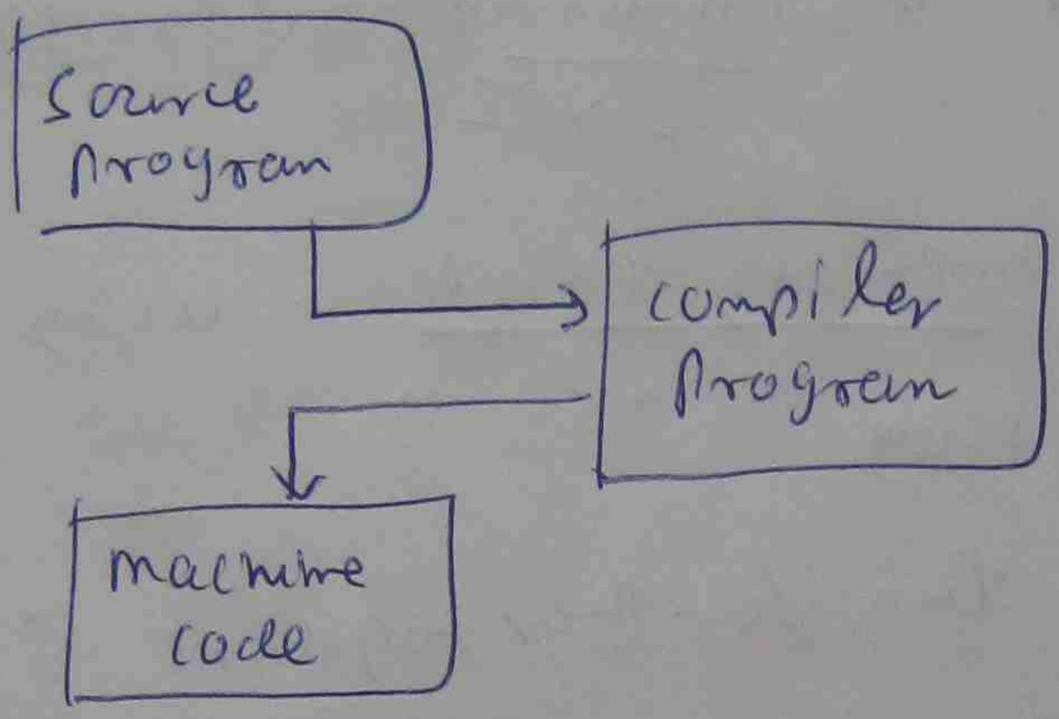
MOV C, M

	Assembly Instruction	Action
1	LXI H, 2040	(L) ← 40 (hex) (H) ← 20 (hex)
2	MVI M, AA	(H)(L) ← AA (hex) (OR) 2040 ← AA (hex)
3	MOV B, M MOV C, M	(B) ← (2040) (OR) (B) ← AA (hex) (C) ← (2040) (OR) (C) ← AA (hex)

The Assembly Process

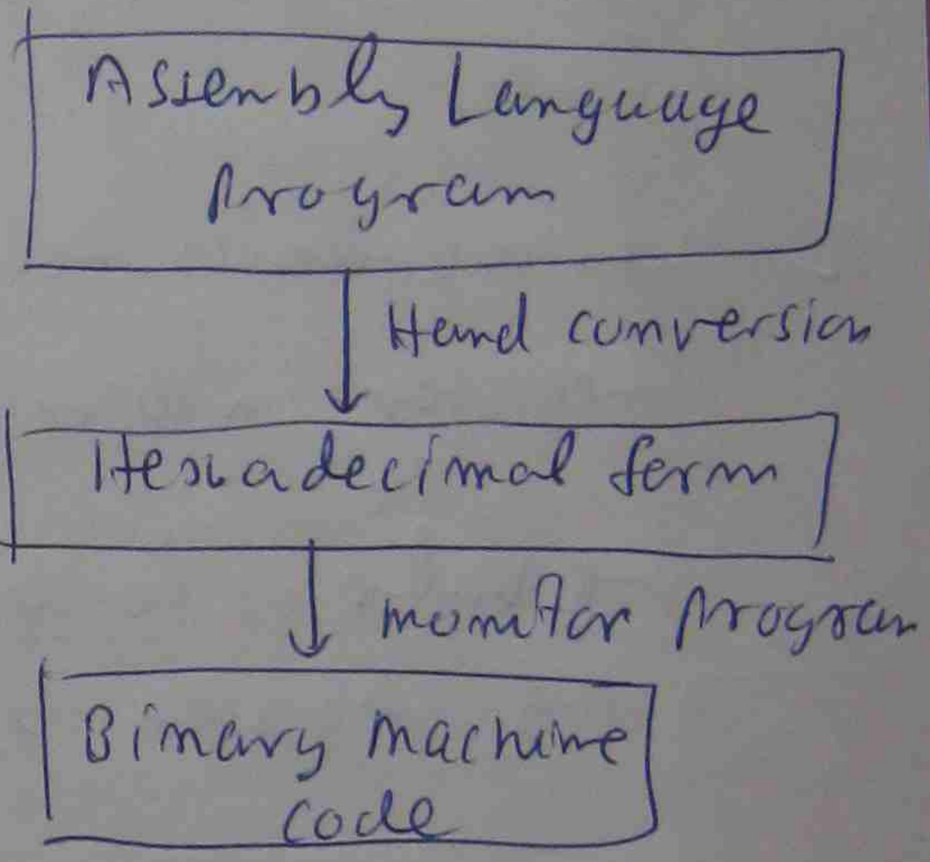
compiler

The compilation process



Assembler

The hand assembly process



8085 Instruction set

MOV	A, A	7f
	A B	78
	A C	79
	A D	7A
	↓	
	M L	75

Appendix (1) Page 140 → 144

move Immediate

MVI	A byte	3E
	B byte	06
	m byte	36

Load Immediate

(Reg, Pair)

LXI	B dble	01
	D dble	11
	H dble	21
	SP dble	31

Load / Store A direct

LDA	addr	3A
STA	addr	32

Load / Store A indirect

LDAX B	0A	STAX B	02
LDAX D	1A	STAX D	12

Load / Store HL direct

LHLD	addr	2A
SHLD	addr	22

Exchange HL/DE

XCHG	EB

Ex MOV A B

(A) ← (B) requires 7, 8

Ex MVI A, FE

(A) ← FE(hex) requires 3 E, FE

operation

Immediate data

Ex STA 2052

32 operation

most significant byte of memory address

Least significant byte of memory address

Pb write the hand coding for the following operation

① the Program load, a value into A register using Immediate addressing. at the address 2000

② Then loads this value into two further registers B and C using register addressing.

3 a/b Finally register pair HL and DE are loaded with 3020 & E027 using immediate addressing

④ then Their contents are exchanged using register addressing.

(28)

LINE NO.	Memory		Assembly Instruction			Comment
	Address	content	MNEMONIC	OP1	OP2	
~~2000~~ 1	2000	3E	MVI	A	FE	(A)←FE hex
	2001	FE	3E for 3085			
	2002	47	MOV	B	A	(B)←(A)
	2003	4F	MOV	C	B	(C)←(B)
		Although data finally reached	MOV CB is 48	A	MOV C,A	EA 4F
	2004	21 for LXI H	LXI	H	202D	(H)(L)← 202D
	2005	2D				
	2006	80				
	2007	21	LXI	D	E027	(D)(E)← E027
	2008	27				
	2009	E0				
	200A	EB	XCHG			(D)(E)↔ (H)(L)

Nvt 2010

200(A) ← 10

Week (4)

Data manipulation

Microprocessor - Represent data within an number of different form

Typical Arithmetic instruction.

Data representation

In general, numbers may be represented in unsigned binary, signed binary or binary coded decimal (BCD) form.

Unsigned binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	= weightings
0	0	1	0	1	0	1	1	= 43 (Decimal)
0	1	0	0	0	1	1	0	= 70
1	0	1	0	0	0	0	1	= 161
1	1	0	0	1	1	0	0	= 204

Signed binary

26 =

2	26	0
2	13	1
2	6	0
2	3	1
	1	

= 11010

for 8 bit

0 00011010 = +26

7 bit

100 =

2	100	0
2	50	0
2	25	1
2	12	0
2	6	0
2	3	1
	1	

= 1100100

for 8 bit

1 1100100 = −100

This form is not possible to perform Arithmetic operation

Two's complement form of representation

S	6 5 4 3 2 1 0

S=0 for positive number and zero

S=1 for negative numbers.

Ex1 +15

2	15	1
2	7	1
2	3	1
	1	

1111

for 8 digit = 00001111

Invert) 11110000

+1) Increase 1

−15 = 11110001

Ex2 +89 =

2	89	1
2	44	0
2	22	0
2	11	1
2	5	1
2	2	0
	1	

1011001

FOR 8 bit = 01011001

Invert) 10100110

+1)

−89 = 10100111

Table Two's complement Representation

Decimal Number	Two's complement
+127	0 1111111
⋮	
+3	0 0000011
+2	0 0000010
+1	0 0000001
0	0 0000000
−1	1 1111111
−2	1 1111110

$-3 \longrightarrow$ 1 1 1 1 1 1 0 1

1 0 0 0 0 0 0 1

-127

-128 1 0 0 0 0 0 0 0

Binary coded Decimal (BCD)

It the input data is from a decimal key pad and the subsequent output data drives a decimal display, use decimal number representation and arithmetic → provide instructions for performing arithmetic on binary coded decimal (BCD) number.

BCD representation is a subset of the hexadecimal system

BCD code

decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

8 bit binary code may be used to store two BCD numbers.

86

1000 0110

(decimal)

$\therefore$ 86 = 1000 0110

51 (decimal)

0101 0001

$\therefore$ 51 = 0101 0001

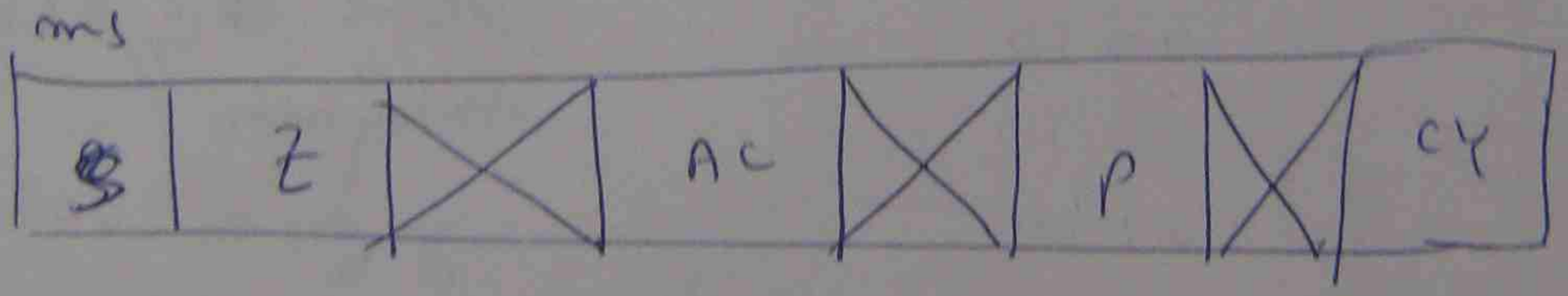
Arithmetic instructions

Arithmetic instructions → add
→ subtract
→ increment
→ decrement

8085 The instructions always involve A register and either another processor register (or) a memory location.

— A microprocessor contains a number of different forms of these instructions so that data can be manipulated in the selected manner.

Flag - Status (or) condition bit
- which are either set (or) reset depending on the particular arithmetic instruction being carried out and the programmer is able to use and interpret these flags to manipulate data in the selected way



F register of 8085

S =	Sign flag	If is set when the result of an arithmetic operation is negative
Z	Zero flag	- The flag is set if the result of an arithmetic operation on the register is zero, otherwise it is reset. - This is used with transfer of control function
AC	Auxiliary Carry	BCD number representation is being used If is set when the result of an arithmetic operation produces a carry out from the least significant half of the A-register

P	Parity flag	This is used with logical operation. The flag is set if the result of a logical operation (AND, OR, XOR) produces an even number of 1's
CY	Carry Flag	CY is set after an ADD instruction if a carry out was generated from the A-register

Addition of two bits

Bit 1	Bit 2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

1st 2nd 1st 1st Bit Bit
0 + 1

A = 1001 1010 = 154 decimal

B = 0101 0111 = 87 decimal

241 decimal

100 ← carry in

001 ← Sum

Addition of two bits & a carry in

	Bit 1	Bit 2	Carry in	Sum	Carry out
(1) 1st	0	0	0	0	0
(2) 2nd	0	0	1	1	0
(3) 1st	0	1	0	1	0
(4) 2nd	0	1	1	0	1
(5) 1st	1	0	0	1	0
(6) 2nd	1	0	1	0	1
(7) 1st	1	1	0	0	1
(8) 2nd	1	1	1	1	1

1001 1010

\+ 0001 0010

0101 0111

1100 1101

A = 1 0 0 1 1 0 1 0 = 154 decimal

B = 0 1 0 1 0 1 1 1 = 87 decimal

1 0 0 1 0 0 sum

0 0 1 0 0 1 0 carry

A :- 1 0 0 1 1 0 1 0 = 154 decimal

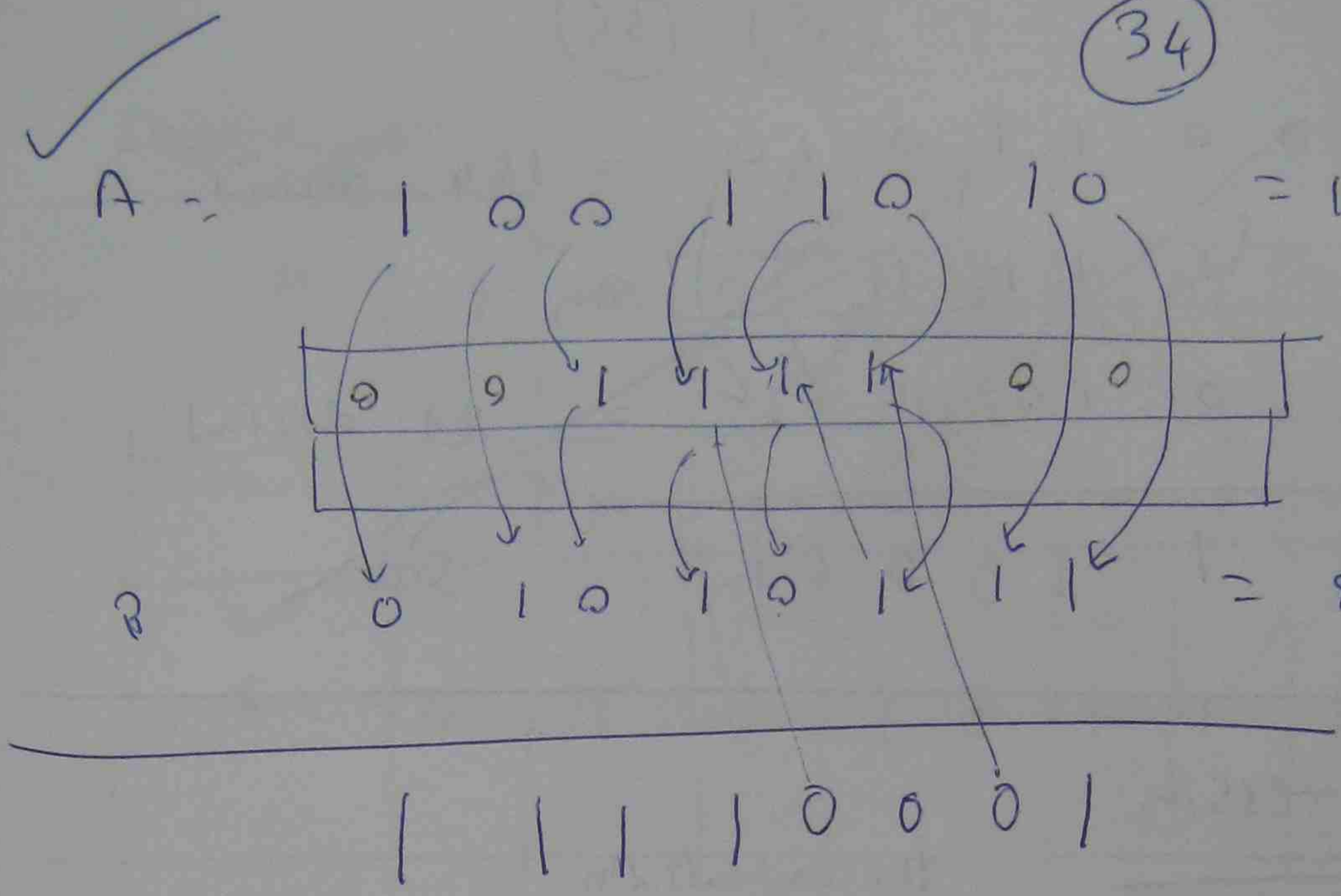
B :- 0 1 0 1 0 1 1 1 = 87

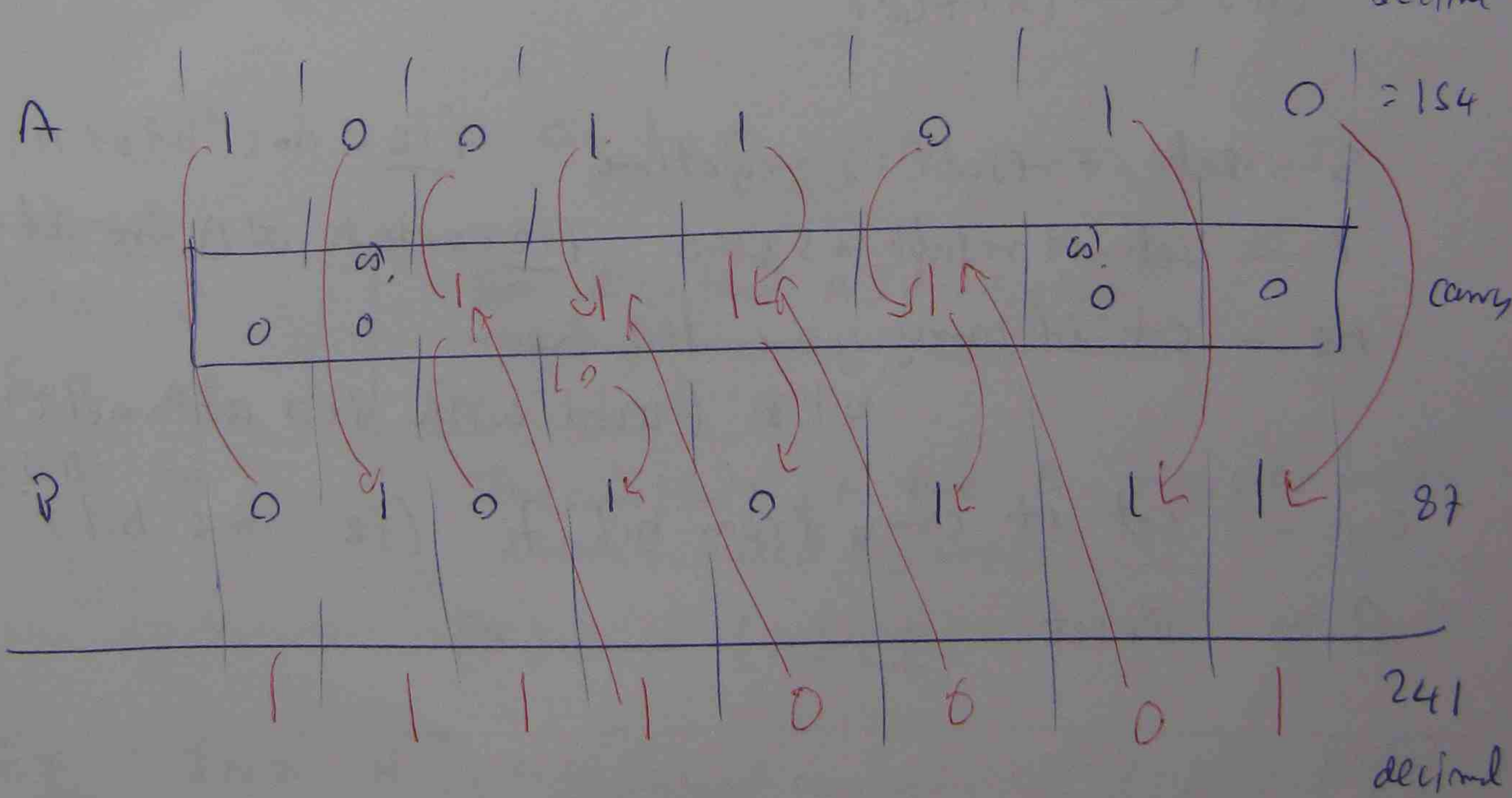
1 1 1 1 0 0 0 1 = 241 decimal.

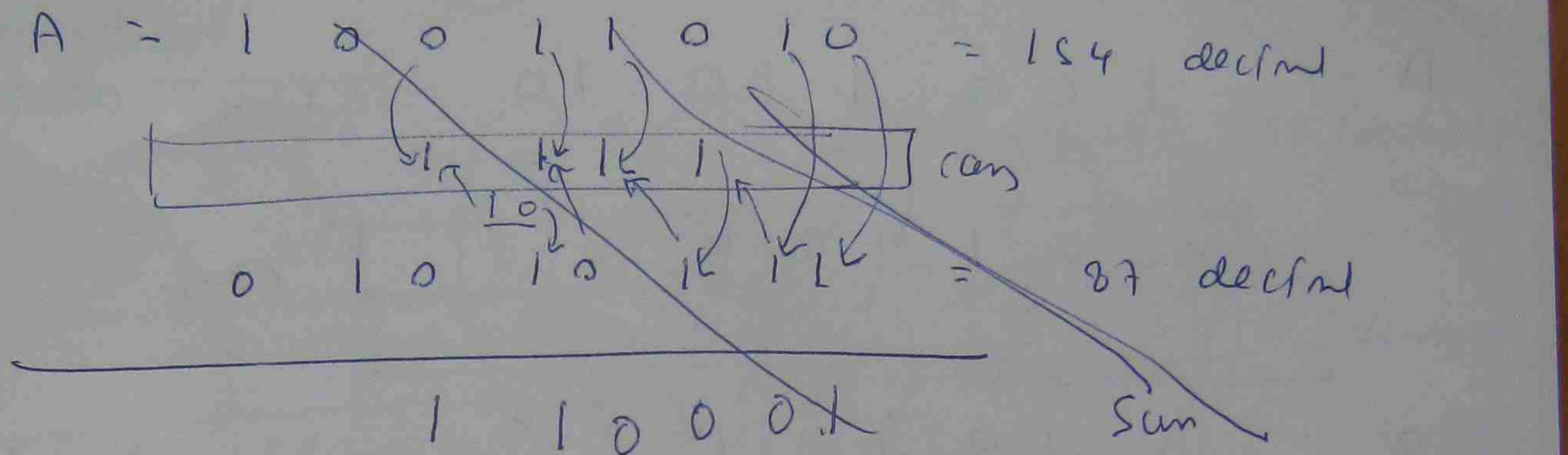
1 0 0 1 1 0 1 0

0 1 0 1 0 1 1 1

1 0 0


A = 1 0 0 1 1 0 1 0 = 154 decimal
0 0 1 1 1 0 0 carry.
B 0 1 0 1 0 1 1 1 = 87 decimal
1 1 1 1 0 0 0 1


A 1 0 0 1 1 0 1 0 = 154
0 0 0 0 carry
B 0 1 0 1 0 1 1 1 87
1 1 1 1 0 0 0 1 241
decimal



Register Addressing

Ex ADD B — the contents in B register is being added to current contents of the A register

(A) ← (A) + (B)

S = set if result is negative (ie ms bit of A is 1
Z - set if result is zero (ie contents of A are all 0s)
AC - set if carry generated from bit 3 (used with BCD arithmetic)
Cy - set if carry from bit 7 (ie ms bit)
P - Reset

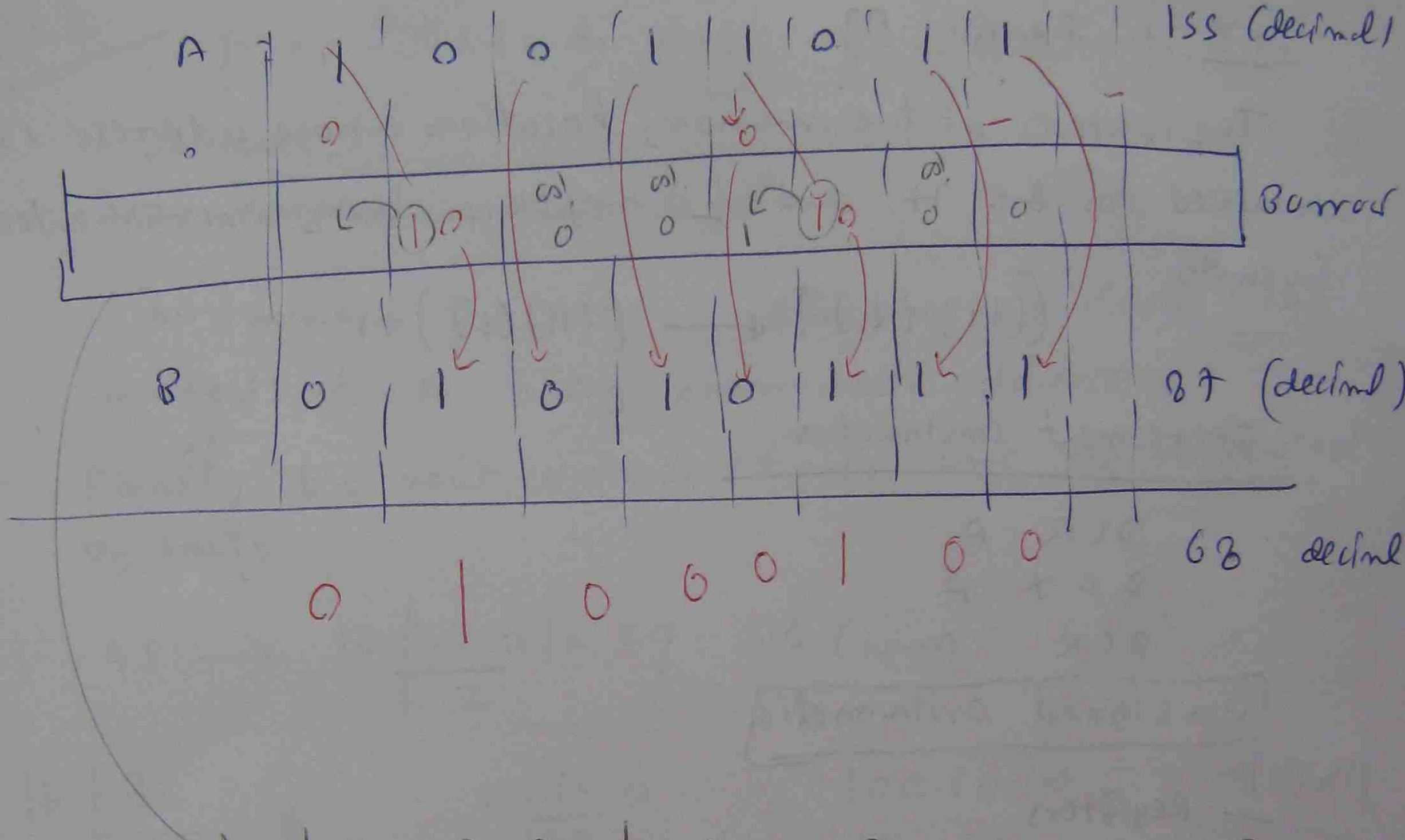
Ex ADI 0F

The immediate data 0F (hex) being added to the current contents of A register

(A) ← (A) + 0F

all Flags are affected.

Substract



Substract

SUB	B
SUI	0f
SUB	m

~~Re~~ Register Addressing

EX INR A

The contents of the A-register being incremented by units.

(A) ← (A)+1

Ex INX H combined contents of register pair H & L being incremented by units

(H)(L) ← (H)(L)+1

Register Indirect Addressing

Ex INR M

The contents of the memory location whose address is contained in the H and L registers being incremented by units.

((H)(L)) ← ((H)(L))+1

Decrement instruction

DCR A
DCX H
DCR M

Unsigned arithmetic

Pb(1) Registers

- A is loaded with immediate data 53 (hex), B is loaded with immediate data 3A.
- Their contents are added together.
- A third numeric is then substracted from the contents of A using immediate addressing. 8C (hex)
- Finally the new contents of A are decremented by units.

Assembly Instruction	Action
MVI A, 53	(A) ← 53 (hex)
MVI B, 3A	(B) ← 3A (hex)
ADD A	(A) ← (A)+(B)
SBI 5C	(A) ← (A)−8C (hex)
DCR A	(A) ← (A)−1

Signed Arithmetic

Ph 2 A is loaded with +35 decimal, B is loaded with −72 decimal.

Their contents are added together

Third numeric DB (hex) is substracted from the contents of A using immediate addressing.

Finally the new contents of A are decremented by unity.

$35 \longrightarrow 16 \left|\frac{35-32}{2}\right| = 3 \rightarrow 23$ (hex)

16	72
	64
	8

2	72	0
2	36	0
2	18	0
2	9	1
2	4	0
2	2	0
	1	

$1001000 = +72$

8 bit

0100, 1000 → Invert

1011 0111 → +1

1

$1011\ 1000 = -72$

$-72 = B8$ (hex)

Assembly Instruction	Action
MVI A, 23	(A) ← 23 (hex)
MVI B, B8	(B) ← B8 (hex)
ADD B	(A) ← (A) + (B)
SBI DB	(A) ← (A) − DB (hex)
DCR A	(A) ← (A) − 1

16 bits arithmetic

8 bit – only B
16 bit B & C → BC

8 bit only D
16 bit only D & E → DE

8 bit – only H
16 bit – H & L → HL

EX INX B

(B)(C) ← (B)(C) + 1

EX DCX D

(D)(E) ← (D)(E) − 1

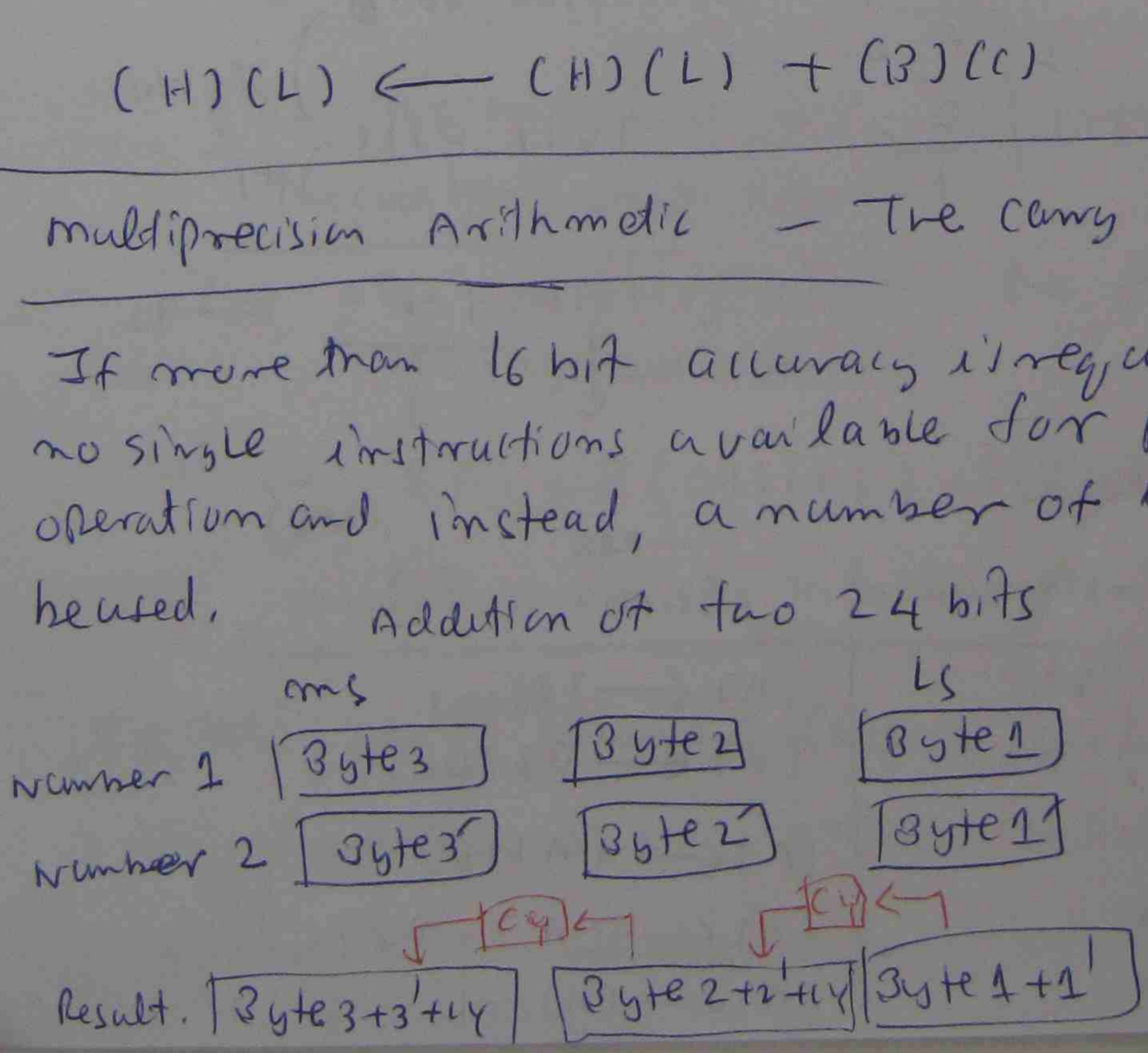
EX DAD B

16 bit contents of the register pair B,C being added to the 16 bit contents of the register pair H,L.

(H)(L) ← (H)(L) + (B)(C)

multiprecision Arithmetic – The carry flag

If more than 16 bit accuracy is required, there are ~~no~~ no single instructions available for performing arithmetic operation and instead, a number of instructions must be used. Addition of two 24 bits



~~Pb~~ EY ADC M

(1) The contents of the memory location whose address is contained in registers H & L

and

(2) The contents of CY Flag being added to the contents of A Register

(3) The result is placed in the A register and all flags are affected.

(A) ← (A) + ((H)(L)) + (CY)

EY SBB M

(1) The contents of the memory location whose address is contained in registers H & L

and

(2) The contents of CY Flag being substracted from the contents of A register

(3) The result is placed in the A register and all flags are affected.

(A) ← (A) − ((H)(L)) − (CY)

Pb 3 multiprecision Arithmetic

24 bits (3 byte) number. first is loaded in 2030, 2nd is loaded in 2031, 3rd is loaded in 2033.

< 24 bits (3 byte) number which is stored in the three consecutive memory location starting at address

2080 to 2082)

They are added together at 2083.

2 Last bit result result replaces first number.

1st → LXI H 2083 (Initialize to contain 2083

1	Load the number at 2080	LDA 2080	Add 1st pair
	Add memory	ADD M	of byte
	Stored	STA 2080	

INX H Increment HL

2	Load the number at 2081	LDA 2081	Add 2nd pairs of byte, together with carry
	Add memory contrive to add 2 carry	ADC M	
	Stored	STA 2081	

INX H Increment HL

3	Load the number at 2082	LDA 2082	Add 3rd pairs of bytes together with carry
	Add memory 2 carry	ADC M	
	Stored	STA 2082	

Replace

Add at , the result is added to 2083 which replace the first number

Program

```
LXI   H, 2083
LDA   2080
ADD   M
STA   2080
INX   H
LDA   2081
ADC   M
STA   2081
INX   H
LDA   2082
ADC   M
STA   2082
```

BCD Arithmetic

Ph (1) Add 62 BCD and 25 BCD

62 = 6 → 0110, 2 → 0010

25 = 2 → 0010, 5 → 0101

∴ 62 + 25 = 0110 0010 + 0010 0101

1000 0111

No carry beyond 4 bits

$C_y = 0$

1000 → 8, 0111 → 7 BCD

Ph (2) Add 79 BCD & 16 BCD

79 = 7 → 0111, 9 → 1001

79 + 16 = 0111 1001 + 0001 0110

1000 1111

Pb 3 Add 39 BCD & 48 BCD

39 = 3 9
↓ ↓
0011 1001

48 = 4 8
↓ ↓
0100 1000

39+48 =

```
  0011   1001
+ 0100   1000
--------------
  1000 ①0001
         ↑
         carry
```

Carry ∴ CY = 1
no bit to put ∴

1000 0001 ← normal BCD

But 39+48 = 87 ∴ Reqd. BCD = 87
↙ ↘
1000 0111

Required BCD 1000 0111 > Normal BCD 1000, 0001

∴ AC = 1

∴ CY = 1, AC = 1

To correct it 1000 0001 will neede to be added with 0110

```
∴   1000, 0001
   +      0110  ← (6)
   -----------
    1000  0111
```

to get 1000, 0111 or 87 hex

$(+6)_{hex}$ is corrected BCD Sum (0110)

∴ Decimal Adjust Accumulator DAA is utilized.

<AC Flag is set 6 is added to A register>

Ph Add the followings BCD nubers & make comection.

(a) 47 BCD + 32 BCD ~~(b) 81 BCD + 69 BCD~~

(a)

47 BCD = 0100 0111

32 BCD = + 0011 0010

7a BCD 0111 1001 ← normal BCD

↓ 1001 + ∴

1001, ~~0001~~ 0000 0000 corrected BCD

↑ The same

Required BCD ←

~~(b) 81 BCD =~~

Ph Substract the following BCD numbers and make the correction

(a) 47 BCD − 32 BCD (b) 81 BCD − 69 BCD

(a)

47 BCD = 0100 0111

32 BCD = 0011 0010

---

15 BCD = 0001 0101 ← Normal BCD

Reqd BCD: 0001, 0101 ← The same

(b) 81 BCD = 1000 0001

69 BCD = 0110 1001

---

12 BCD = 0010 1000 ← Normal BCD different

↓

0001, 0010 ← Reqd.

∴ Normal BCD ≠ Reqd. BCD different

∴ AC = 1

Correction:?

Required = 1111 0010 − 0001 0010 = 0000

Normal = 0001 1000

ADD → 1111 1010

F A

0001 0010
− 0001 1000
1111 1010

Hexi decimal numbers

4 bit binary	Hexi decimal symbol
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

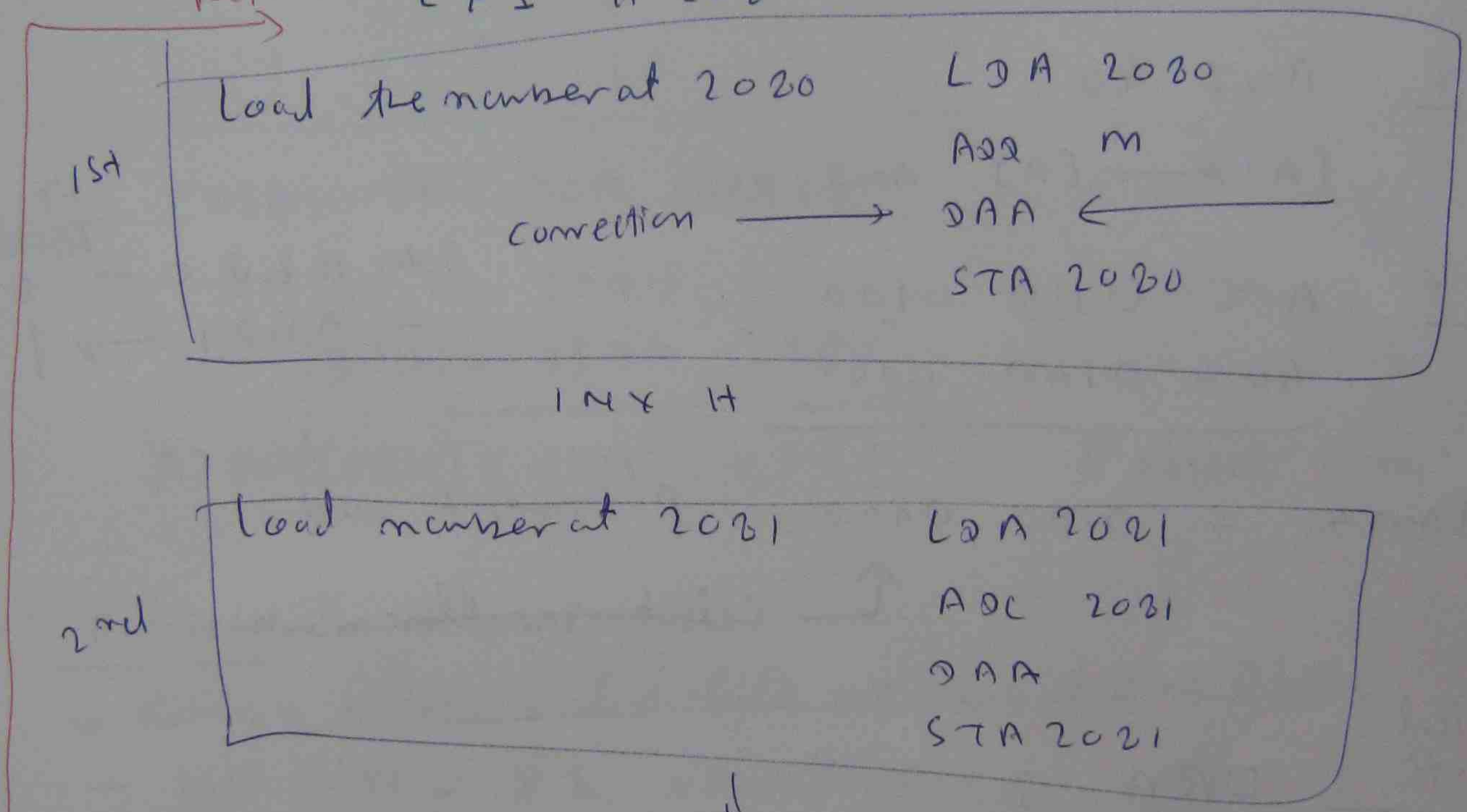
Logical operation

Ex First number 2nd number

A is loaded at 2020 and B is loaded at 2021

They are added together and correction is made

The result is put into 2082 and 2082 replaces first number



Program

```
LXI   H   2082
LDA       2020
ADD       M
DAA
INX   H
LDA       2021
ADC       2021
DAA
STA       2021
```

Carry CY	Upper hex digit b7 7-4	Aux carry AC	Lower Hex b7 3-0	correction
0	0-9	0	0-9	00
0	0-8	1	6-F	FA
1	7-F	0	0-9	A0
1	6-F	1	6-F	9A

Logical operations

Logical AND

The Logical AND instructions perform the bit by bit AND operation between the contents of the A-register and either immediate data (or) the contents of another Processor register (OR) a memory location.

Ex ANI 40

C

(A) ← (A) AND 40

Ex A = 0110 0100
40 = 0100 0000

3OM A & B 0 → Result 0
— A & B 1 → 1

(A) AND 40 = 0100 0000

↑ ~~Select smaller~~ number

P is reset odd.

Logical OR

ORA B

The contents of A register are ORed with those in B register

(M) ← (A) OR (B)

Ex (A) = 0110 0100
(B) = 1001 0101

Either A or B(1)
Result (1)

~~Select bigger~~

(A) XOR (B) = 1111 ~~0000~~ 0101

(A) + (B)

OR		
Bit 1	Bit 2	Bit(1) (or) Bit(2)
0	0	0
0	1	1
1	0	1
1	1	1

~~1110~~
~~1110 0001~~

XOR function truth table

Bit 1	Bit 2	Bit 1 XOR Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

1st

2nd

Ex (A) ← (A) XOR (H)(L)

(A) = 1001 1011

(H)(L) = 1100 1101

(A) XOR (H)(L) = 0101 0110 P is set even

Rotate

- Rotate a binary value left or right one place
  - Left shift = X 2 operation
    Right shift = /2 operation
  - useful for performing multiplication & division

Ex (A) = 1001 0110

RLC(A) = 0 0 1 0 1 0 1 1 (CY) ← 1

1 carry.

∴ CY ← 7 ← 0

↻ Rotate left

Compare

Compare two values - the contents of the A-register and either immediate data (or) the contents of a processor register or memory location.

CMP B

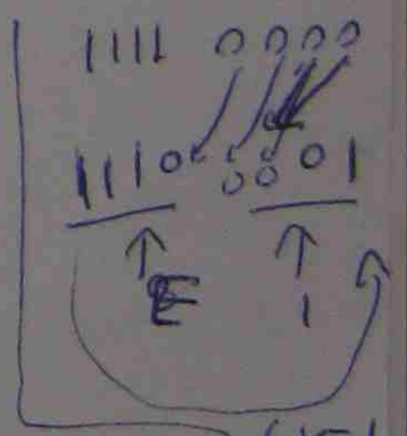
The contents of B registers are compared with the contents of A register.

Z flag is set ~~zero~~ to 1 if the contents are equal and the cy flag is set to 1 if contents of A are less than contents of B

Ex The Program Instrd to & of
First loads immediate data into registers A and B and then performs a series of logical operation

First the contents of and B are compared
2nd the ~~rotated~~ contents of A are then rotated left
3rd the new contents of A are then AND-ed with a constant 81
4th the resulting contents of A are OR-ed with the contents of B

(A) F0 = 1111,0000
(B) 0F = 0000 1111



Instrd	MVI A, F0	(A) = F0 (hex)
	MVI B, 0F	(B) = 0F (hex)
1st	CMP B	Z ← 0 CY ← 0
2nd	RLC	(A) ← E1 CY ← 1
3rd	ANI 81	(A) ← 81, CY ← 0, P ← 1
4th	ORA B	(A) ← 8F

MVI A F0 A = 1111, 0000

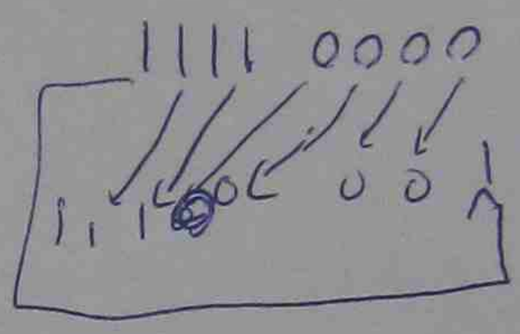
MVI B 0F B = 0000, 1111

1st CMP B A 1111, 0000, B 0000 1111

∴ A > B ∴ A = 1111, 0000

2nd RLC

A = 1111 0000



= 1110 0001

↑ CY ↑ 1

3rd ANI 81

∴ (A) CY = 1

Compare

1110 0001

81 => 1000 0001

Smaller Thesae

1000 0001 = 81

4th ORA B

A = 81 = 1000, 0001

B = 0F = 0000 1111 > OR

1000 1111

↓ ↓

8 F

Exercise Look at exercise questions 4.1 to 4.8
check the answer provided for exercise 4.1 to 4.8
Determine the ways to approach the solution.
Take exercise with microprocessor Training software

week(5)

Transfer of control

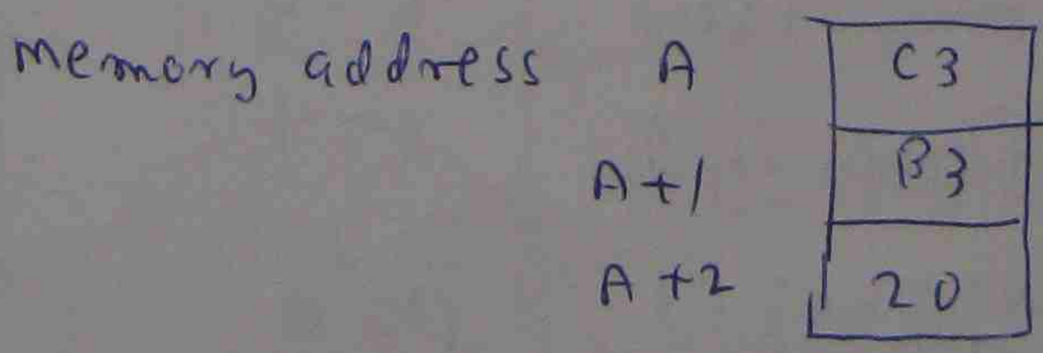
- To achieve the ability to transfer control, branch or to an instruction that is not in sequential order, it is achieved with instructions from the transfer of control group.
- All these instructions act on the program counter.
- It is possible to execute a block of instructions many times over with the number of times determined either by Program data or the state of a Processor flag.

Jump instructions

- Break normal sequential execution
- Branch to a different part of the Program.
- Loading the address of the next out of sequence instruction in to the Program counter.
- Forcing the Processor to fetch the contents of this new location for its next instruction
- The new address is specified in the instruction.

JMP 2083

<an conditional jump to memory location 2083 for the next instruction>.



unconditional Jump

Jmp LAB 1

↑ to indicate the destination address of jump instruction

- use label to indicate jump during assembly language development

Absolute address	Instruction
2003	Jmp LAB 1
2083	LAB1: Destination Instruction

Symbolic Addressing

conditional Jump

JNZ LAB1

Jump if zero flag not set to LAB 1.

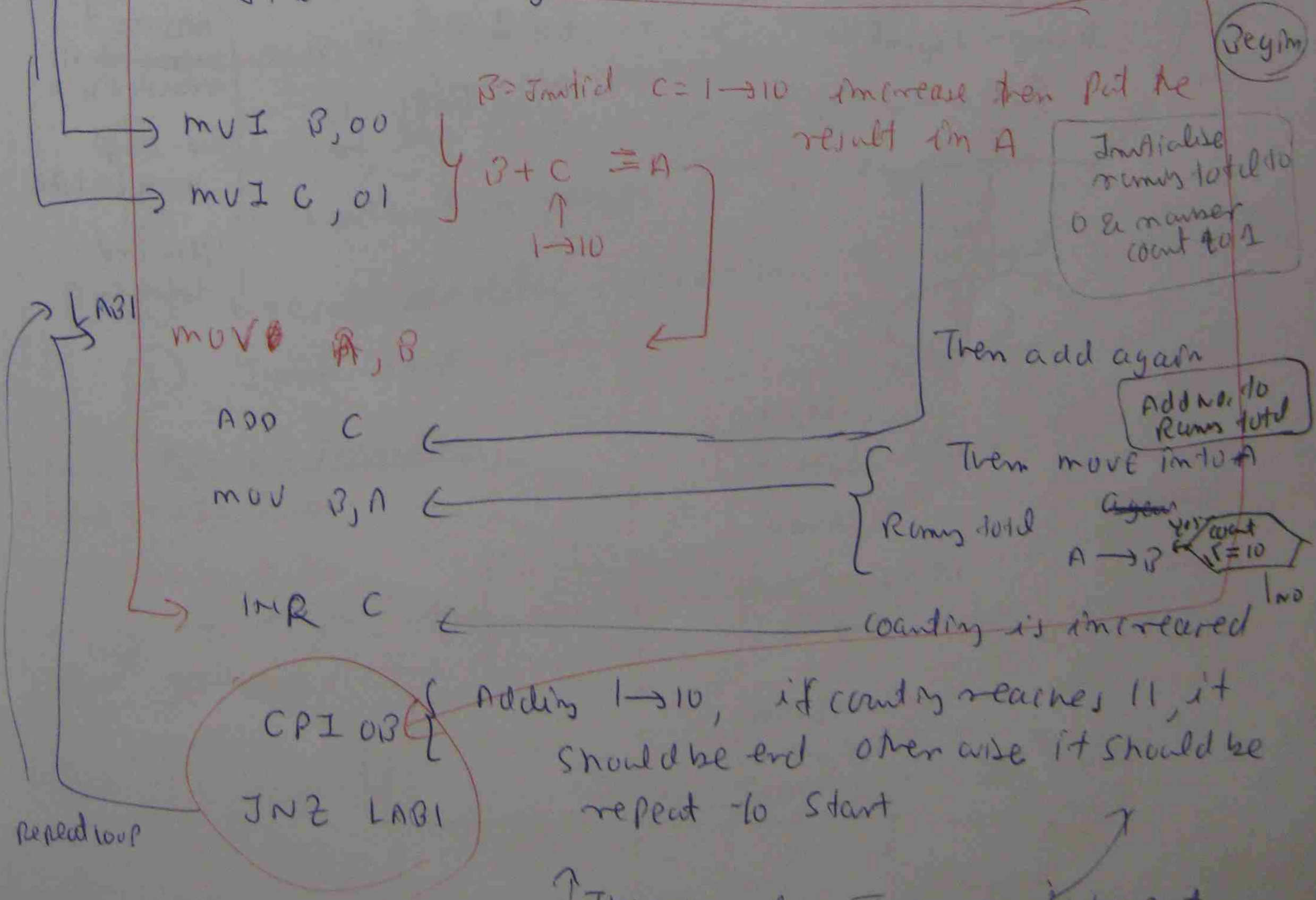
OP-code	condition	Flag status
JNZ (LAB1)	Jump if zero flag is not set to (LAB1)	Z = 0
JZ	Jump if zero flag is set to (LAB1)	Z = 1
JNC ( )	Jump if carry is zero	C = 0
JC	Jump if carry is 1	C = 1
JPO	Jump if parity is odd	P = 0
JPE	Jump if parity is even	P = 1
JP	Jump if S = +	S = 0
JM	Jump if S = −	S = 1

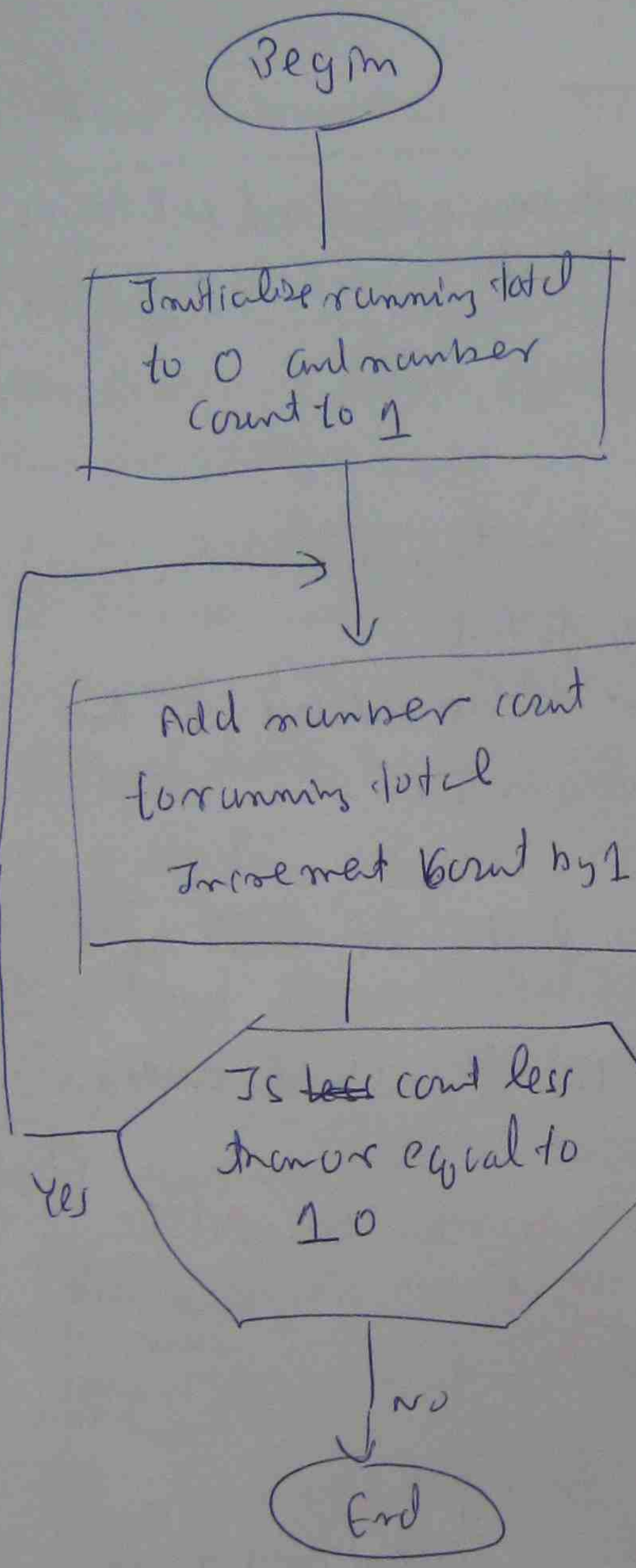
Flow Chart

A diagram which indicates the sequence of, and actions required in, a program and the points where branching is required.

Pb1 Construct the program to add together ten numbers 1→10

- B register contains the running total
- C register contains the number to be added to the running total. (~~Incs~~ start from 1)
- Increment by 1 & add
- If count is less than or equal to it will be repeated
- If the count is greater than 10 it will be ended.





	Assembly Instruction	Comment
	MVI B, 00	Initialise running total
	MVI C, 01	Initialise number count
LAB1	MOV A, B	Bring running total to A
	ADD C	Add number count
	MOV B, A	Restore running total in B
	INR C	Increment count
	MOV A, C	Bring count to A
	CPI 0B	Has count reached 11?
	JNZ LAB1	No- Jump back to LAB1
		Yes- End total in B

Imp loop ... Loop ... register

Jump Instruction

Time delay in the Program

- A common requirement when a microcomputer is interfaced to other equipment is to compute a time delay in the program.
- A micro computer-based road traffic light controller, for example, would need to compute a time delay to implement the sequencing of the light changes.
- Desired delay & loop count to be held in C-register.

Time delay NOP- No-operation instruction.

---

write a Program

① Load desired delay time parameter into C-register.
delay parameter is 02

② Clear A

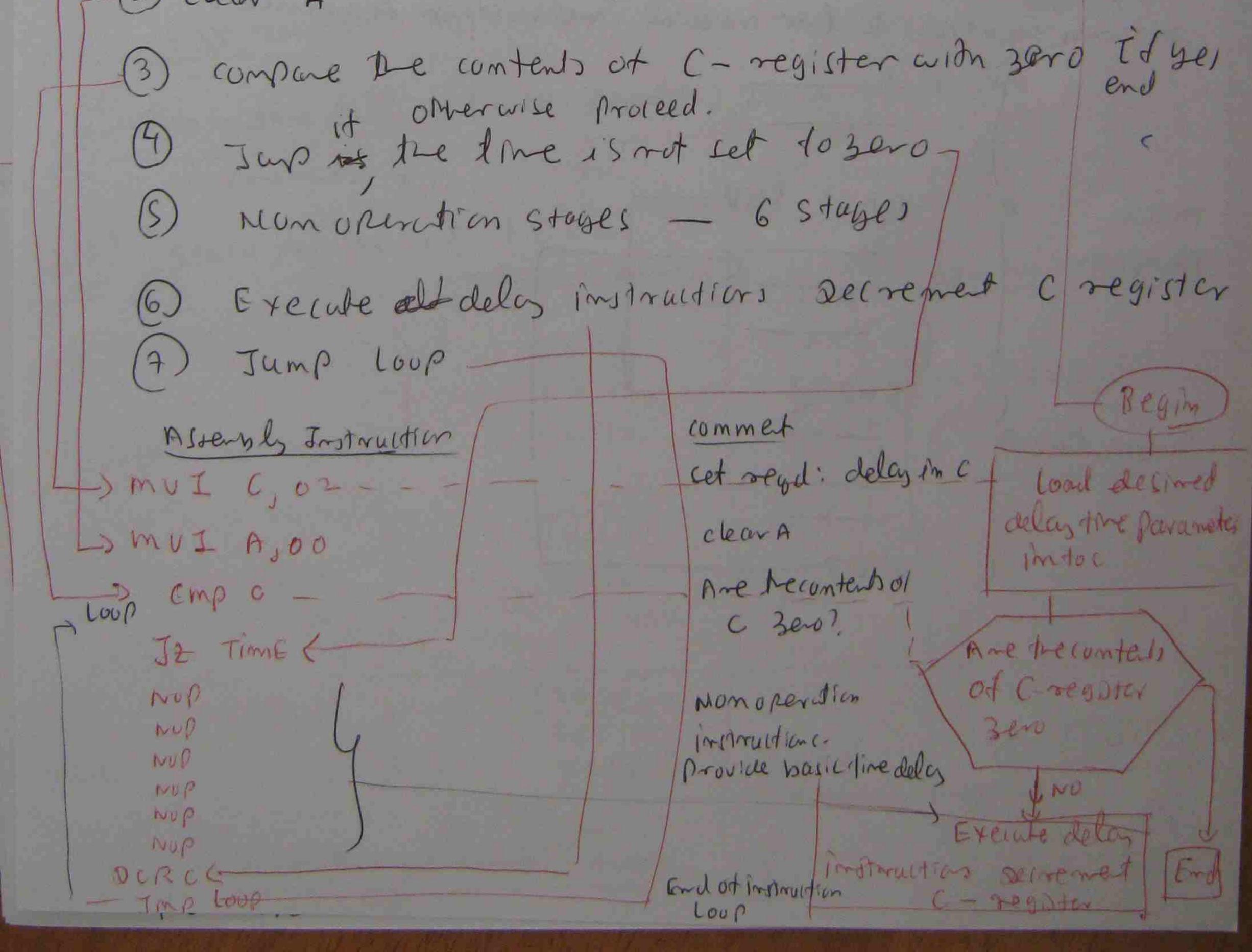
③ Compare the contents of C-register with zero If yes end

④ Jump if not, the time is not set to zero

⑤ Non operation stages — 6 stages

⑥ Execute delay instructions Decrement C register

⑦ Jump Loop



subroutines

pb 2 ← A short program to compute a time delay

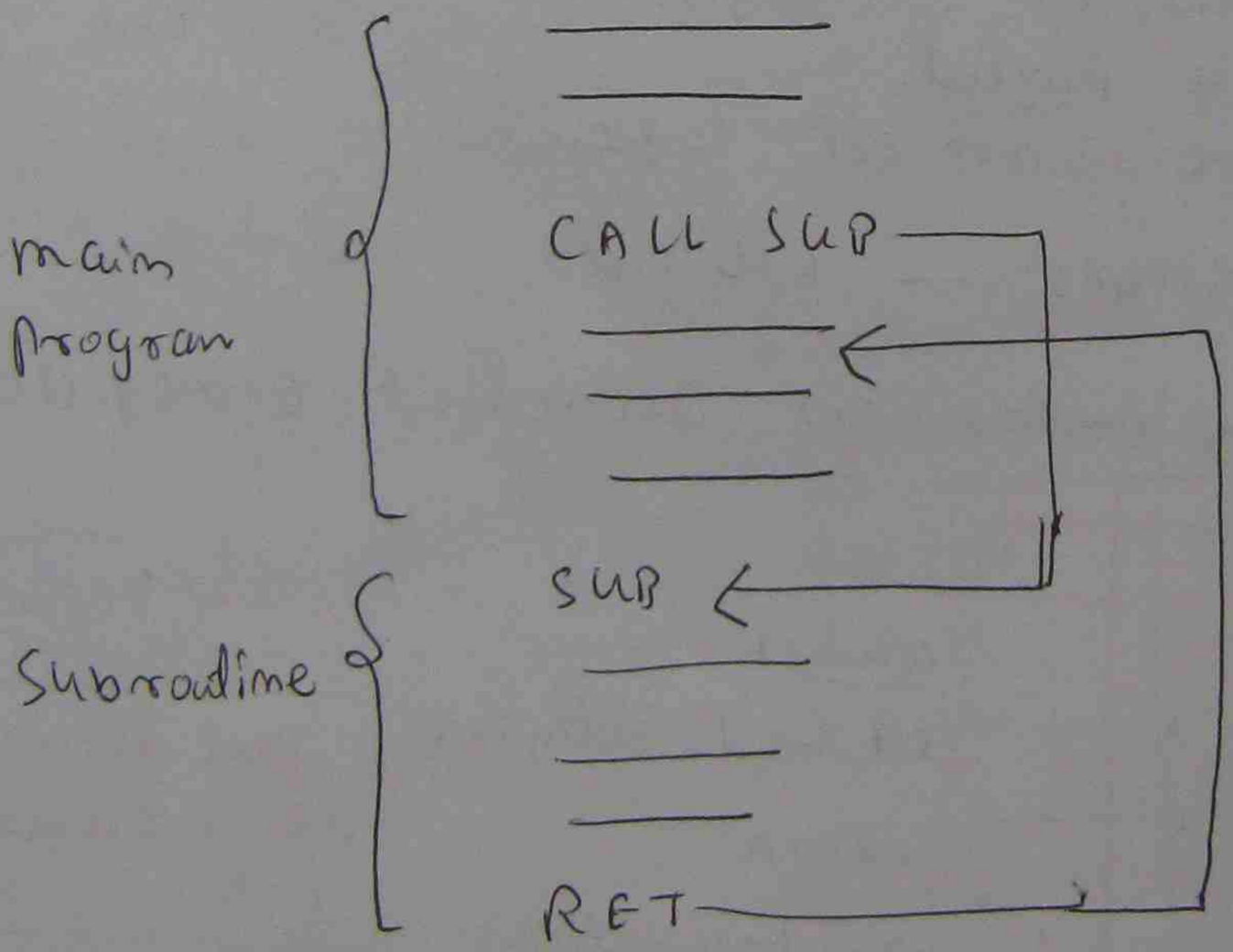
— This can be made into a subroutine by placing a RET instruction with the label TIME at the end of the program.

subroutine

Frequently within in a program, it may be necessary to perform a particular sub-task many times over.

~~It is highly desirable~~

subroutine is designed to perform the sub task and then return control to the main instruction sequence.



Stack

A last in first out queue which is implemented as a set of successive locations either within the processor itself or more usually, in the system memory.

Stack pointer register (SP)

it points to the address which currently holds the entry at the top of the stack, and consequently its contents change as each subroutine call and return instruction is executed.

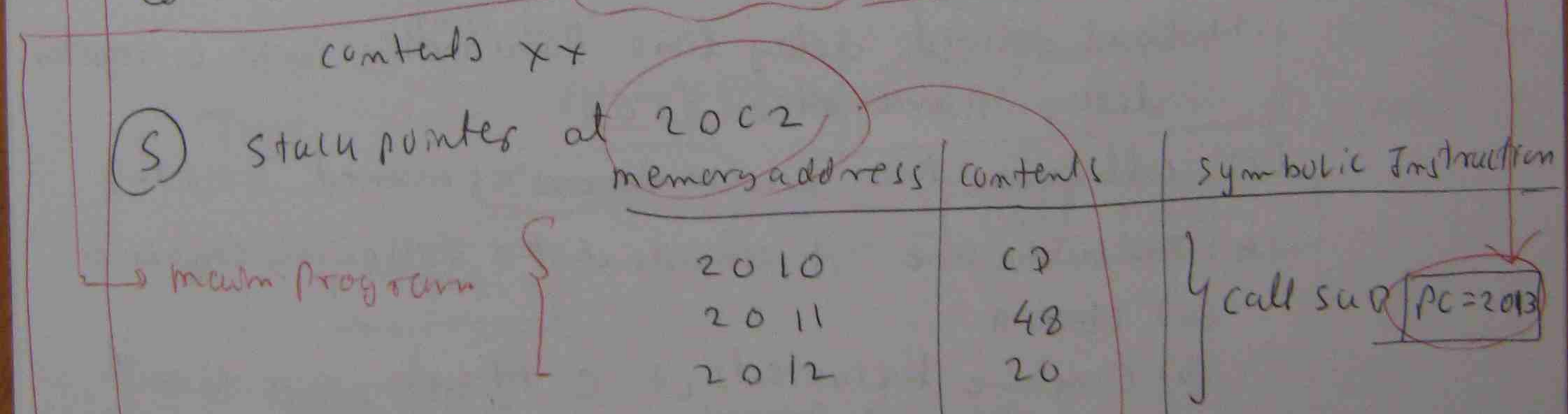
Ex ① main program memory address 2010 to 2012
contents CD, 48, 20 respectively

② Program counter PC - two bytes at 2013 call subroutine

③ Sub routine instruction 2048 → 2049 contents XX

④ Sub routine call (Stack ~~pointer~~) 2000 to 2002
contents XX

⑤ Stack pointer at 2002



Pb 3

memory Address		contents	symbolic instruction
main Program	2010	CP	CALL SUB PC = 2048
	2011	48	
	2012	20	
Subroutine	2048	XX	SUP
	2049	XX	
Stack	2010	13	SP = 2010
	2011	20	
	2012	XX	

Subroutine

Pb 3

(1) Initialize stack pointer at 2012
(2) Load desired delay time parameter into C-register
delay parameter is 02
(3) call subroutine which ~~is~~ ~~one~~ is named TIMEDLY
(4) In subroutine it consists of the following sequence
(a) Clear A
(b) Compare the contents of C if yes, goes to end
otherwise proceeds
(c) Jump if the time is not set to zero
(d) Non operation stages → 6 stages
(e) Execute delay instructions. Decrement C register
(f) Jump loop
(g) Return subroutine.

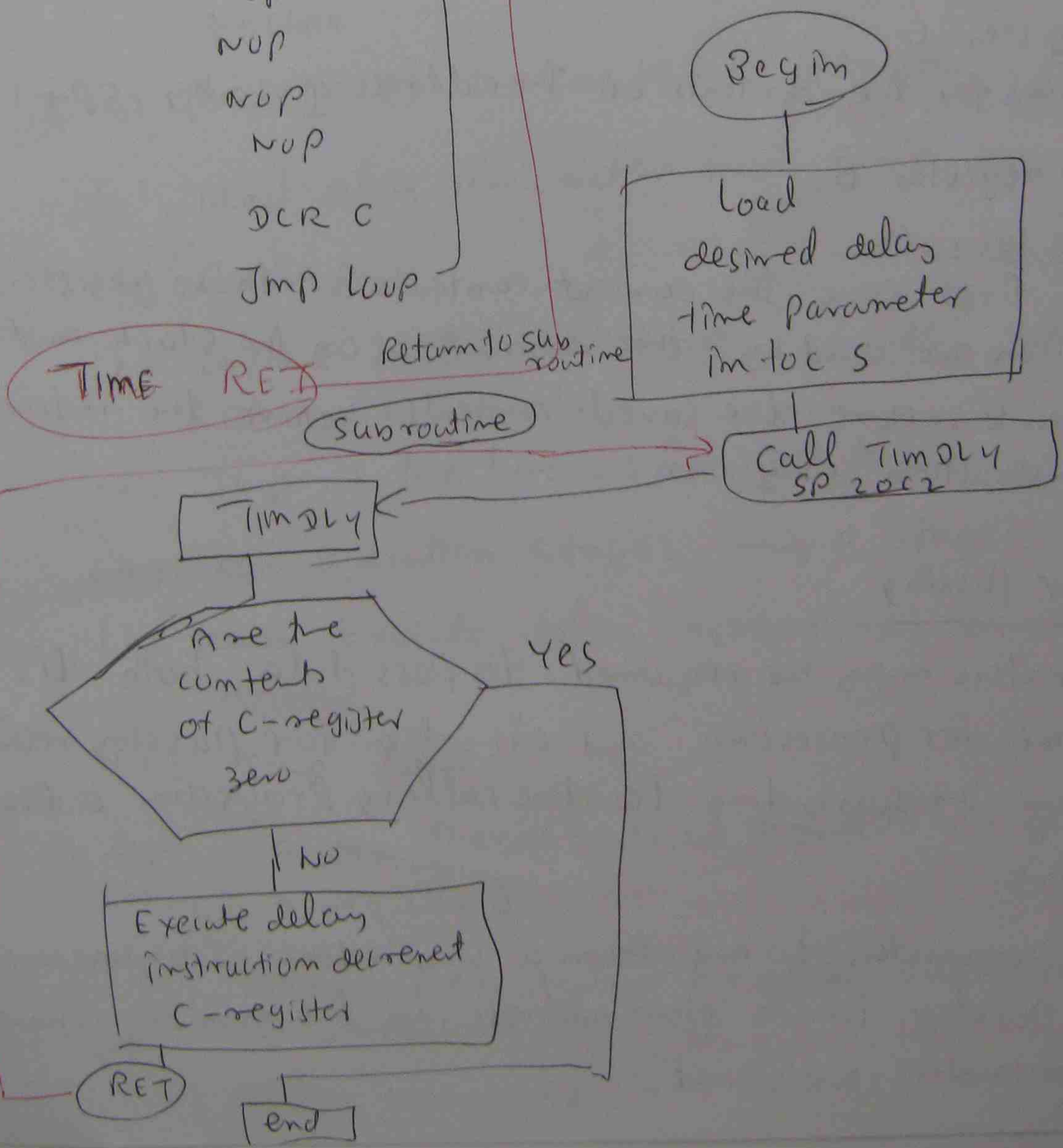
ORIGINAL PROGRAM

Additional for subroutine

```
LXI SP, 2002    <- Initialize Stack Pointer
MVI C, 02       Load TIMDLY Parameter
CALL TIMDLY     <- call subroutine
TIMDLY  MVI A, 00   Time delay subroutine
LOOP    CMP 0
        JZ TIME
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DCR C
        JMP LOOP
TIME    RET     Return to sub routine
```

delay routine

Flow Chart



Stack operation

The stack ones alsobe used as a temporary deposit for thecontents of processor register.

PUSH PSW (Push Processor status word)

This push (saves) the combined 16 bit contents of the A-register and flags register on the top of the stack.

POP BC

(i) Transfer the contents of the address given by SP to register C
(ii) Transfer the contents of the address given by SP+1 to register B

Writing subroutine

* To first save the current contents of those registers which are used by the subroutine on the stack and then to restore the saved contents before the return instruction is given.

Parameter Passing

- Parameters may be required to pass data both (i) to subroutine for processing and (ii) also for passing results back from the subroutine to the calling program after processing.

- Passing parameters to and from a subroutine is by means of a pointer to the start address in the memory where the parameters are stored.

Pb 4

(1) Initialize stack pointer at 20C2

(2) Store delay parameter in memory location 2080

(3) Load delay time parameter into C register
delay parameter is 02

(4) call subroutine which is named TIMDLY

(5) In subroutine, it consists of the following sequence.
(a) Push (save) the combined 16 bit contents of the A register and flags register on the top of the stack
<Push PSW>

(b) Push (save) the contents of register C on stack
<Push B>

(c) Read delay parameter from memory C

(d) Clear A

(e) compare the contents of C, if yes, goes to end otherwise proceeds

(f) Jump if the time is not set to zero

(g) Nom operation stages ⟶ 6 stages

(h) ~~Decre~~ Execute delay instructions - Decrement C register

(i) Jump loop

(6) ~~(j)~~ Transfer the contents of the address given by
(a) sp to register B <Restore contents of register C>

(b) Transfer the contents of ~~the given~~ address given by
sp to register A <Restore contents of register A>

(c) Return subroutine

Original Program with subroutine

Address for parameter passing

```
LXI  SP, 20C2      Initialise stack pointer
LXI  H, 2080   }
MVI  M, 02     }   ——  Store delay parameters in memory location 2080
CALL TIMDLY        ——  call subroutine
```

MVI M, FF →

```
TIMDLY  PUSH PSW        Sa
        PUSH B          Sb      }  Save content of registers A2C on Stack
        MOV C, M        Subroutine 1
        MVI A, 00       Sc
LOOP    CMP C                   Read delay parameter from memory
        JZ  TIMC
        NOP
        NOP
        NOP
        NOP
        NOP
        DCR C
        JMP LOOP
TIMC    POP B           Ga      }  Restore contents of registers A2C from Stack
        POP PSW         Gb
        RET                     Return subroutine
```

LXI H, 2081
MVI M, FF
CALL TIMDLY2 →

Store delay parameter in memory location 2081

Remove.

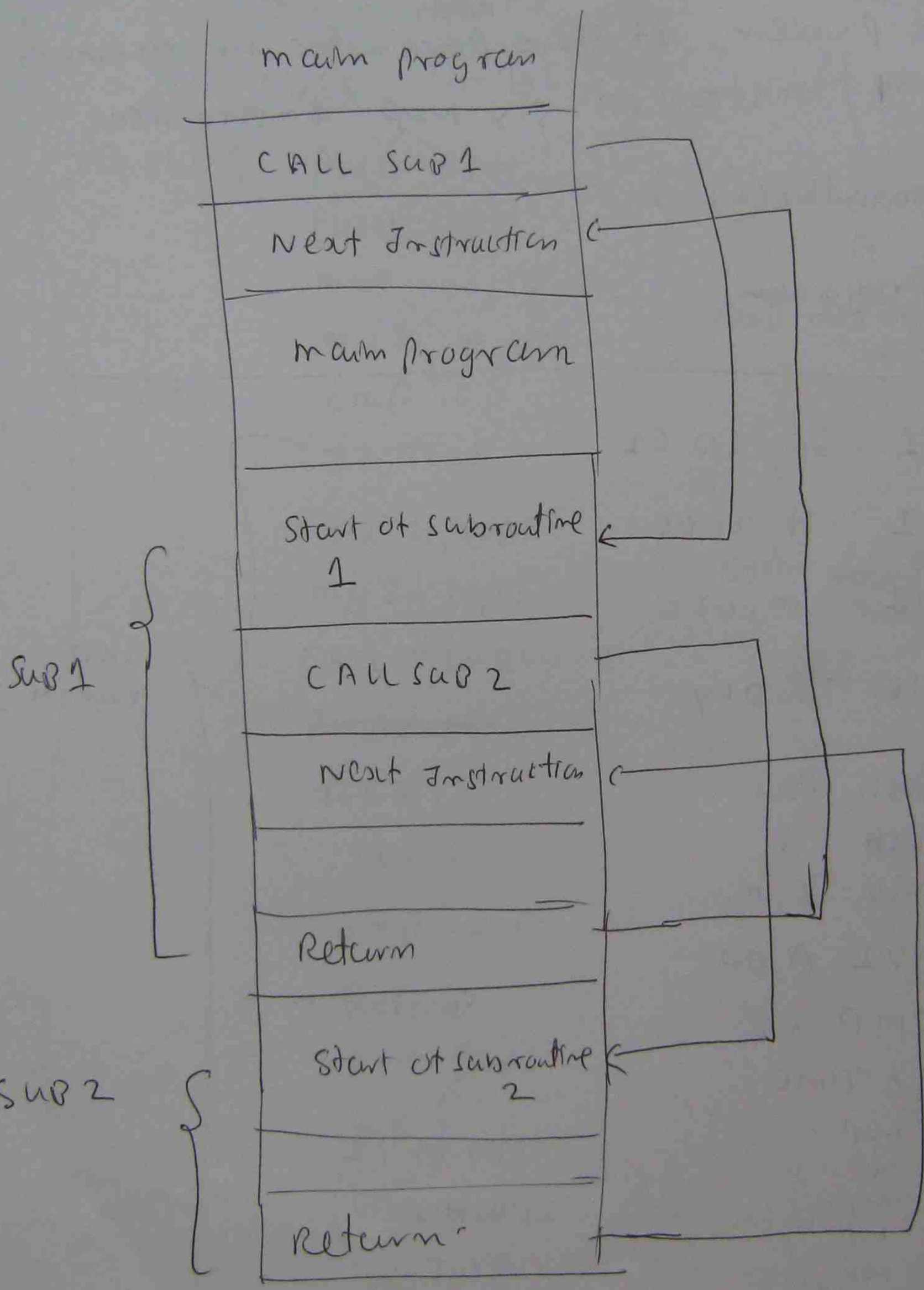
Delay routine

Sub routine 2:

```
TIMDLY2  PUSH PSW
         RET
```

Nested subroutine

A sub routine calls another subroutine with in itself



Pb 5 — modification of Pb 4

In above problem, it store delay parameter in memory location 2080 instead of six NOP instructions, as subroutine (2)

write the program.

```
        LXI   SP, 20C2
        LXI   H, 2080
        MVI   M, 02
        CALL  TIMDLY
TIMDLY  PUSH  PSW
        PUSH  B
        MOV   C, M
        MVI   A, 00
LOOP    CMP   C
        JZ    TIME
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DCRC
        JMP   LOOP
TIME    POP   B
        POP   PSW
        RET
```

MVI M, ff

LXI H, 2081
MVI M, ff
CALL TIMDLY2

sub routine 1

Subroutine 2

```
TIMDLY2 PUSH PSW
        RET
```

main program

```
LXI  SP, 20C2      ← Initialise stack pointer
LXI  H, 2030       } ← store delay parameter in memory location 2030
MVI  M, FF         }
CALL TIMDLY1       ← call subroutine 1
```

Sub routine 1

```
TIMDLY1  PUSH PSW      } ← save contents of register A & C
         PUSH B        }
         MOV  C, M     } ← Read delay parameter from memory
         MVI  A, 00
LOOP     CMP  C
         JZ TIME
         LXI H, 2031   } ← store delay parameter in memory location 2031
         MVI M, FF     }
         CALL TIMDLY 2 ← call subroutine 2
         ~~PUSH PSW~~
         ~~RET~~
         DCR C
         Jmp LOOP
         POP B
         POP PSW
         RET
```

Sub routine 2

```
TIMDLY2  PUSH PSW   } ← As for TIMDLY1 except MVP instructions are replaced by subroutine 2 call
         RET
```

Exercise

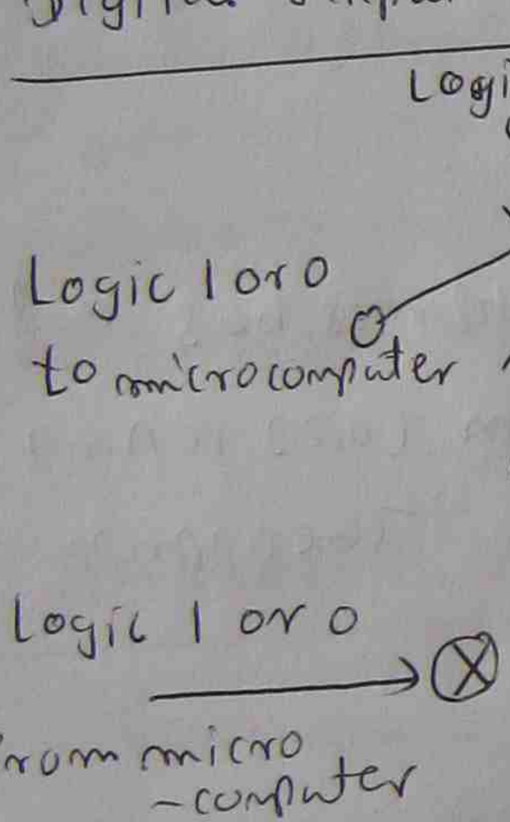
- Look at questions in Exercise 5.1 to 5.7
- Study the answers provided for exercise 5.1 to 5.7
- Observe the way to find the solution
- Take practice with microprocessor training software

Week ⑥

Digital Input and output

- A microcomputer is basically a digital component that can examine digital input signals and perform functions as a consequence of these inputs to yield digital output signals.
- The external devices outside the micro computer produce (or) accept signals which are not necessarily digital in nature, special interface circuitry is often required to transform these external signals into a form suitable for the microcomputer

Digital Input and output

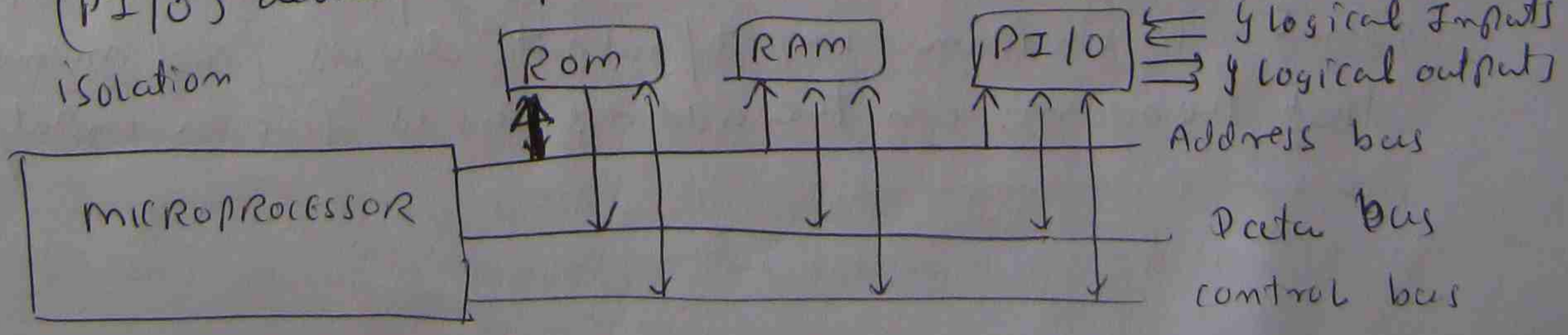


A simple digital input to a microcomputer can be produced by a single pole switch.

Logic 0, 1 depends on switch position

LED ON for logical 1
LED off for logical 0

- The information intended for an output indicator must be latched by a suitable circuit.
- The processor can then send data to the output latching device which captures the data at the appropriate time determined by bus control signals and then provides a continuous output until new data is sent to it.
- The system incorporates a programmable input/output (PI/O) device to provide the necessary latching and isolation

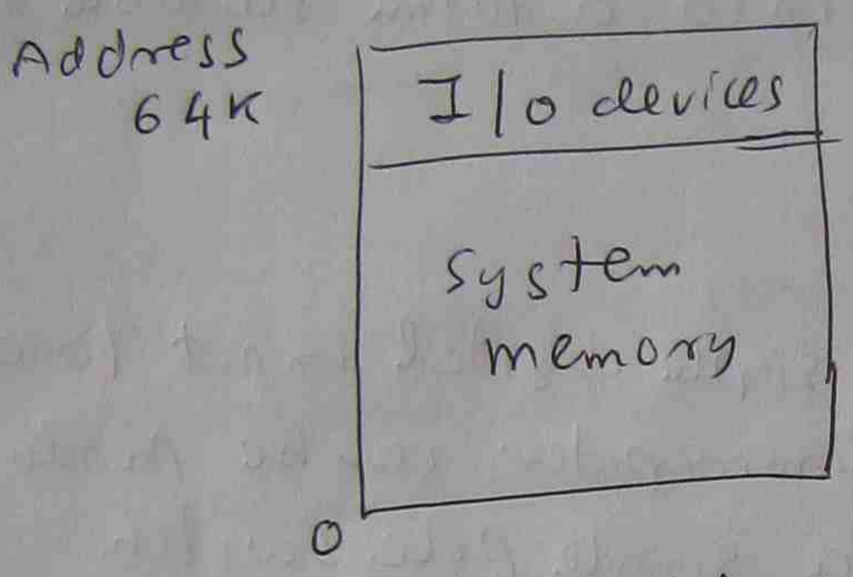


Data transfer of input/output data between a microcomputer bus and input/output device

---

- memory mapped input/output
- Programmed input/output.

i memory mapped input/output

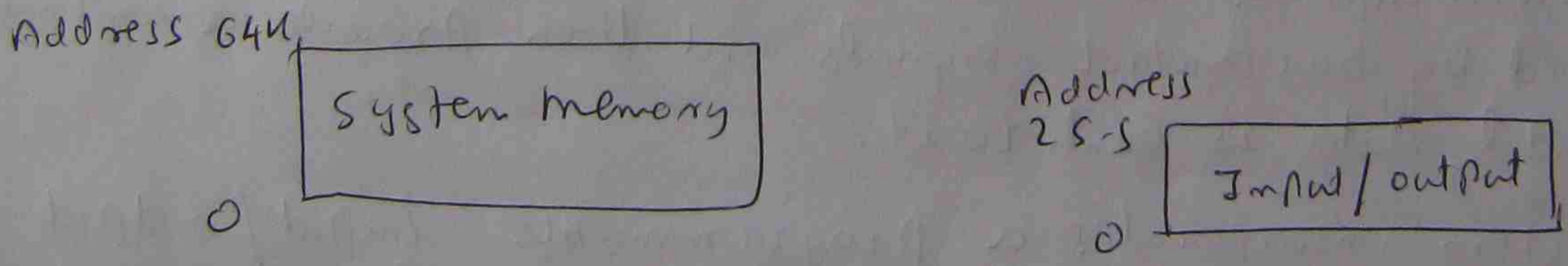


- The same instructions are used for both memory and input/output data transfer

The appropriate address is output on the address bus and recognised either by a memory device (ROM (or) RAM) or input/output device (PI/O) or port. The appropriate data is transferred on data bus.

ii Programmed input/output

- Input/output data transfers are accomplished by means of special instructions executed by the Processor - IN and OUT for the Intel 8085.



- microprocessor generates an input/output request signal to inform input/output devices (and memory) that the address on the address bus is for an input/output device.

iii Programmable input/output

Digital input and output in most microprocessors is controlled by Programmable input/output devices and Programmed input/output is normally used. A PI/O device can control a number of individual input and output lines. These are normally grouped into a number of ports, each comprised of eight lines which may be Programmed to operate either as inputs or as outputs.

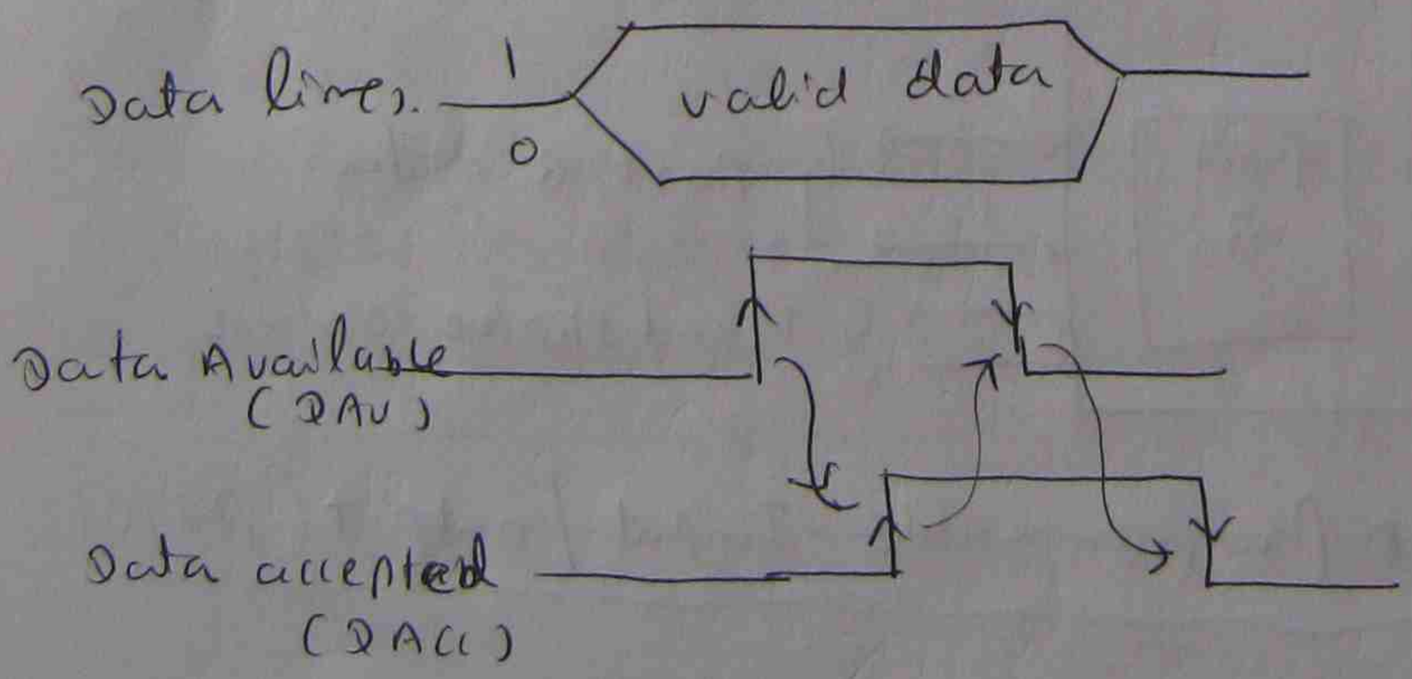
Steps

- write appropriate command information into a specific addressable registers within the device when the system is being initialised
- Data is read from (or) written to a port.

Handshake control

Synchronise the transfer of data between PI/O and external device and consequently most PI/Os provide control lines for this function.

Hand shake – Typical transfer sequence.

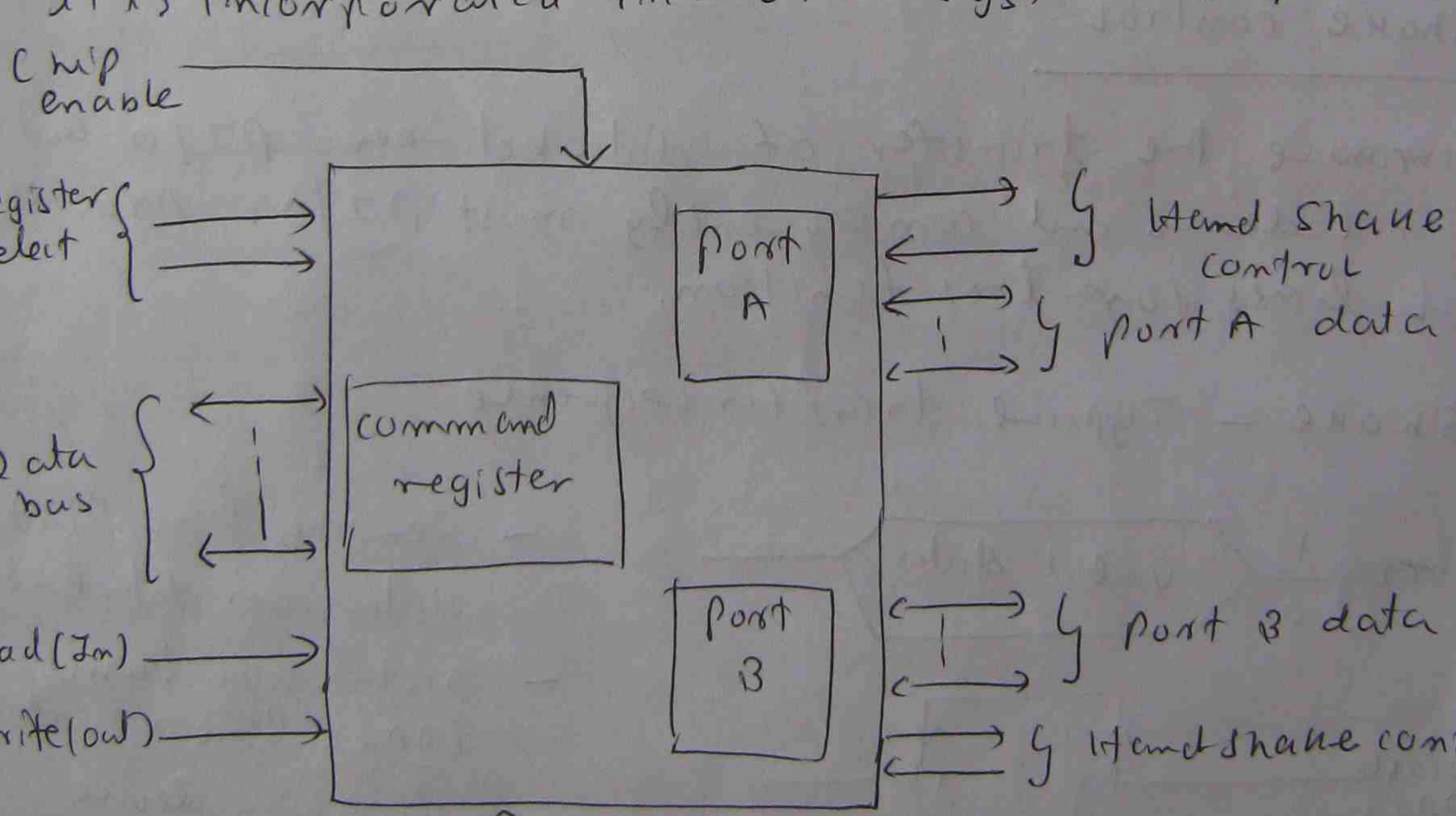


- Sending line Places data on data line
- Data available (DAV) line is set
- Receiving device detects the setting or DAV line
- Then responds by setting the data accepted "DACC" line
- Sending device interprets the setting or DACC line

- Receiver detects that DAV line has been reset
- Reset DACK line to permit further data transfer.
- There is a chip enable input on all the devices which are connected to the microprocessor bus.
  - RAMs, ROMs, PIOs - They are used to ensure that only one device responds to each data transfer on the bus.
- Port A, B select the appropriate register within PIO itself - command.

Port Initialisation

Some microprocessor, each programmable input/output device is a separate integrated circuit but in others. it is incorporated into other system components.



A Programmable Input/output (PIO)

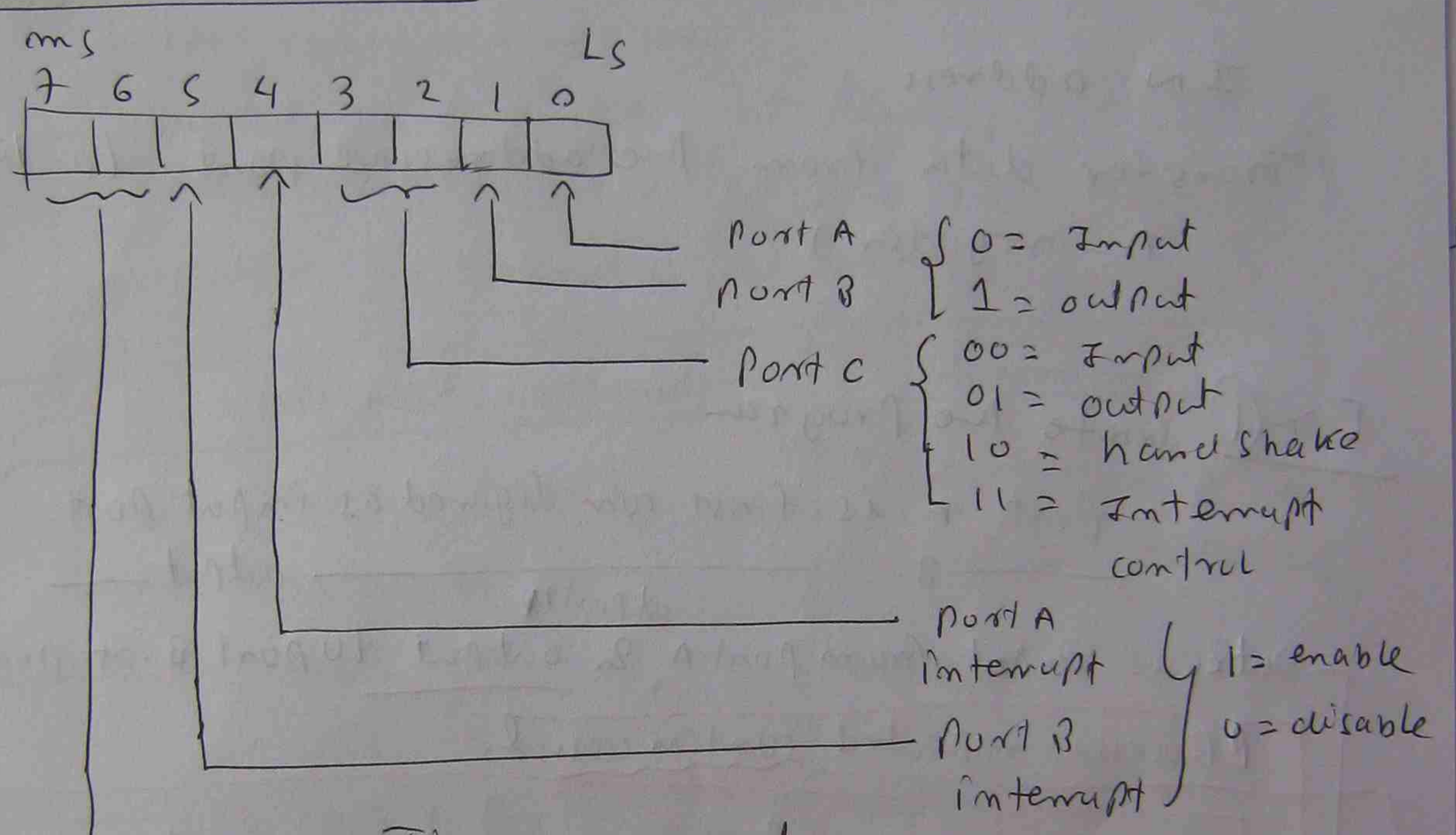
Command information

- Initialise the port, Timer and input/output ports are programmed together

OUT address

(Transfer the contents of the Processor A register to the addressed input / output device)

8155 command byte

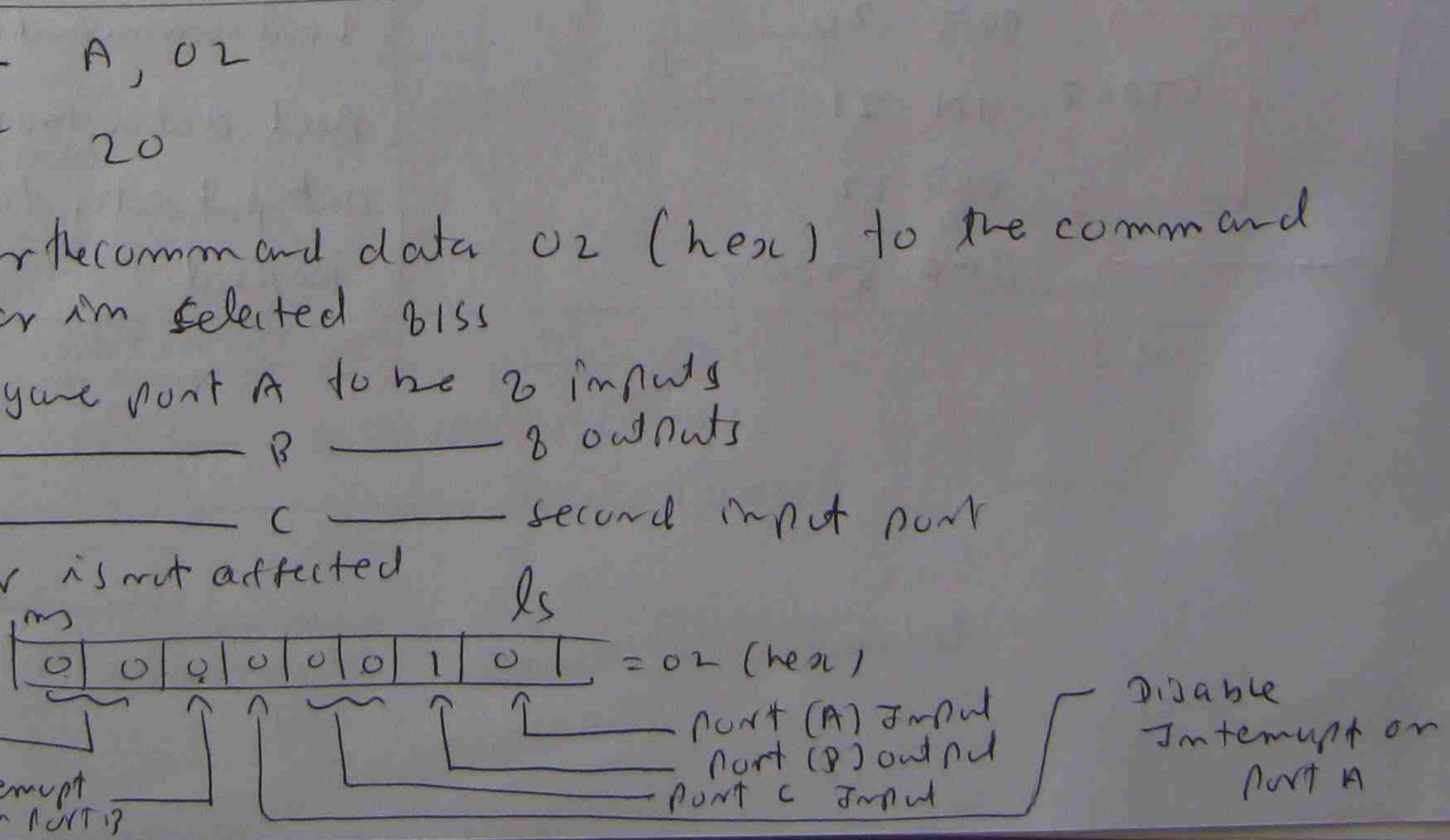


MVI A, 02

OUT 20

- Transfer the command data 02 (hex) to the command register in selected 8155
- configure port A to be 2 inputs
- ——— B ——— 8 outputs
- ——— C ——— second input port

Timer is not affected



Address (hex)	Port / Register
20	command / status register
21	Port A
22	Port B
23	Port C

Typical Port / register addressing

IN address
(Transfer data from the addressed port to the A - register)

Ex① write the Program
- Port A is first configured as input port
- B ——— output ———
- Data is input from port A & output to port B. at port 22
- Process is repeated continuously

out port 22

Assembly Instruction		comments
	MVI A, 02	Load command byte in A
	OUT 20	Load command register
START	IN 21	Read data from Port A
	OUT 22	out put data to Port B
	JMP START	Repeat

Ex2 Initialize port A, B, C as output ports.
then output decimal number 24937 in BCD form.

then
- load command byte in A
- Load command register
- output 24 to port A at line 21
- output 93 to port B at line 22
- output 07 to port C at line 23

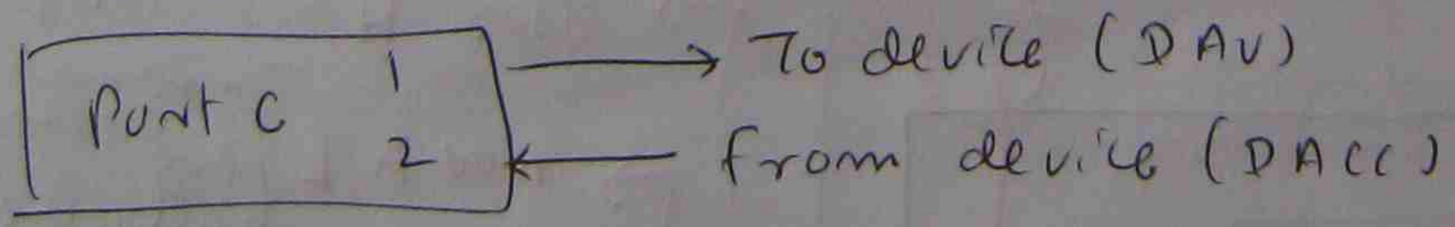
Assembly instruction	comments
MVI A, 07	Load command byte in A
OUT 20	Load command register
MVI A, 24	output 24 to port A
OUT 21	
MVI A, 93	output 93 to port B
OUT 22	
MVI A, 07	output 07 to port C
OUT 23	

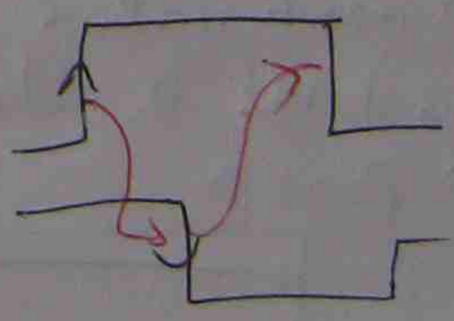
Port A
0010 0100
2 4

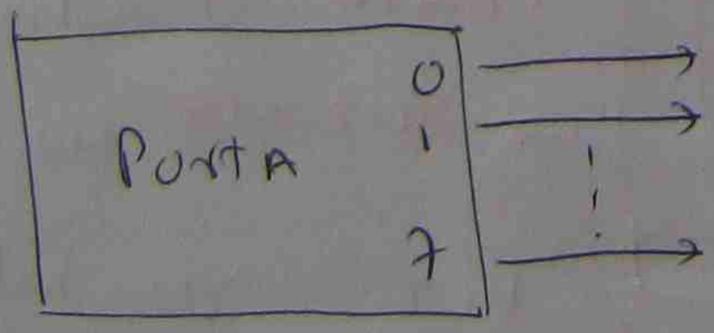
Port B
1001 0011
9 3

Port C
xxxx 0111
7

Handshake control







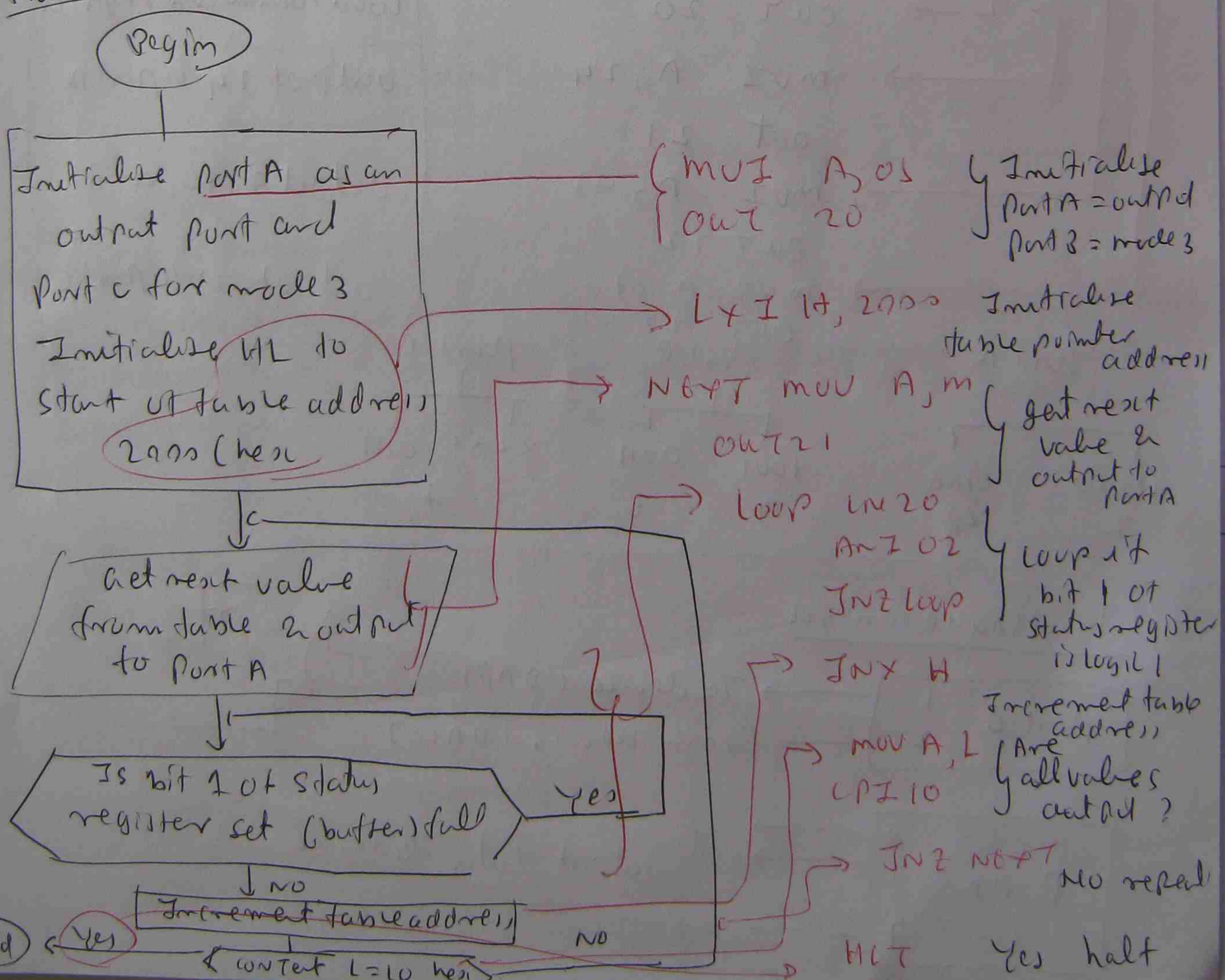
3 & 4M

E+3

Handshake control lines of Port C to output 16 values from a table in memory - starting address 2000 (hex) to an external device connected to Port A.

- Port C has been initialised to operate in mode 3 (hand shake mode)
- After data has been output to Port A bit 1 of Port C will automatically go to logic 1, indicating to the external device that new data available.

Flow

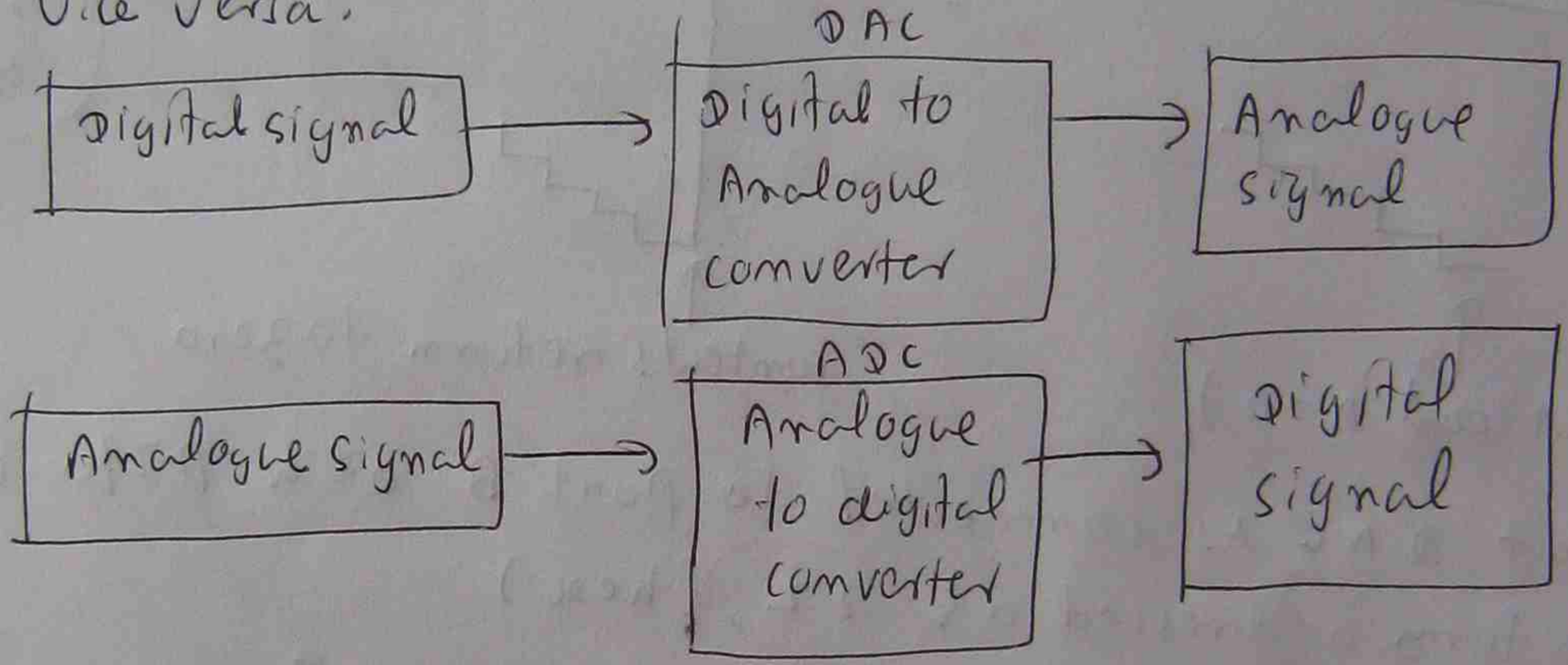


WEEK (7) ANALOG INPUT/OUTPUT & INTERRUPT

Input data - varying analogue signal, output voltage from a temperature transducer.

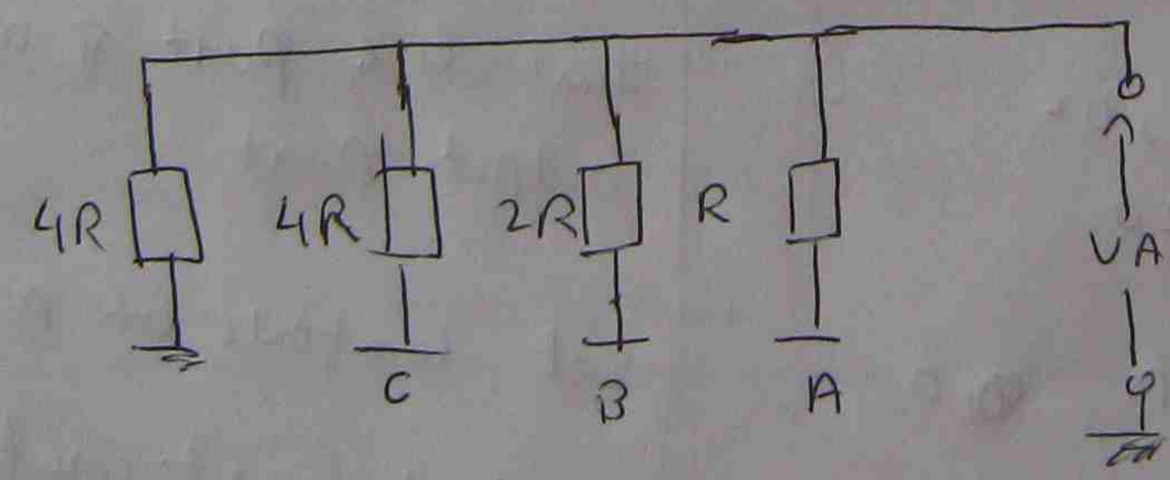
Output data - The output data from the microprocessor is often required in analogue form. For example to drive a motor.

It is necessary to have additional interface circuitry between the input/output ports of the microprocessor and the controlled peripheral devices, both to convert analogue signals in to digital form and vice versa.

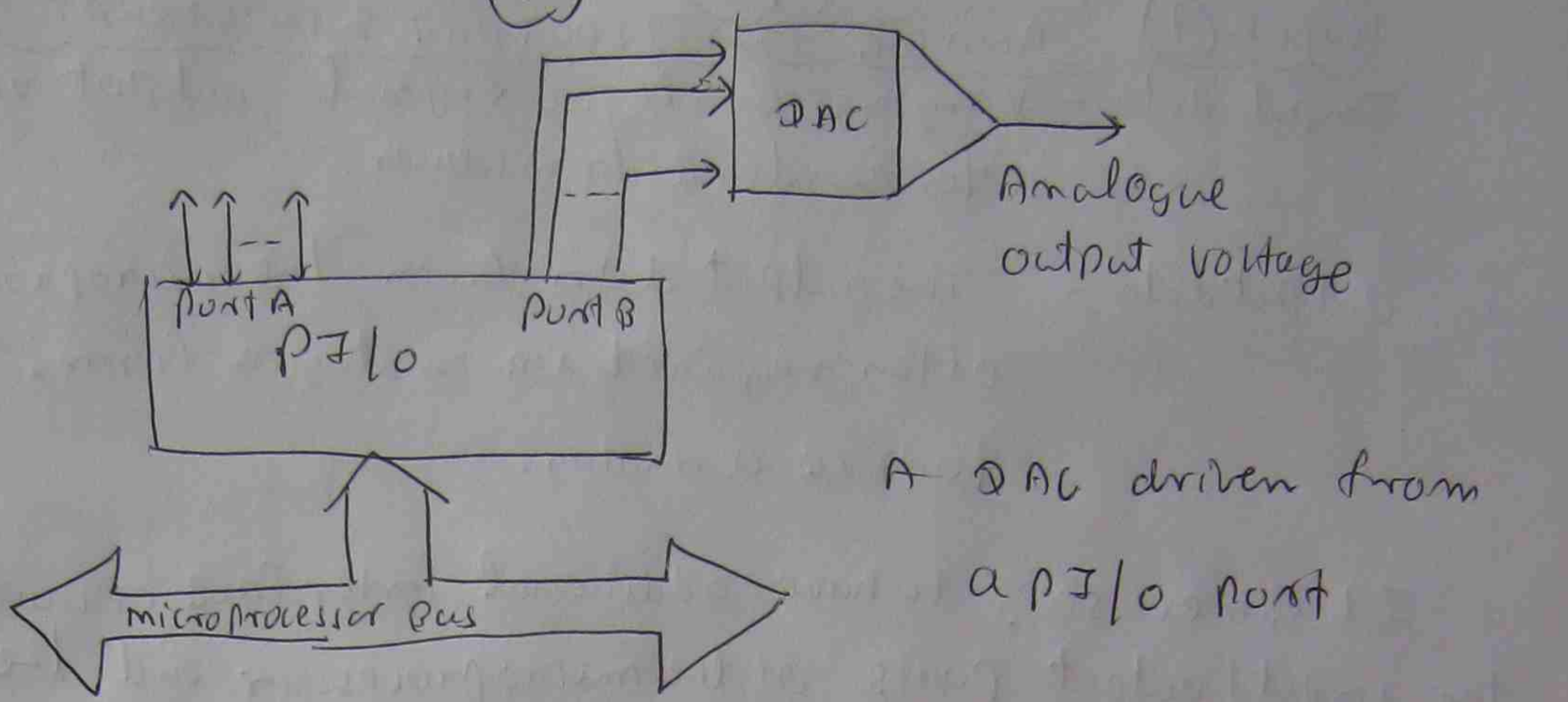


Digital to analogue conversion

A digital number can be converted to an analogue voltage by selectively adding voltages which are proportional to the weighting of each binary digit.

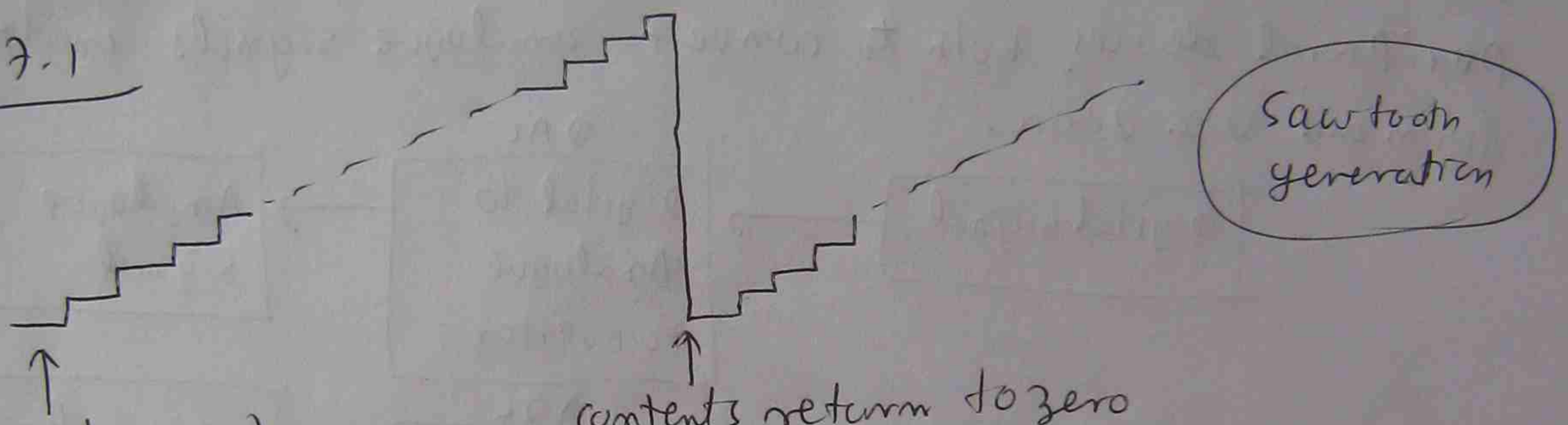


A	B	C	VA
0	0	0	0
0	0	1	V/8
0	1	0	V/4
0	1	1	3V/8
1	0	0	V/2
1	0	1	5V/8
1	1	0	3V/4
1	1	1	7V/8



A DAC driven from a PIO port

Ex 7.1



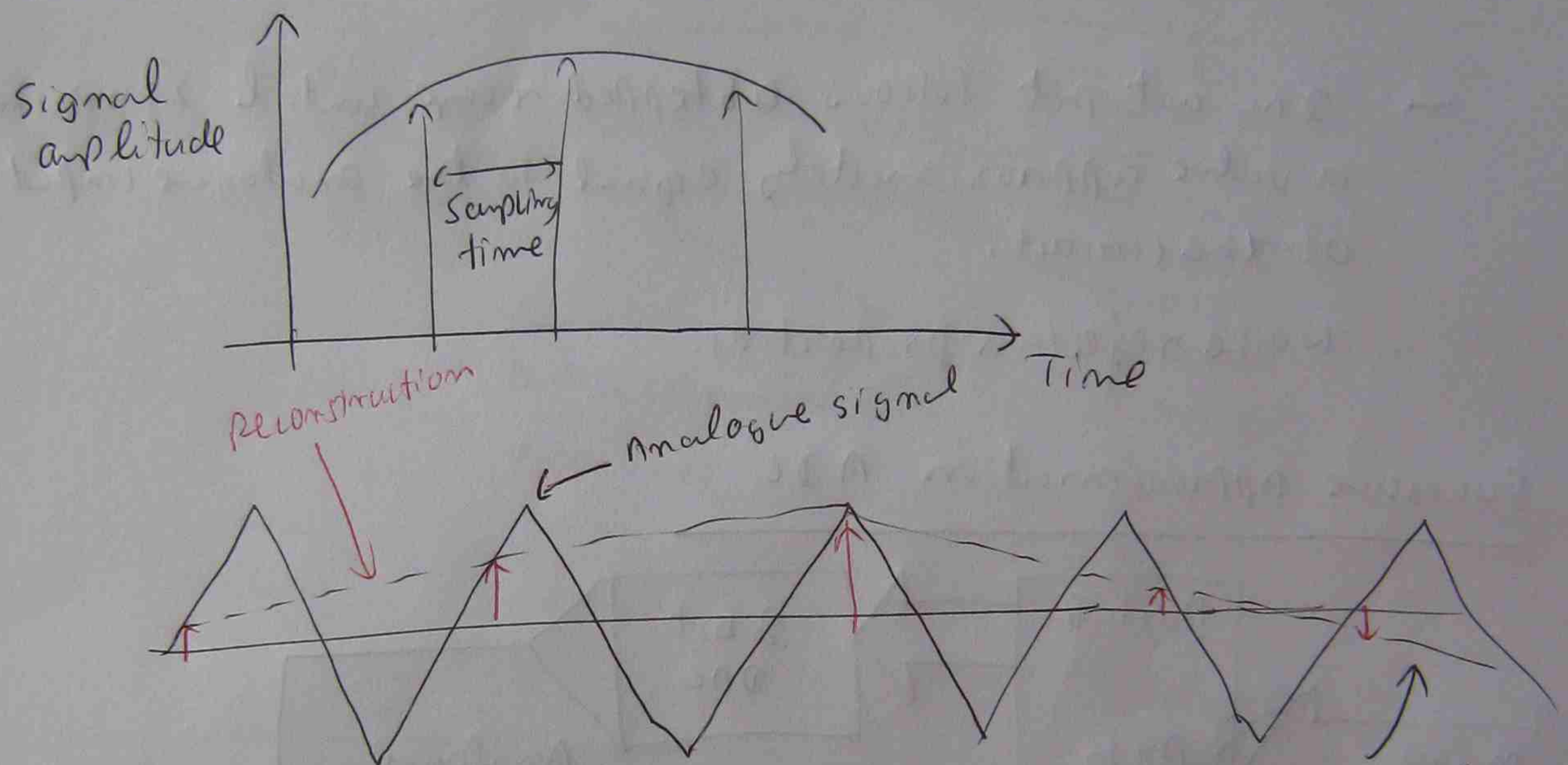
8 bit DAC is connected to Port B of a PIO which is in turn addressed as 22 (hex)

A saw tooth wave form is then readily generated by using the A-register as a counter. & outputting its contents after each increment. Write a program

Assembly Instructions	comments
MVI A, 02 OUT 20	Initialize Port B as an output port
MVI A, 00	Set contents of A to zero
COUNT OUT 22	Output current count
INR A	Increment count
JMP COUNT	Loop back

Analogue to digital conversion

The conversion of an analogue signal to a digital number implies a process of signal sampling. A digital number can only accurately represent a changing analogue signal for a short period of time.

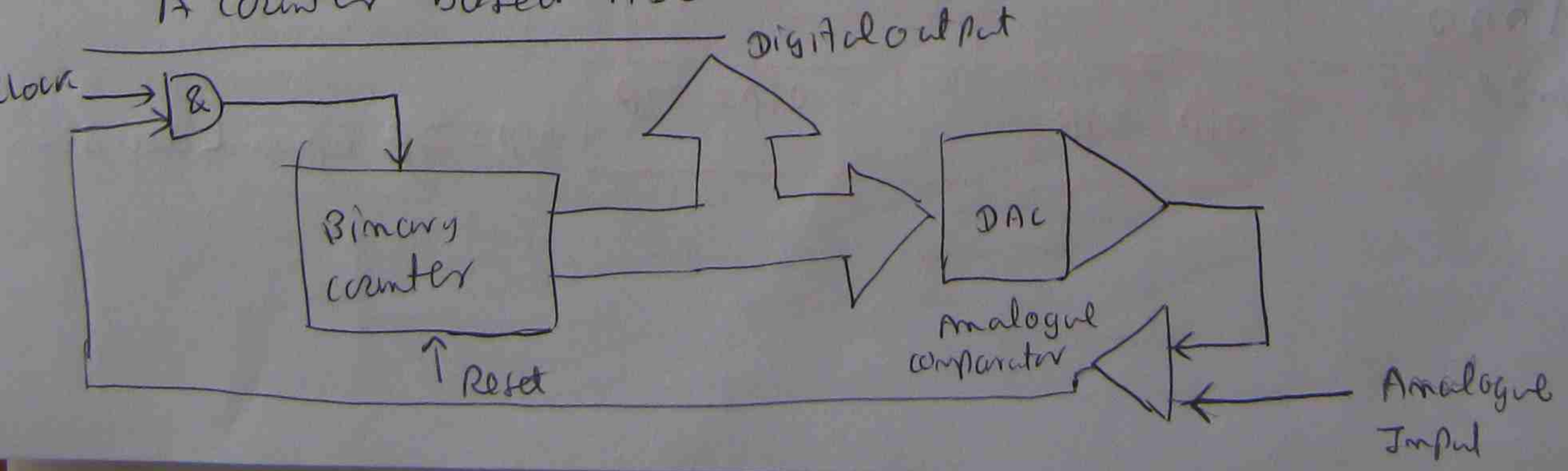


The shannon sampling theorem

It is possible to severely distort the digital representation of an analogue signal by sampling it too infrequently.

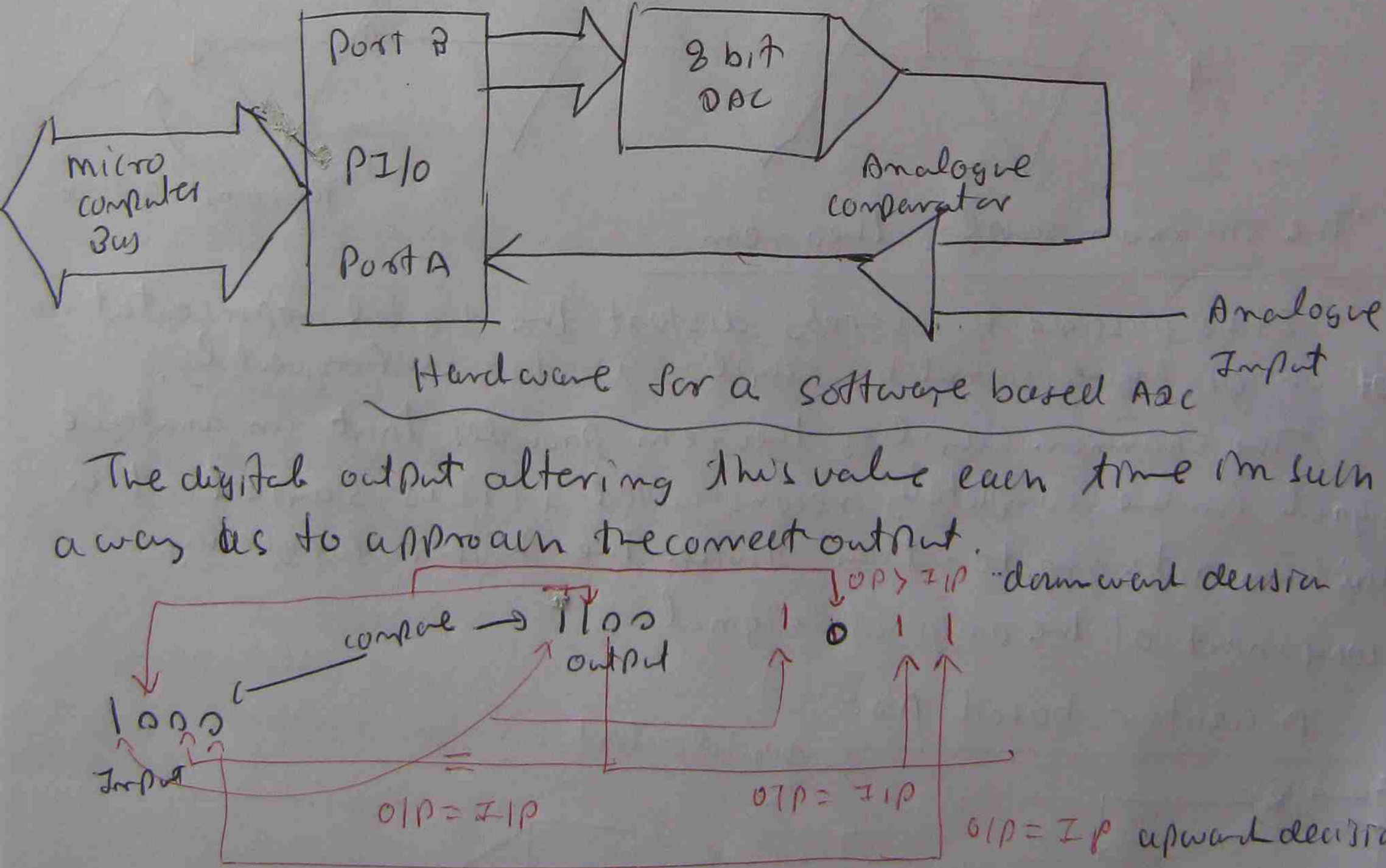
The shannon sampling theorem provides that an analogue signal can be completely reconstructed if it is sampled at a uniform rate greater than twice the higher frequency component of the original signal.

A counter based ADC



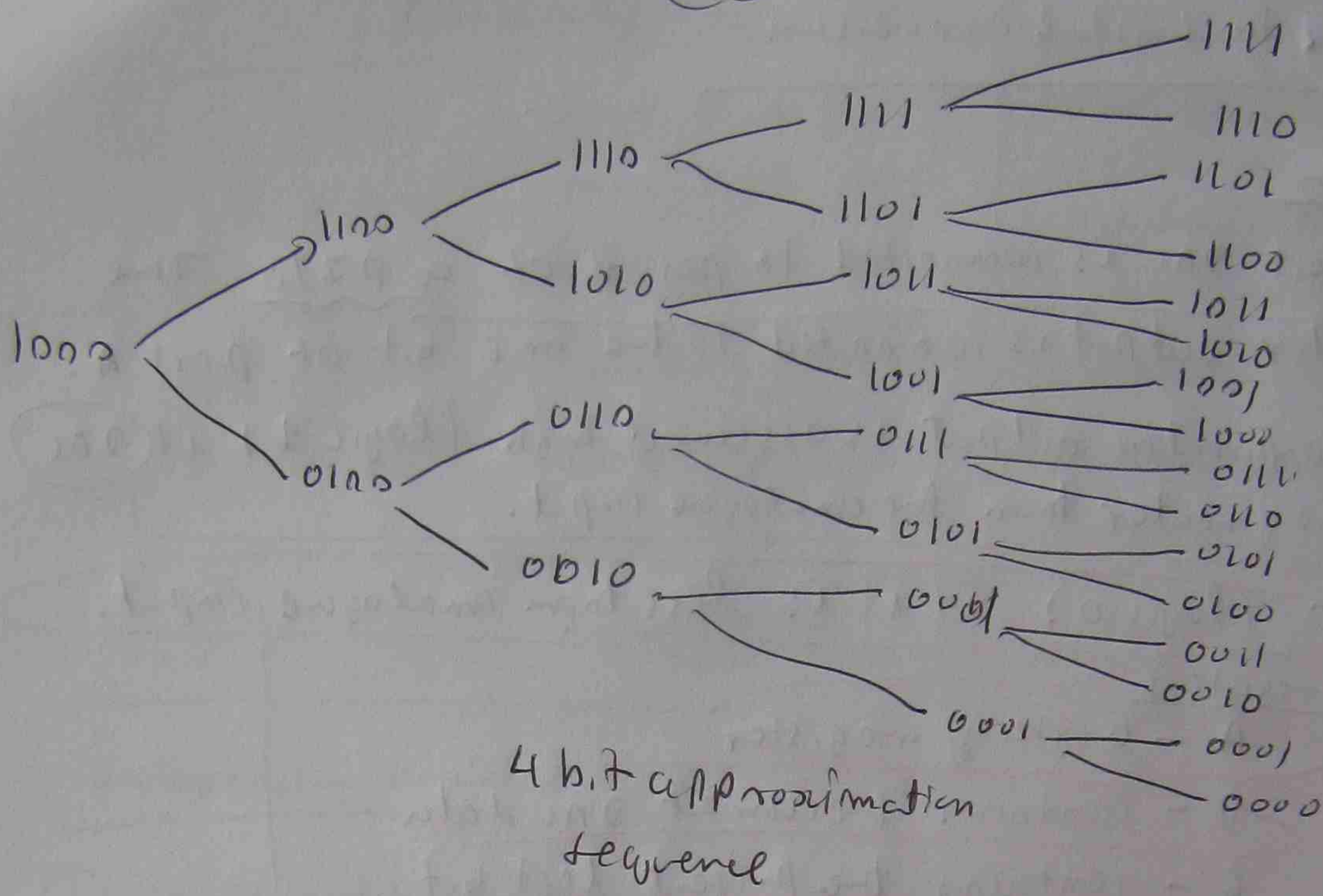
- The counter is initially reset at the start of a conversion.
- until DAC output just exceeds the analogue input, counter clock pulses are then enabled, causing the comparator to further counter clock pulses.
- Digital representation of the analogue input is then binary output of the counter
- DAC out put follows a stepped ramp until it reaches a value approximately equal to the analogue input of the circuit.
- Noise rejection properties.

Successive Approximation ADC

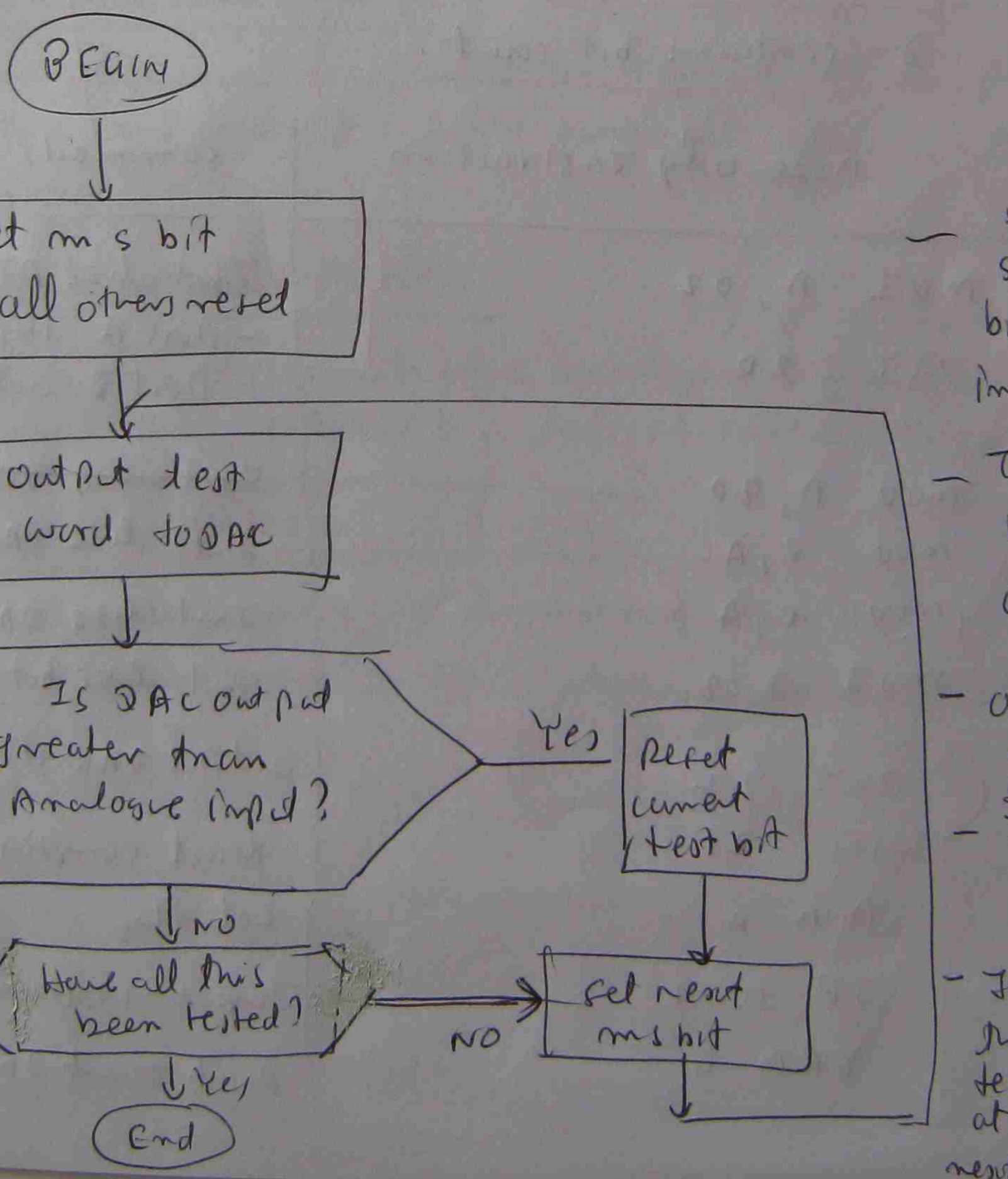


Hardware for a software based ADC

The digital output altering this value each time in such a way as to approach the correct output.



4 bit approximation sequence



Analogue to digital conversion

Ex 7-2

8 bit DAC is connected to port B of a PIO. The comparator output is connected to the ms bit of port A.

The comparator output is assumed high (logic 1) if DAC output is greater than the analogue input.

Low (logic 0) if it is less than analogue input.

Processor registers

A - working register

B - contains the current DAC data

C - contains the present test bit

D - contains bit count.

Assembly Instruction		comments
MVI	A, 02	Initialise PIO
OUT	20	- Port A Input
		- Port B output
MOV	A, B0	~~Initialise DAC data~~
MOV	B, A	Initialize DAC ~~test data~~ med
MOV	C, A	Initialise DAC test bit
MVI	D, 08	Initialise bit count
REPEAT: OUT	22	output DAC data
IN	21	Read comparator output
ANA	A	Set flag
JP	CONT	Jump if comparator o/p = 0
XRA	C	Reset count DAC test bit

Assembly Instruction		comments
COMZ	MOV B, A	save DAC data
	MOV A, C	update DAC test bit
	RAR	
	MOV C, A	
	ORA B	set next ms bit of DAC data
	DCR D	Decrement bit count
	JNZ REPEAT	Jump if not zero

Interfacing analogue devices

(1) Sample and hold circuit

There are used to sample a signal at a precise time and hold the value constant during the conversion process

(2) Analogue multiplexers

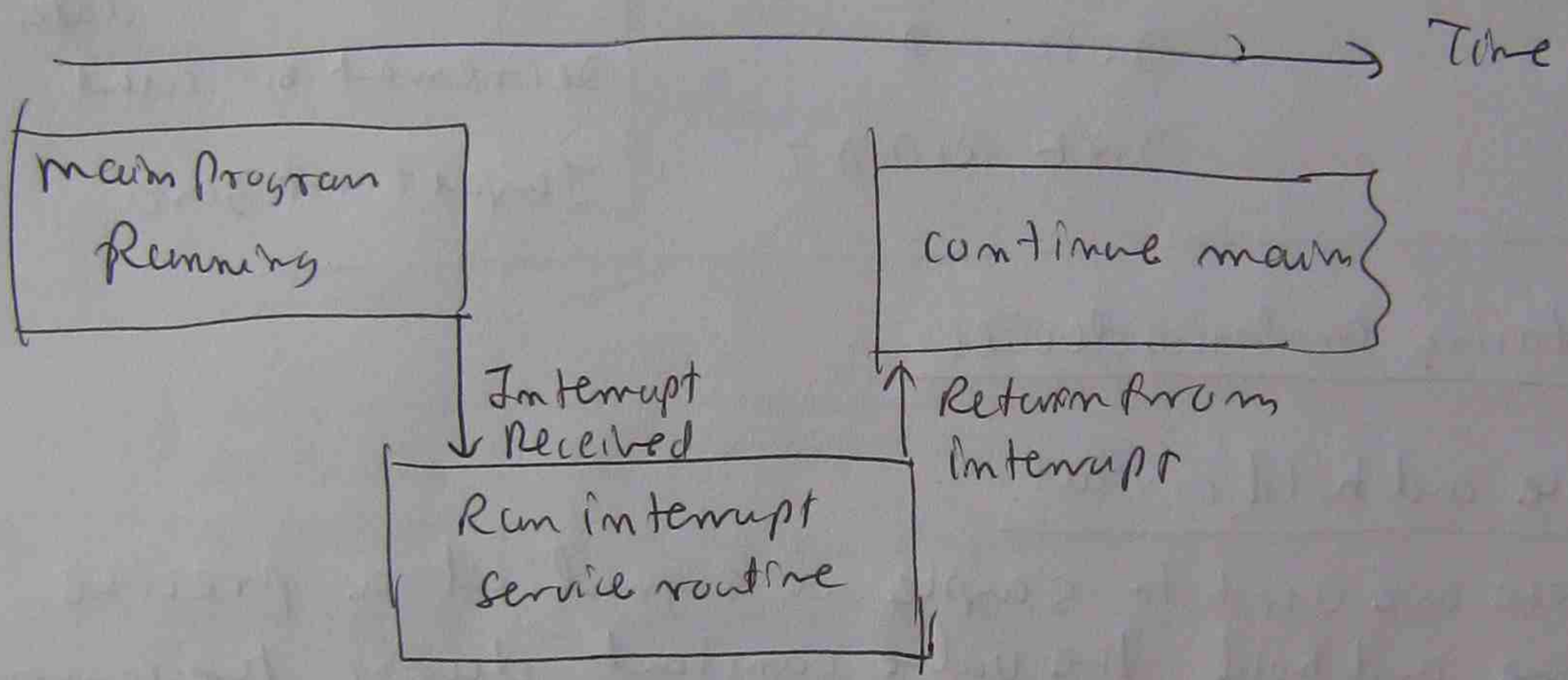
These devices permit one analogue signal out of several to be selected by logical control signals.

(3) Real time clock

Signal sampling and construction is often performed in conjunction with an interrupt driven real time clock.

Interrupts

To enable a peripheral device to inform the microprocessor when it wishes to transfer data. On receipt of the interrupt, the microprocessor temporily suspends its current activity, performs the required input or output operation, and then returns to its previous task.



Interrupt

(1) Save the contents of the Program counter on the System stack

(2) Load the program counter with the start address of the interrupt service routine

(3) Run the Interrupt Service routine

(4) Finally, return control to the interrupted Program by restoring the saved contents of the Program counter from the system stack.

circular buffer

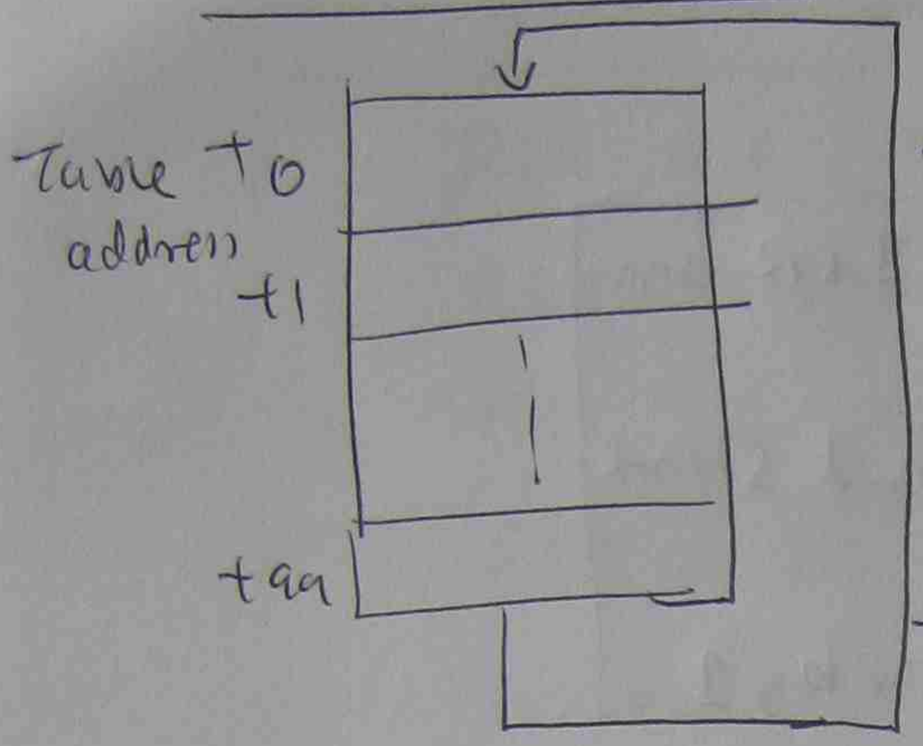
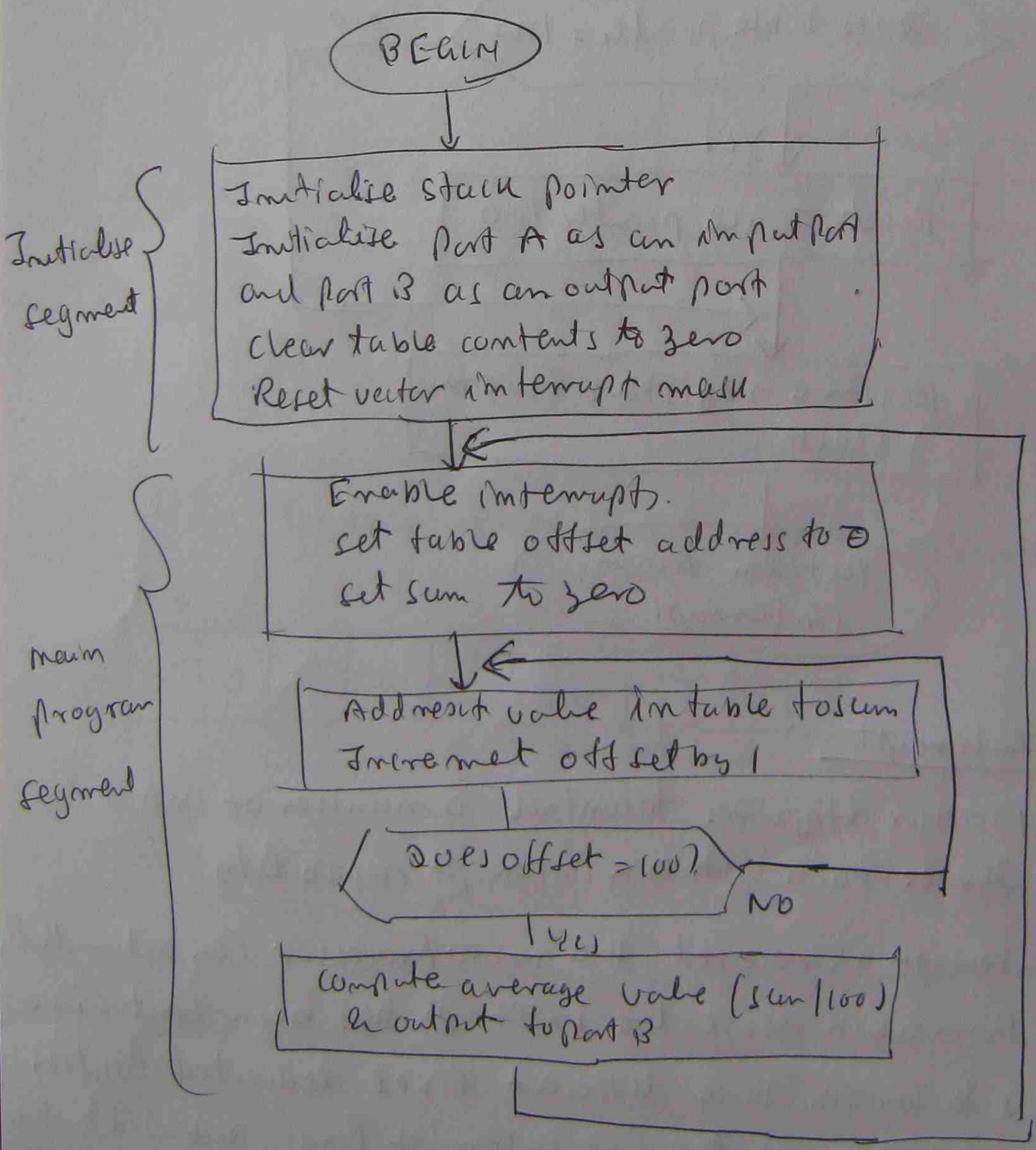


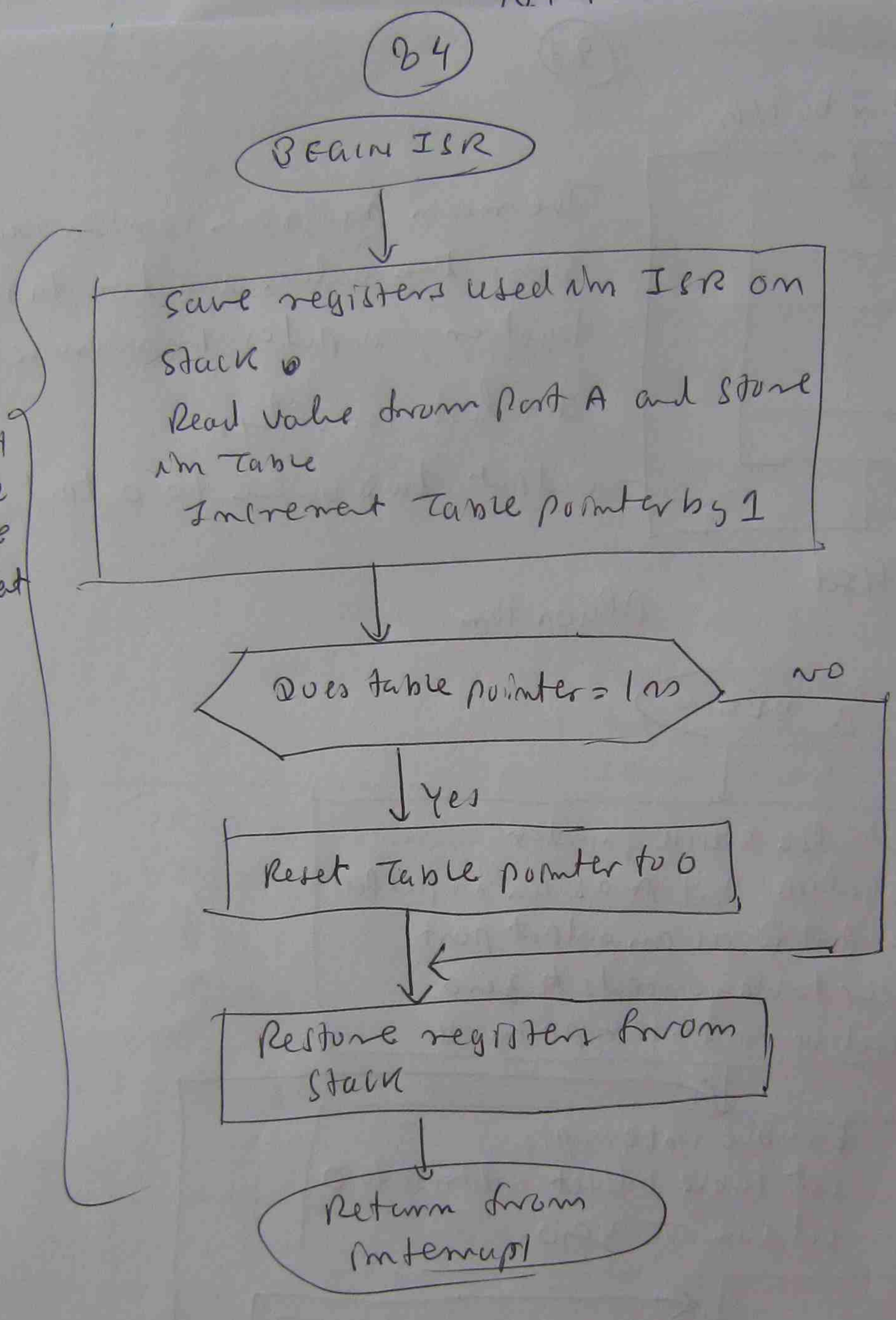
Table offset

- The main program continuously sums the values in the table together, computes the average of the last 100 values.
- output this value to output port.

Algorithm.



(b) 4



Multiple Interrupt

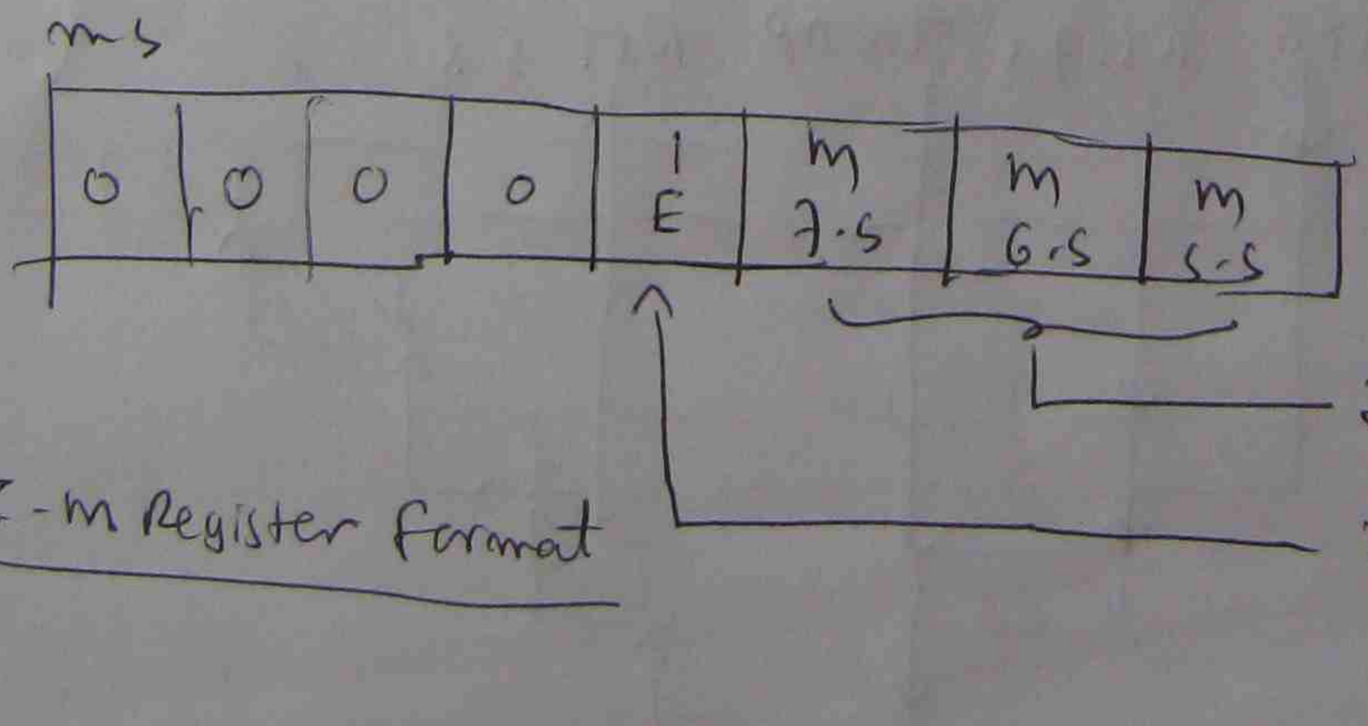
The microprocessor requires to control a number of input (or) output devices each with own interrupt capability.

When an interrupt is received, the microprocessor can automatically determine from which device the interrupt has been sent since it is caused to branch to a different fixed dedicated locations in memory for each of the five interrupt lines. Dedicated locations — vector address

Interrupt Input	Vector address
RST 4.5 (TRAP)	0024 (hex)
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C
INTR	The vector address for this input is not fixed and is part of the instruction placed on the data bus by the interrupting device when the interrupt request is acknowledged

RST 4.5 (TRAP) Highest Priority
RST 7.5
RST 6.5
RST 5.5
↓
Lowest Priority

EI - Enable Interrupt, DI - Disable Interrupt



I-m Register Format

SIm - set interrupt mask

The A register is first loaded with the required mask bits in the three least significant bit positions together with a logical 1 in the 4th bit and the SIm instruction is then given

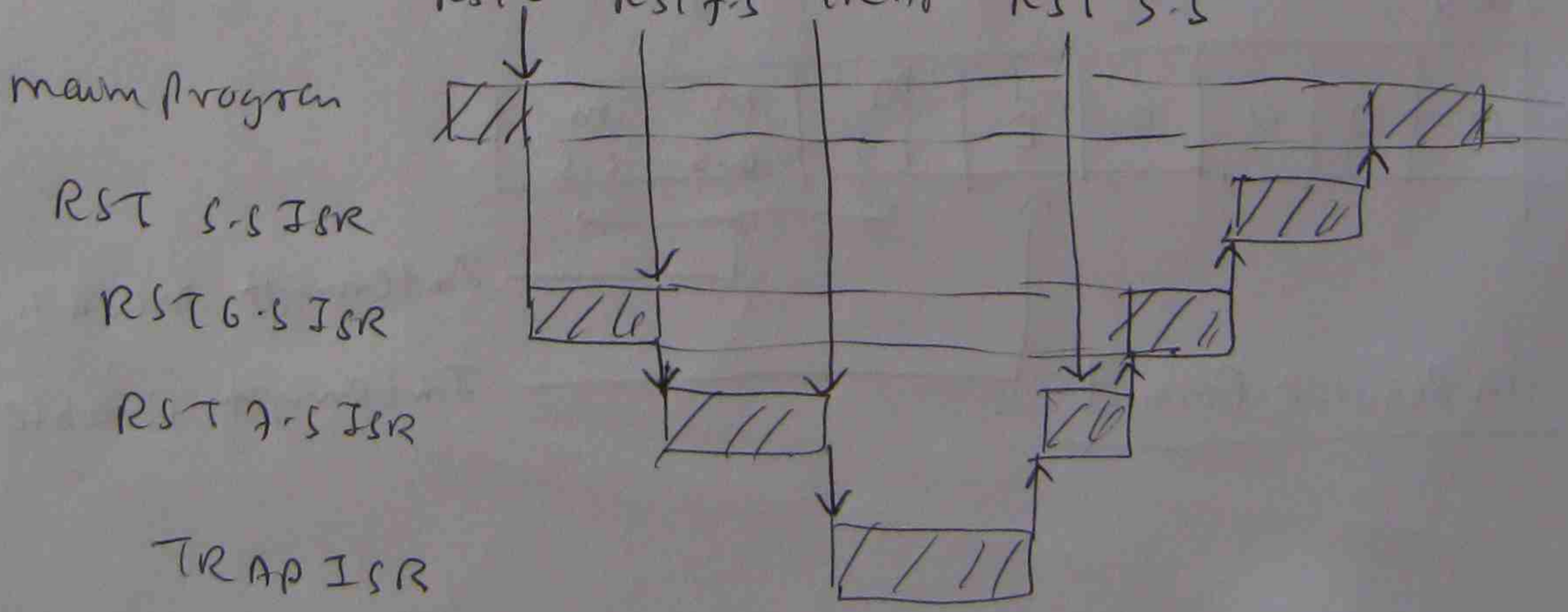
```
MVI A,08
SIM
H
```

RIm - Read Interrupt mask } After a RIm instruction the least significant 3 bits of A register indicate the state of the corresponding mask bit

Interrupt Priority Levels

Priority assignment of different interrupt inputs.

		Priority
TRAP	Power failure	HI
RST 7.5	Over temperature alarm	
RST 6.5	Real time clock	
RST 5.5	Read rear temperature set value	LO



Running

Suspended

	Assembly Instruction		comments
main program	BEGIN	MVI A, 02 SIM EI	Reset all mask bits and enable Interrupts
S.S Interrupt Service Routine	ISR S.S	PUSH PSW MVI A, 00 SIM EI RET	Disable S.S Interrupt & enable 6.S & 7.S Interrupts.
6.S Interrupt Service Routine	ISR 6.S	PUSH PSW MVI A, 0B SIM EI RET	Disable S.S and 6.S Interrupt and enable 7.S Interrupts.
7.S Interrupt Service Routine	ISR 7.S	PUSH PSW MVI A, 0f SIM EI RET	Disable S.S, 6.S and 7.S Interrupts.

Use of micro processor with interrupt

Ex 7.3 1 The program first initialize an area of memory as a stack

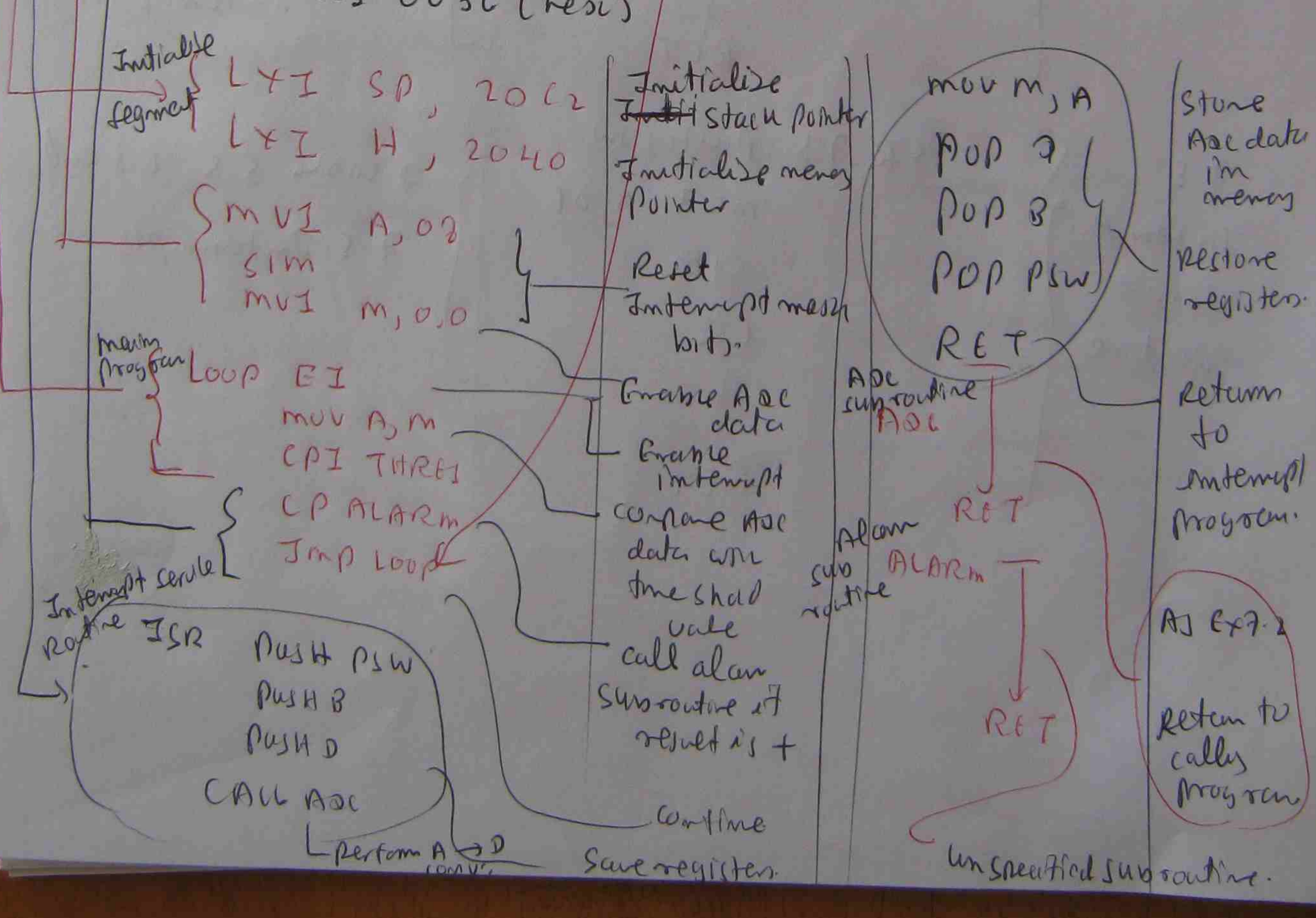
2 Reset the interrupt mask bits.

3 Then continuously compare the data returned by subroutine ADC. with a preset threshold value. (THRES)

4 If this value is equalled or exceeded, subroutine alarm is called.

5 Analogue to digital conversion is performed on receipt of interrupt on RST.

6 The instruction JMP ISR must therefore be stored in memory starting at the interrupt vector address 003C (hex)



Application Examples

Micro processer applications →
- sequence controller
- digital clock, timing sequence
- digital to analogue converter
- Analogue to digital converter

Basic Sequencing

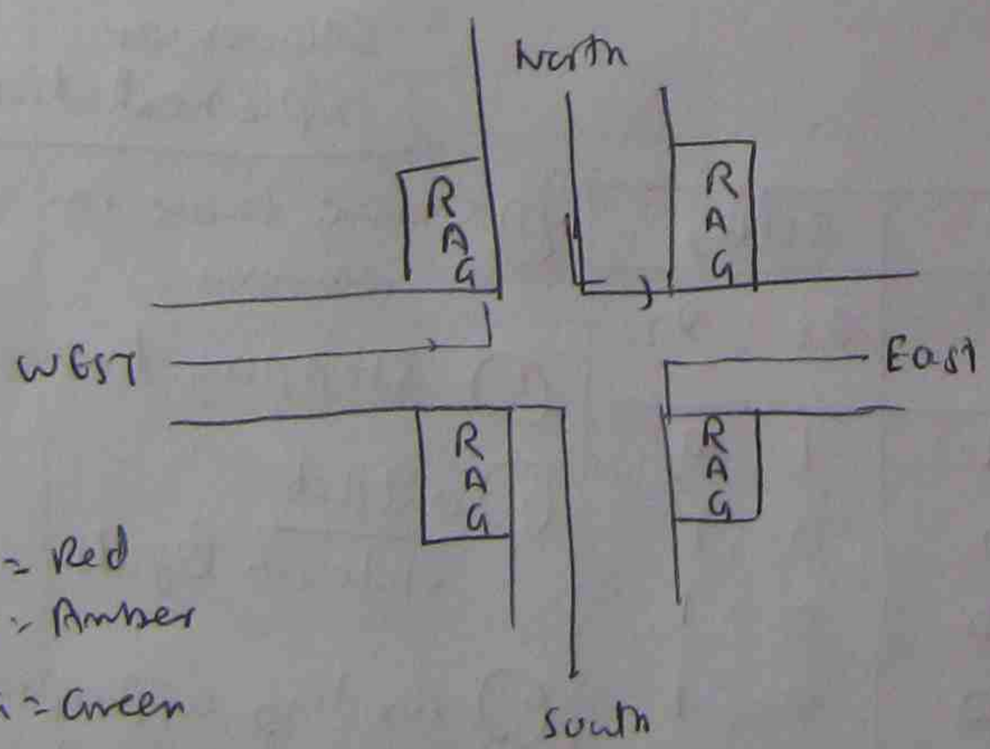
- The instructions provided by a microprocessor to input & output logical data to and from an external device.
- Programmable input/output (PIO) port may be used to provide the necessary latching (&) isolation functions.

Thermostat → Logical Input

Logical output → Relay (ON/OFF control)

A sequencer activates a number of devices in a preset sequence with a preset time delay between each new device state.

Ex(1) Traffic light sequencer



I SEQUENCING

- Sequence

Red, Red + Amber, Green, Amber

Red again

North & South - change in the same sequence

East & West - change in the same sequence.

Traffic light sequence table

	North / South			East / west		
	Red	Amber	Green	Red	Amber	Green
7 state 0 delay 1	ON	OFF	OFF	OFF	OFF	ON
1 delay 2	ON	OFF	OFF	OFF	ON	OFF
2 delay 2	ON	OFF	OFF	ON	OFF	OFF
3 delay 2	ON	ON	OFF	ON	OFF	OFF
4 delay 1	OFF	OFF	ON	ON	OFF	OFF
5 delay 2	OFF	ON	OFF	ON	OFF	OFF
6 delay 2	ON	OFF	OFF	ON	OFF	OFF
	ON	OFF	OFF	ON	ON	OFF

Light ON = 1 Light OFF = 0

delay 1 presents → $P_1 = 1$ delay 1 is not present $P_1 = 0$

delay 2 presents → $P_2 = 1$ delay 2 is not present $P_2 = 0$

delay 1 = Long delay between overall direction changes (~2 mins)

delay 2 = Short delay between transitional light settings (~3 sec)

Traffic light state table

State	N/S	E/W			delay	
	R	A	G	R'	A'	G'	D_1	D_2
0	1	0	0	0	0	1	1	0
1	1	0	0	0	1	0	0	1
2	1	0	0	1	0	0	0	1
3	1	1	0	1	0	0	0	1
4	0	0	1	1	0	0	1	0
5	0	1	0	1	0	0	0	1
6	1	0	0	1	0	0	0	1
7	1	0	0	1	1	0	0	1

Sequencer Implementation

① Store table in memory

② stepping through it

③ output / state of lights.

④ writing appropriate delay time between steps

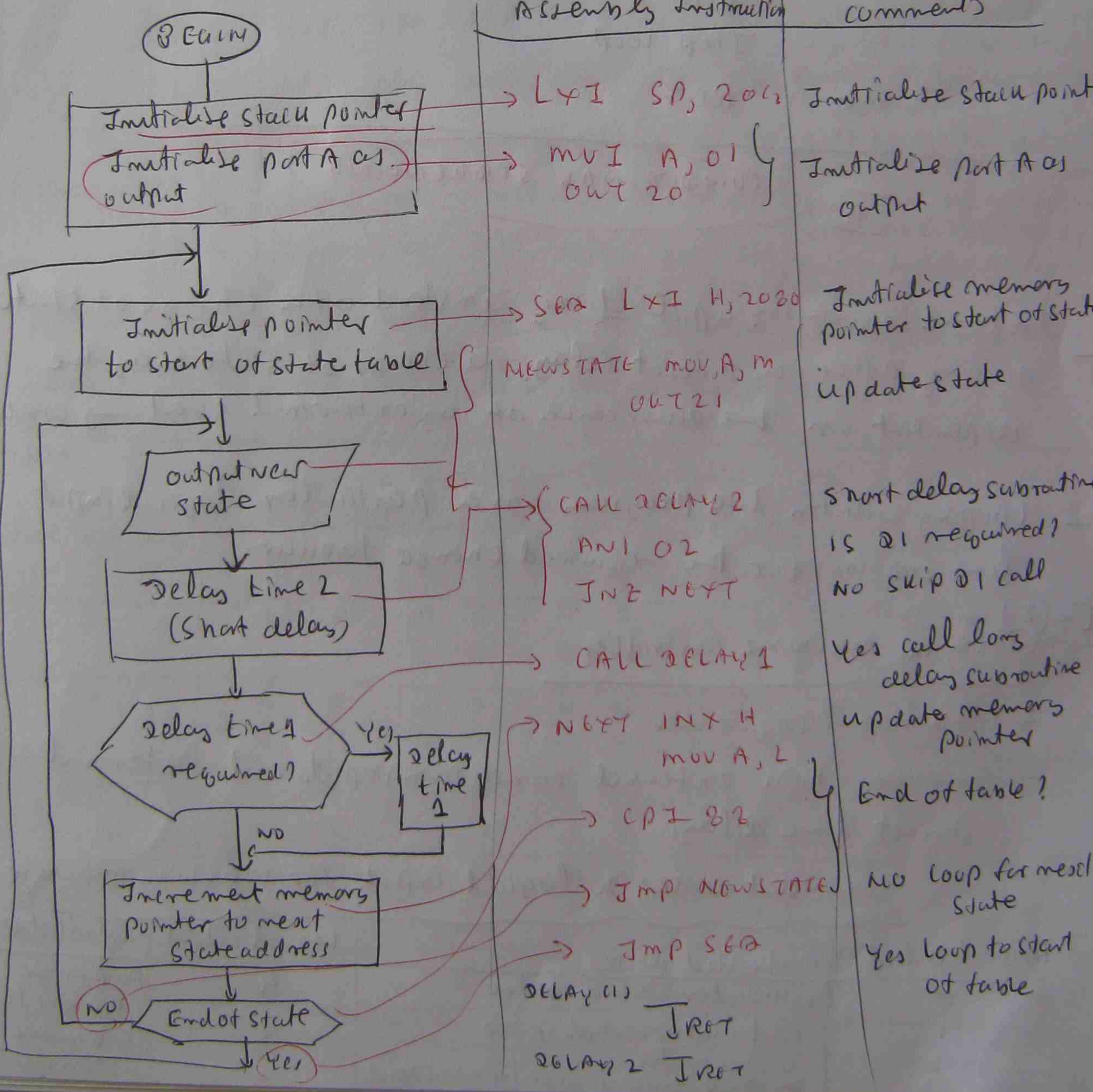
8155 PIO

- most significant 3 bits of port A device arrive North/south lights (RAG)
- The next most significant 3 bits arrive the East / west lights (R', A', G')

Relay time;

Stage 1 – New state, a delay of D_2 is executed

Stage 2 – if required a further delay of $D_1 - D_2$ is executed.



DELAY (1)/(2)

```
TIM DLY     MVI A,00
LOOP        CMP C
            JZ TIME
            NOP
            NOP
            NOP
            NOP
            NOP
            DCR C
            JMP LOOP
TIME        RET
```

II CONDITIONAL SEQUENCING

In some sequencing applications, instead of a change of state occuring after a preset delay, a change of state may be dependent on the occurance of an external event. → conditional sequencing

- Looping within the program on a particular logical input line waiting for the required change to occur.

EX(2)
washing machine controller

- conditional sequencer
- Involves both external condition inputs and internal Preset time delays.

controlled devices & logical inputs for washing machine

Logical output	controlled device	Logical output	controlled device
0	Hot water control valve	4	Pump motor (emptying tub)
1	cold water control valve	5	Tub motor spin speed
2	water heater		
3	Tub motor/ wash/rinse speed		

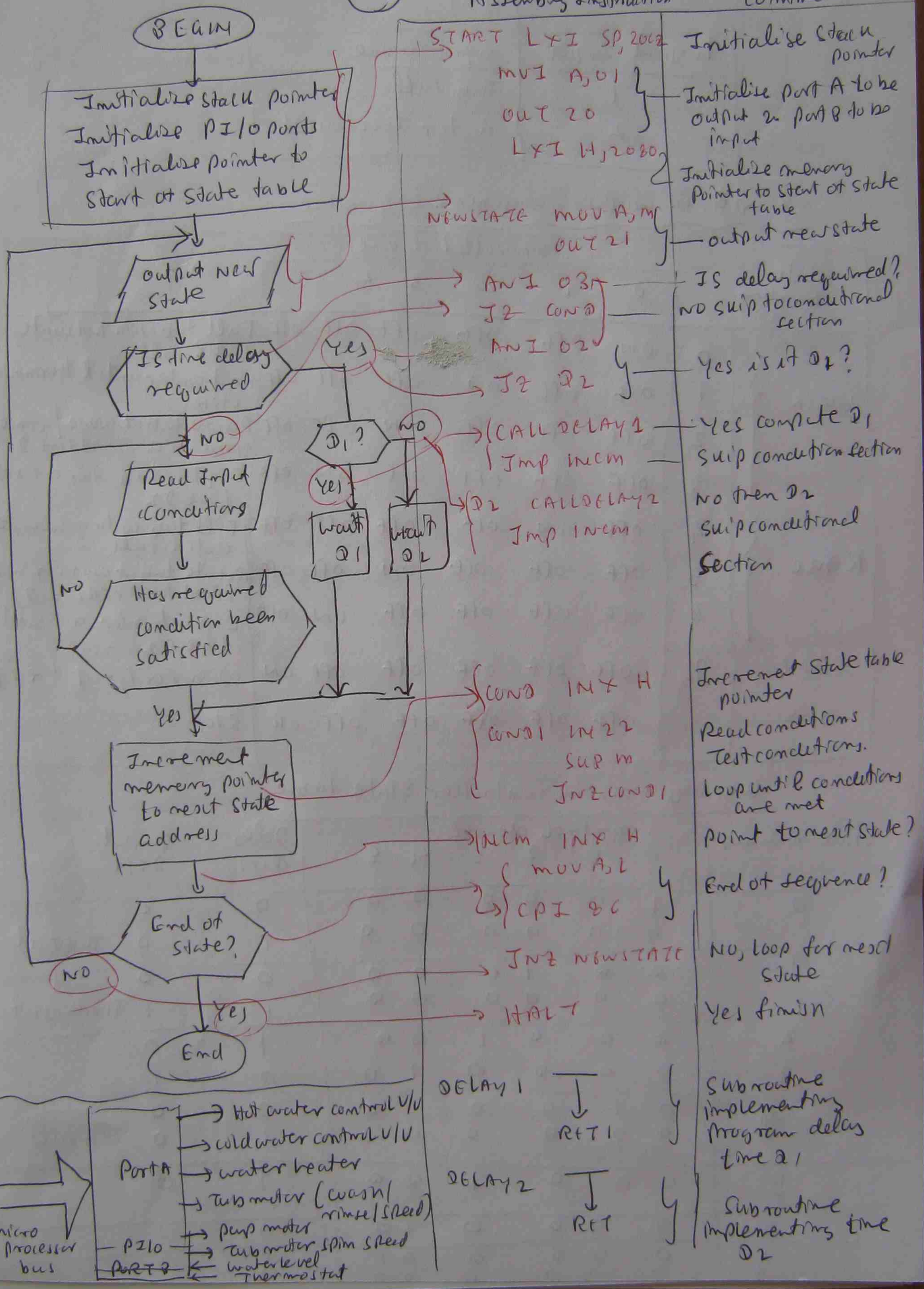
Logical Input	significance
1	Tub full
2	water thermostat

simplified washing machine sequence

State Number		controlled device 0	1	2	3	4	5	Action
Wash	0	ON	Off	Off	Off	Off	Off	Fill tub with hot water until full
	1	Off	Off	ON	Off	Off	Off	Heat water until thermostat closes.
	2	Off	Off	Off	ON	Off	Off	Rotate tub at wash/rinse speed for a fixed time D1
	3	Off	Off	Off	Off	ON	Off	Empty tub for a fixed time D2
Rinse	4	Off	ON	Off	Off	Off	Off	Fill tub with cold water until full
	5	Off	Off	Off	ON	Off	Off	Rotate tub at wash/rinse speed for fixed time D1
	6	Off	Off	Off	Off	ON	Off	Empty tub for a fixed time D2
SPIN	7	Off	Off	Off	Off	Off	ON	spin for fixed time D3
OFF		Off	Off	Off	Off	Off	Off	Stop

Washing machine controller state table

State number	control device 0	1	2	3	4	5	Delay D1/1	Input D2/2	
0	1	0	0	0	0	0	0	0	
	0	0	0	0	0	0	1	0	Tub full
1	0	0	1	0	0	0	0	0	
	0	0	0	0	0	0	1	1	Thermostat
2	0	0	0	1	0	0	1	0	
3	0	0	0	0	1	0	0	1	
4	0	1	0	0	0	0	0	0	
	0	0	0	0	0	0	1	0	Tub full
5	0	0	0	1	0	0	1	0	
6	0	0	0	0	1	0	0	1	
7	0	0	0	0	0	1	1	0	
8	0	0	0	0	0	0	0	1	


BEGIN
Initialize stack pointer
Initialize PI/O ports
Initialize pointer to start of state table
output NEW state
Is time delay required
Yes
NO
Read Input conditions
NO
Has required condition been satisfied
Yes
Increment memory pointer to next state address
End of state?
NO
Yes
End
D1?
Yes
NO
wait D1
wait D2
Hot water control V/V
cold water control V/V
water heater
Tub motor (wash/ rinse/ SPeed)
pump motor
Tub motor Spin SPeed
water level
Thermostat
Port A
PZIO
PORT B
micro Processor bus
START LXI SP, 2002
MVI A, 01
OUT 20
LXI H, 2030
NEWSTATE MOV A, M
OUT 21
ANI 03H
JZ COND
ANI 02
JZ D2
CALL DELAY 1
JMP INCM
D2 CALLDELAY2
JMP INCM
COND INX H
COND1 INX H
SUB M
JNZ COND1
INCM INX H
MOV A, L
CPI 8C
JNZ NEWSTATE
HALT
DELAY 1
RET 1
DELAY 2
RET
Assembly Instruction
comment
Initialise stack pointer
Initialise Port A to be output & Port B to be input
Initialize memory Pointer to start of state table
output new state
IS delay required? NO skip to conditional section
Yes is it D1?
Yes compute D1 skip conditional section
NO then D2 skip conditional section
Increment State table pointer
Read conditions
Test conditions.
Loop until conditions are met
Point to next state?
End of sequence?
NO, loop for next state
Yes finish
Subroutine implementing Program delay time D1
Subroutine implementing time D2

11) Programmable Timer

Intell 8155 - Single integrated circuit which incorperates a programmable timer with a 256 x 8 bit state RAM and 3 programmable input/output ports

- 14 bit counter — Programmed to generate either a square wave of a selectable period (or) terminal count pulse.

Tmax ≤ 8.1915 ms

Timer may be programmed either to stop after the terminal count pulse is generated (OR) to continue counting and hence generate a new count pulse every, say, 8.1915 ms.

Digital clock Ex(3)

use the programmable timer to continuously generate a count pulse every 5 ms and to connect the timer output to one of the interrupt lines (RST 7.5) of the microprocessor.

COUNT - contains the current number of interrupts since the last change in SECS

SECS - contains the two BCD digits corresponding to seconds. The contents are incremented by 1 each time time COUNT reaches 200.

MINS - contains the two BCD digits corresponding to minutes. The contents are incremented by 1 each time SECS reaches 60.

HOURS - contains the two BCD digits corresponding to hours. The contents are incremented by 1 each time MINS reaches 60.

LXI SP, 20C2 ← Initialise stack pointer
LXI H, 2030
MVI M, 00 } clear count
INX H
MVI M, 00 } Sec min to 0
INX H
MVI M, 00
INX H
MVI M, 12 } Preset Hr=12

MVI A, 10 } Initialise count for 5ms
OUT 24
MVI A, 27
OUT 25

MVI A, C3 } Initialise Port A, B as o/p & timer
OUT 20
MVI A, 0B } Reset interrupt mask unit
SIM

Loop EI
LXI H, 2032 } o/p to Port B
OUT 22
INX H
MOV A, M } Load Hr digit to A o/p to port A
OUT 21
JMP LOOP

ISR

Initialise stack pointer

Add 1 to count

IS COUNT 200?

NO

Yes

Reset count to 0
Add 1 to SECS & adjust for decimal

Is SECS = 60?

NO

Yes

Load min: digit to A

Reset SECS to 0
Add 1 to MINS and adjust for decimal

IS MINS = 60?

NO

Yes

Reset MINS to 0
Add 1 to HOURS & adjust for decimal

IS HOURS = 24

NO

Yes

MVI M, 00
Yes reset Hr to 0

RET

Return from interrupt

ISR PUSH PSW } save regist:
PUSH H
LXI H, 2030 } Add 1 to count
INR M
MVI A, C8 } IS count = 200?
CMP M
JNZ End

BCD → digit from BISS → BCD to 7 segment decoder driver → 7 seg LED → Display

MVI M, 00
Yes reset count to 0

INX H
MOV A, M
INR A
DAA
MOV M, A } Add 1 to SECS for decimal

MVI A, 60 } IS SECS = 60?
CMP M
JNZ END no end

MVI M, 00

INX H
MOV A, M
INR A
DAA
MOV M, A } Add 1 to MIN & adjust for decimal

MVI A, 60
CMP M } IS MIN = 60?
JNZ END no end

Interrupt Service routine
EMPIM

MVI M, 00
INX H
MOV A, M
INR A
DAA
MOV M, A } Add 1 to HOURS & adjust for decimal

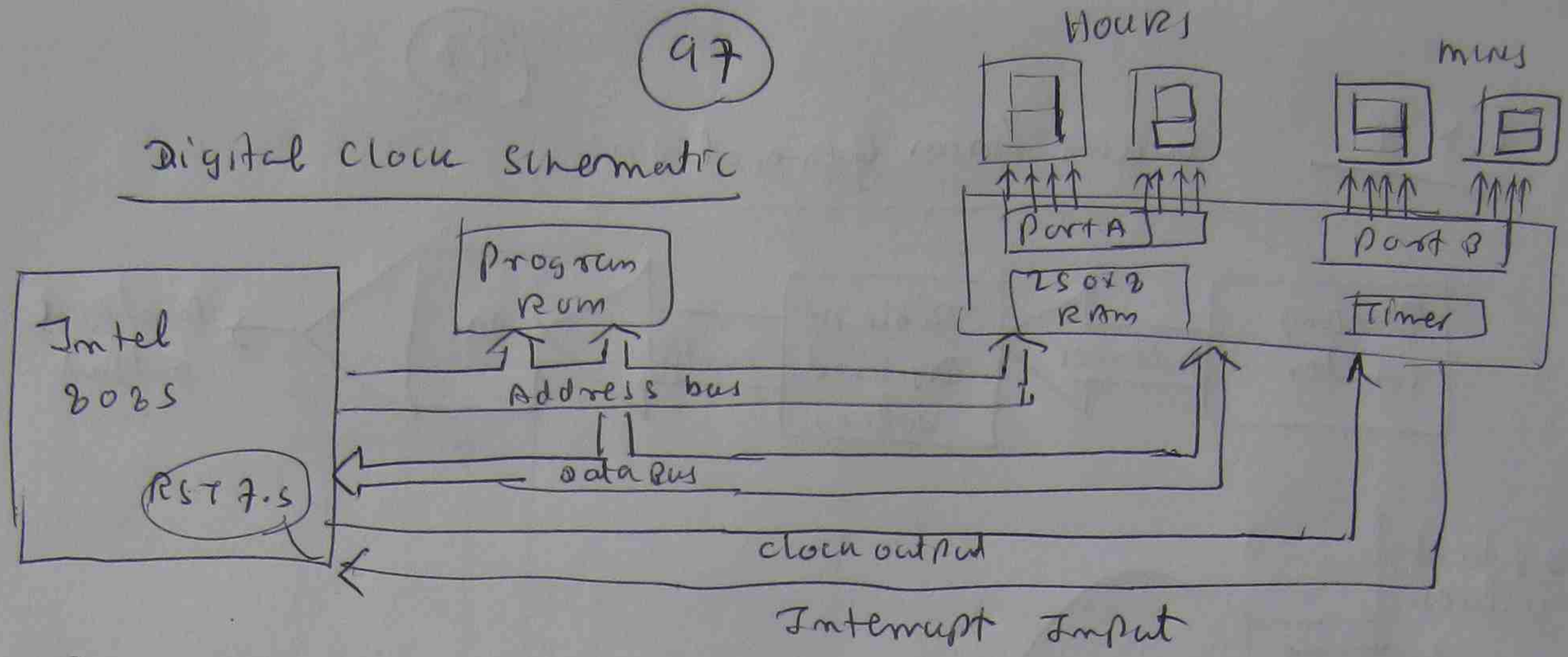
MVI A, 24 } IS HR = 24?
CMP M
JNZ END NO end

Yes reset Hr to 0

MVI M, 00

END } Restore register content
PUPH
PUPPSW
RET

Digital clock schematic



RAm addresses

2000 - Stack pointer
2080 - count
2081 - secs
2082 mins
2083 Hours

8155 addresses

command Register 20 (hex)
Port A data 21
Port B data 22
low order byte of count length 24
High order byte of count length 25

IV WAVE FORM GENERATION

Direct wave form

- Use of DAC to convert the digital output from a microprocessor into an analogue form
- use of look up table.
- Micro processor was used as a counter
- Its contents were incremented by unity with in a program loop
- The contents were output to DAC after each count increment.

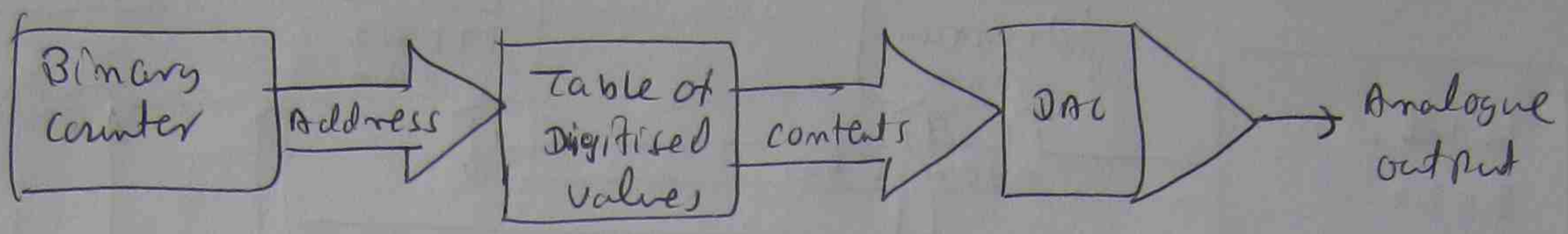
Alternative wave form

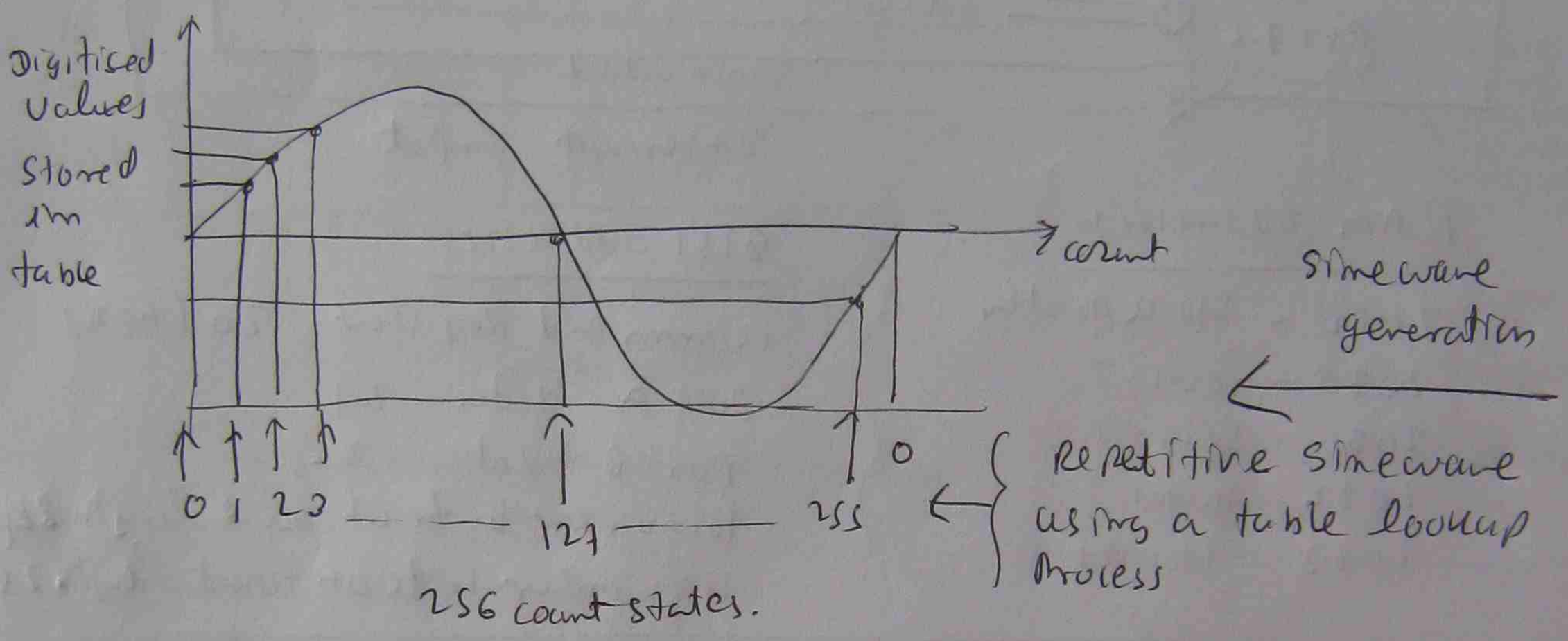
- The contents of the counter not to drive the DAC directly
- To provide addresses to sucessive memory locations, the contents of which store the required digital value which is to be output to the DAC.

Table — The contents of the block of sucessive memory locations
Table look up process.

DAC - 8 bits, 256 count states.

Ex 4 — Wave form generation

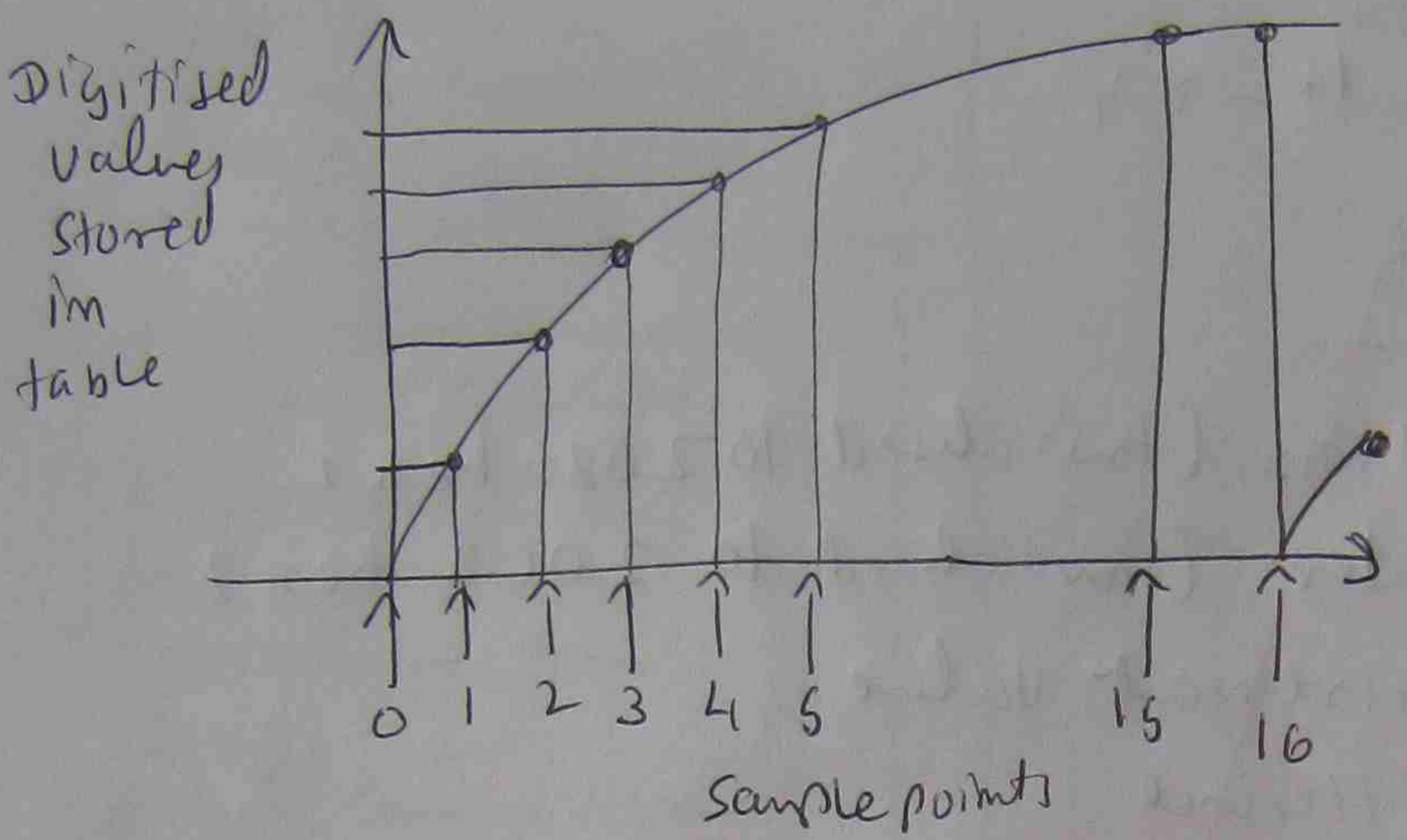


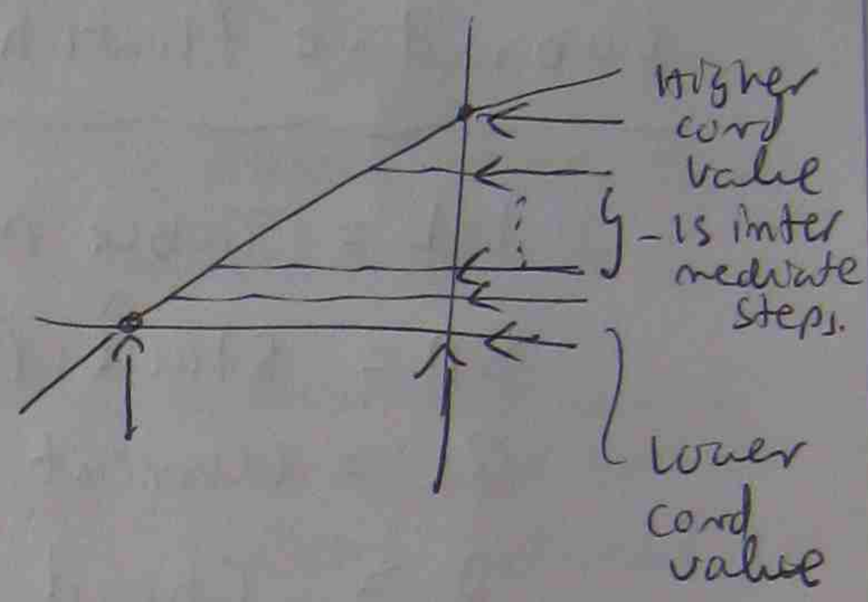


- The table is stored in memory starting at address 2000 and
  - Register pair HL is initialized to this value.
  - Register L is 8 bit counter & the combined contents of register pair HL.
  - Provide complete 16 bit memory address automatically.
- On reaching FF, the contents of register L will return to 00
- the combined contents of HL return to 2000 & the process will repeat.

Assembly Instructions		Comments
	MVI A, 01	Initialize port A as an output port
	OUT 20	
	LXI H, 2000	Initialize register pair HL to point to start of table
LOOP	MOV A, M	Lookup value from table
	OUT 21	output value to DAC
	INR L	Increment table offset
	JMP LOOP	

V Piece wise Linear interpolation





↑ Piecewise linear interpolation ↑

↑ Linear interpolation process ↑

17 stored values → Chords

15 values in between → Steps
adjacent chords

Intermediate values between adjacent chord values are obtained by first evaluating the chord difference.

Ex 5 The program generates an exponential wave form using a table look up process and piece wise linear interpolation.

- To find intermediate value, it maintains a current increment value (Chord x n) and simply adds the cord difference to this following each iteration.
- The current increment value is held as a 16 bit number in register pair BC.
- The combined contents are divided by 16 to form each intermediate value.
- (Shifting the current increment value four places to the right. The process is performed in separate subroutine)

Subroutine

IF (B) = 0x (hex) (ms 4 bits of B are always zero)

(C) = YZ (hex)

Then (A) ← result - XY

Subroutine flowchart

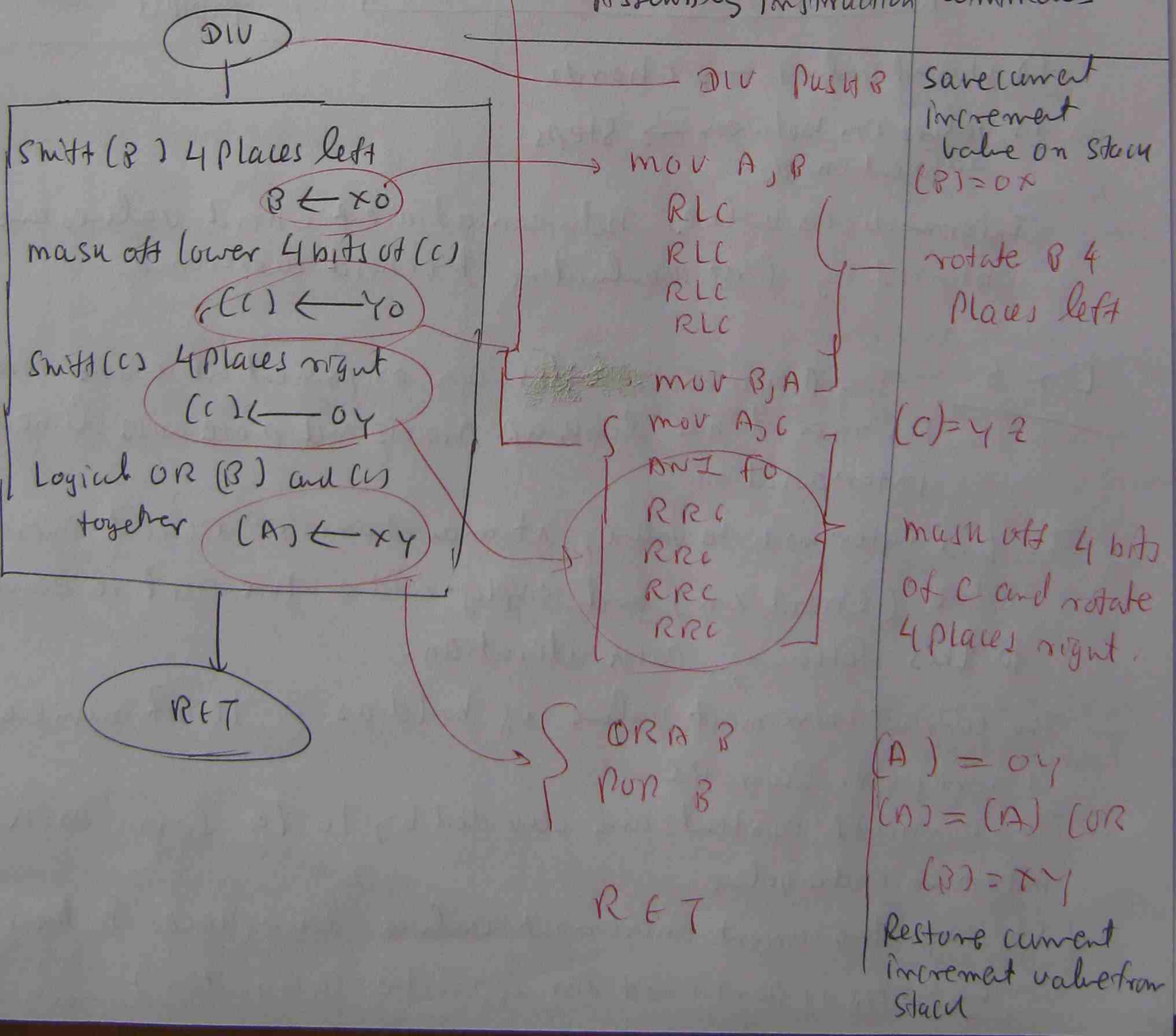
HL = Table Address (initialised to 2080 hex)

SP = Stack Pointer (initialised to 20C2 hex)

BC = Current Increment value

D = Chord Difference

E = Step count



101

BEGIN

Initialise Table Offset to zero

Set current Increment to zero

Read lower & higher Chord values from table

Compute Chord difference

Initialise stepcount to 15

Add Chord Difference To temporary Increment value

Divide result by 16 (subroutine DIV) and add to lower Chord value

Output result to DAC

Decrement stepcount

NO — Is stepcount = 0?

Yes

Increment Table Offset

Is offset = 16? — NO

Yes

```
LXI SP, 20C2
MVI A, 01
OUT 20
```

Initialise Stack Pointer

Initialise Port A as an output Port

```
START LXI H, 2020
      LXI B, 0000
```

Initialise HL to start of Table

clear BC

```
CONT  INR L
      MOV A, M
      DCR L
      SUB M
      MOV D, A
```

Obtain Higher Chord value from table

Substract lower Chord value to from Chord Differenc & save D

```
      MVI E, 0F
```

Set setupcount to 15

```
STEP  MOV A, C
      ADD D
      MOV C, A
      JNC NXT
      INR B
```

Add Chord Difference to 16 bit current Increment value

```
NXT   CALL DIV
      ADD M
```

Divide current Increment value by 16 & add to lower Chord value

```
      OUT 21
      DCR E
      JNZ STEP
      INR L
      MOV A, L
      CPI 10
      JNZ CONT
      JMP START
```

output value to DAC

Test if step count = 0

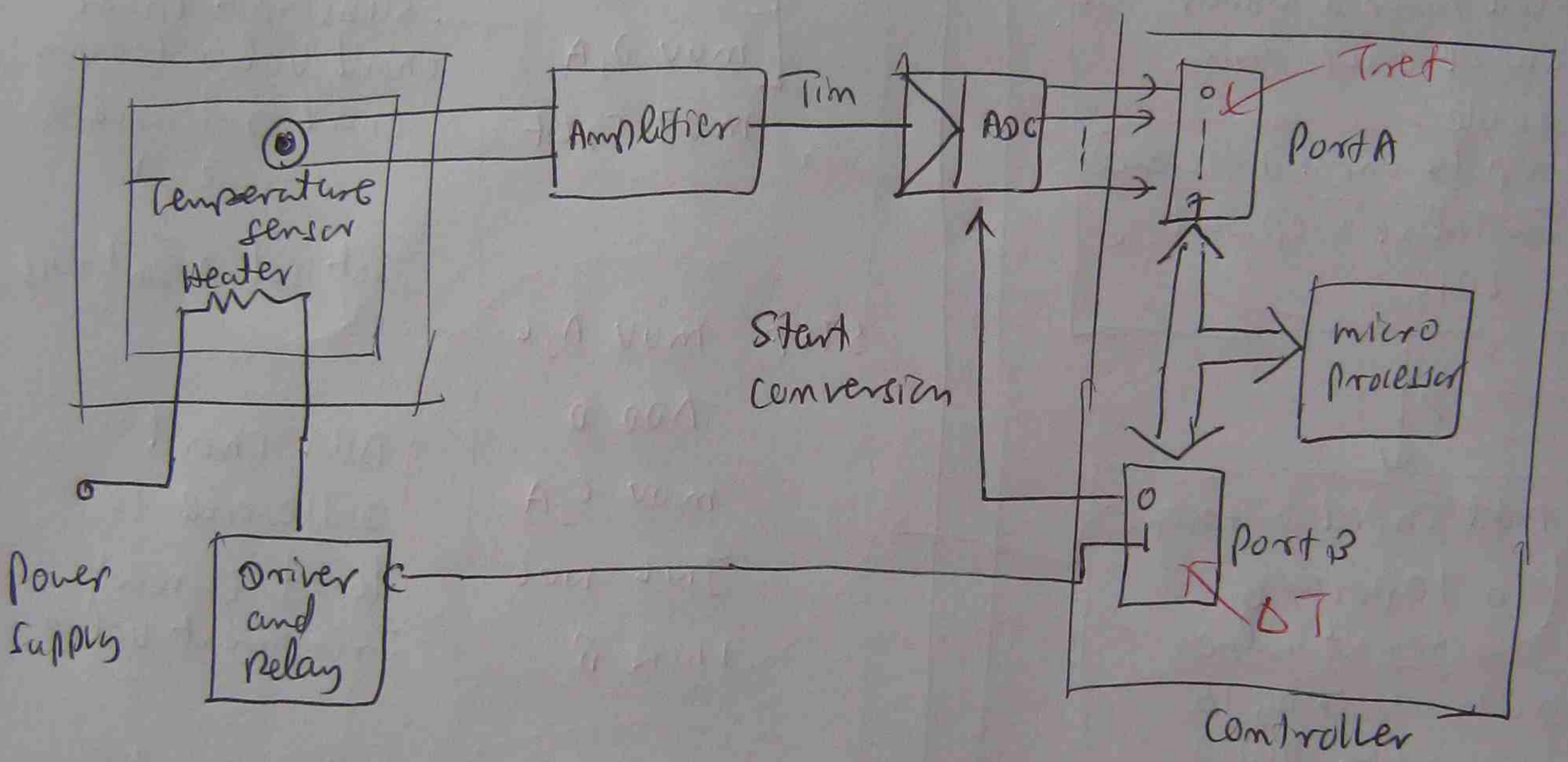
Increment Table Offset & test if = 16

NO continue

Yes - Reinitialise Table offset & repeat

Temperature controller

- The required temperature Tref is variable and is stored in a location in RAM.
- The tolerance limits on the required temperature $\pm \Delta T$ are also assumed variable in stored in a second RAM Location.



- The controlled temperature is controlled by first deriving an analogue voltage Proportional to the temperature - from a thermistor & associated amplifier.
- converting this voltage into an 8 bit digital value by means of an ADC
- supply to heater is controlled by a single digital output from the microprocessor via a suitable driver & relay

1 ⟶ heater on 0 ⟶ heater off

If Input $T_{im} > T_{ref} + \Delta T \Longrightarrow$ Heater is ~~off~~ turned off

If Input $T_{im} < T_{ref} - \Delta T \Rightarrow$ Heater is turned on

BEGIN

Start conversion and wait convert time

Read digitised ~~output~~ input temperature from ADC

Tim < (Tref + ΔT) ?

NO

Yes

Switch heater ON

T im > (Tref + ΔT)

NO

Yes

Switch heater Off

Temperature controller flow chart

Assembly Instruction	comments
LXI H, 2000	
MOV A, M	Read Tref from A2
MOV E, A	Save in E
INX H	
SUB M	Form (Tref − ΔT) and
MOV B, A	Store in B
MOV A, E	
ADD M	Form (Tref + ΔT)
MOV C, A	and Store in C
MVI A, 02	Initialise Port A
OUT 20	as input
MVI A, 00	Port B as output
OUT 22	Switch heater off
START MOV A, D	Recall state of Port B
ORI 01	Switch bit 0 on
OUT 22	
ANI 02	Switch bit 0 off
OUT 22	Start ADC
CALL DELAY	wait conversion time
IN 21	Read Tim from ADC
CMP B	
JC HTON	If Tim < (Tref − ΔT)
JZ START	Jump to switch on (HTON)
CMP C	
JC START	If Tim < (Tref + ΔT)
JZ START	Jump to start
MVI A, 00	Switch heater off
MOV D, A	
OUT 22	
JMP START	
HTON MVI A, 02	
MOV D, A	Switch heater on
OUT 22	
DELAY	
JMP START	
RET	

Development Aids

- Single board system - Intel 8085 microprocessor microprocessor, system clock source, Random access memory for holding a system monitor program, a number of input/output ports, key pad, associated numeric display, bus control logic. application
- Program stored in ROM
- Development phase for program stored in RAM.

monitor command

Digit displayed are in hexadecimal code.

xxxx. xx
address field — data for content field

* Reset → Enable the user to force the monitor to restart from the beginning of the program

Substitute memory - Allow the user to examine the contents of successive memory location
If required, to modify its contents

Run - Execute a program which is already stored in RAM.
Run key is first pressed, followed by 4 digit start address.

Examine Register allows the users to display and if required, modify the contents of each of the microprocessor registers.

Single step - Enable the user to examine the state of the complete system.

To overcome the errors, additional Software / hardware are provided.

Assembly Language → [Assembler] → machine code

High level language → [compiler] → machine code.

Assembler

```
LDA  : SECS    Increment contents of memory
INR    A       Location with symbolic name SECS
STA    SECS    by unity

SECS DS 1      Define one storage location for SECS

LXI H SECS     This load the absolute value
                 in to register pair HL

INR    M       contents of memory location
                 SECS (2800 (hex)) are incremented
                 by 1

SECS EQU 2800H Define SECS to be
                 absolute value 2800 (hex)
```

macro

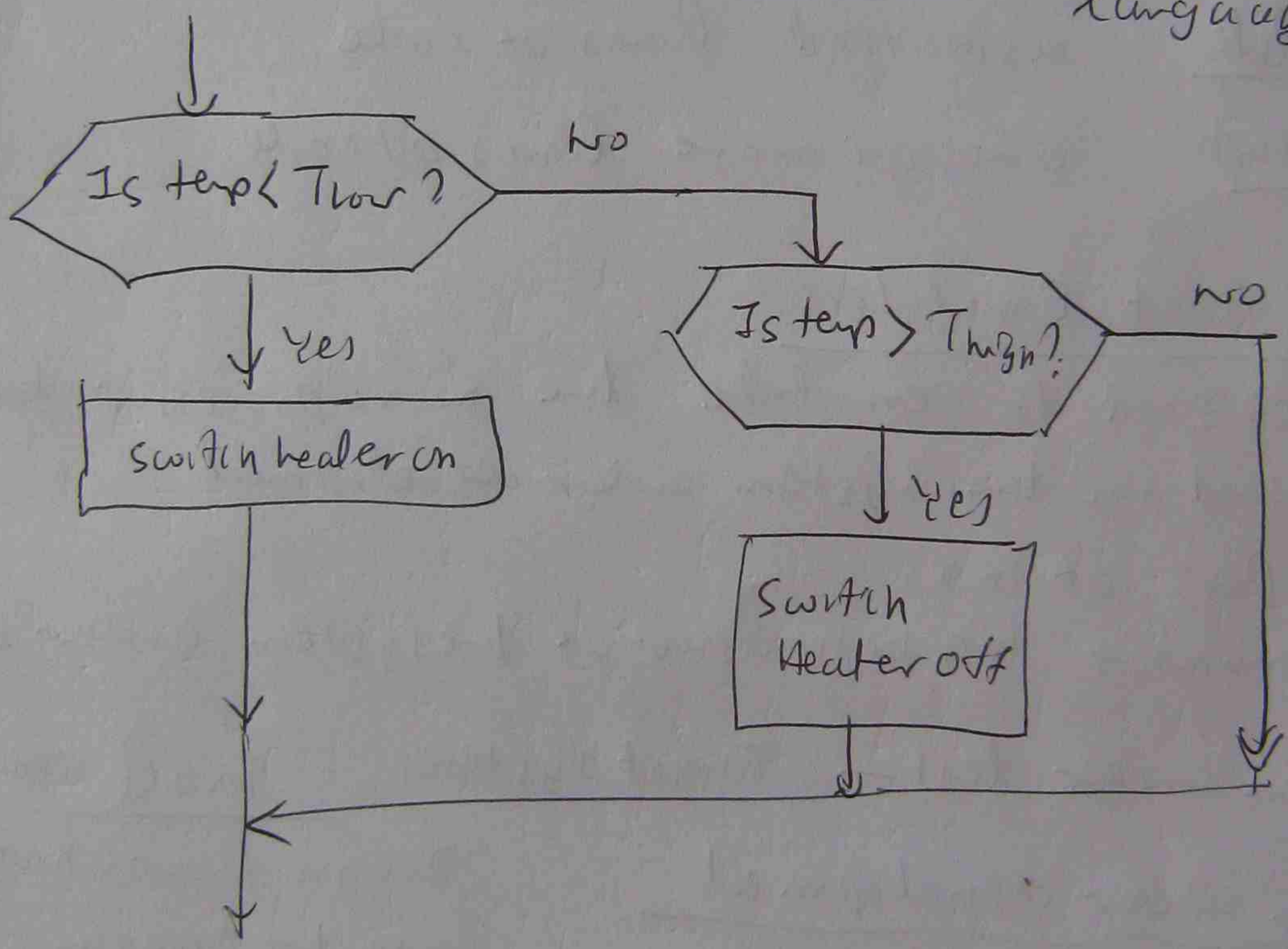
```
SHIFT : MACRO      start of macro SHIFT
        RRC
        RRC
        RRC
        RRC
        MEND       end of macro
```

To define the start address of compiler

```
ORG   2000H
List of Program
Instructions
  END
```

High level language → compiler → machine code

machine code → Interpreter → High level language.



IF TEMP < TLOW THEN HEATER : -1

ELSE IF TEMP > THIGH THEN HEATER :-0

Subroutine CALL DELAY (COUNT)

Editors

Enable Perser to readily modify Pe source Program code

Run interactively.

list a specified number of lines of source code on Pe visual display screen

move to a specified line in Pe source Program

delete a specified lines of code

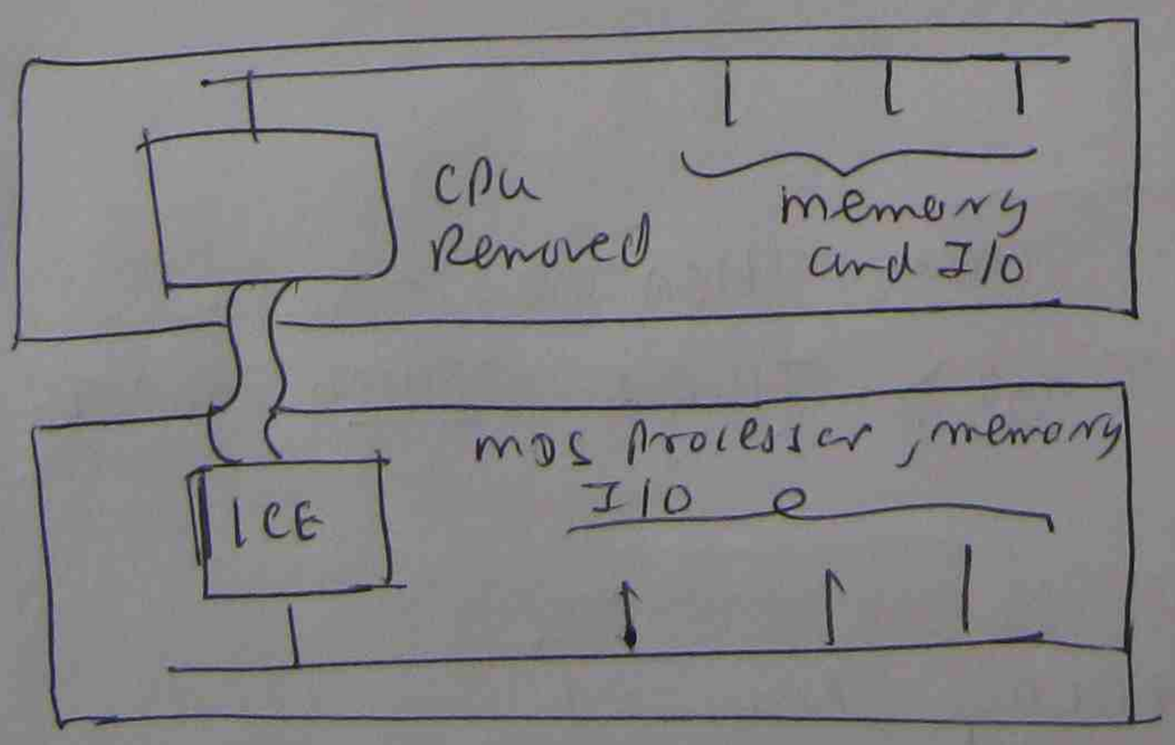
insert one or more lines of code.

In circuit Emulators

- designed to emulate Pe microprocessor being used in Pe system under development
- Part of mos.
- monitor Pe behaviour of Pe system under development

System under test — Target system

System under development



In circuit emulation

mos (Development System)

Design flowchart

code Pe program

Enter Pe program in to mos memory

Assemble / compile

Run Pe object program

Debug using ICE

Edit Pe source program