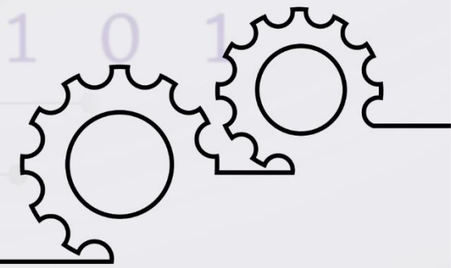
SIMATS
School of Engineering

Microprocessors and Microcontrollers

Electronics and Communication Engineering



Saveetha Institute of Medical And Technical Sciences,Chennai.

SAVEETHA SCHOOL OF ENGINEERING
DEPARTMENT OF ECE
CONCEPT MAP TOPICS

Engineer to Excel

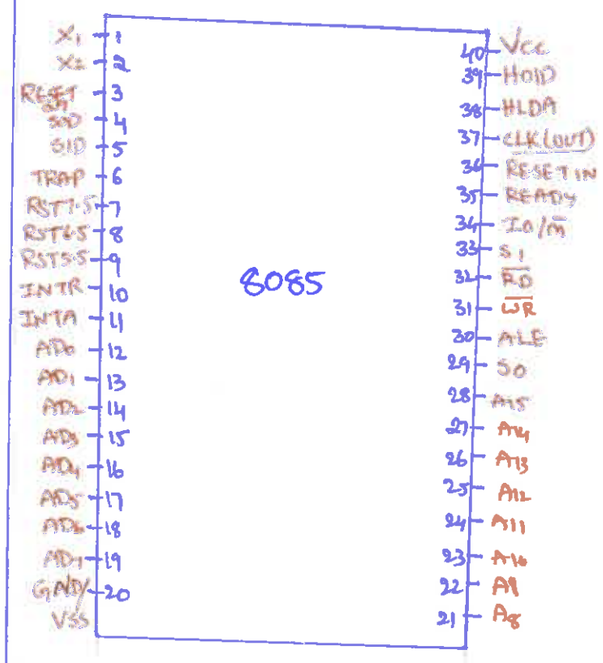
ECA10 Microprocessors and Microcontrollers

MICROPROCESSOR 8085 Pin Diagram & Registers

What is microprocessor?

* Computer's Central Processing (CPU) built on single IC (Integrated circuit) is called a microprocessor.
* Microprocessor consists of ALU (Arithmetic & Logic unit), Control unit and Registers.

8085 Pin Diagram :-



Some important Pins are:

AD0 – AD7 : multiplexed Address & Data lines.

A8 – A15 : Higher order Address lines.

ALE : Address latch enable is an output signal. It goes high (1) when operation started.

S0, S1 : Status signal used to indicate type of operation.

RD : used to read data from I/O devices on memory.
Read → Active Low

WR : used to write data on I/O Devices or memory
write → Active Low

READY : used to check the status of output Devices.
Low → Mp will wait till (0) high (1)

TRAP : After Enabled, restart occurs and execution starts from address 0024H
→ Highest Priority interrupt
→ Non maskable interrupt

RST5.5, RST6.5, RST7.5 :
* Maskable interrupt
* Low Priority than TRAP

INTR & INTA : INTR → interrupt request signal after Mp generates INTA (Interrupt Acknowledge) Signal. (Lowest Priority Interrupt)

IO/M :
Two modes of operation
1) I/O mode (IO/M = 1)
2) memory mode (IO/M = 0)

X1/X2 :
* Crystal oscillator used for internal clock generator.
* Frequency divided by two.

CLK (out) : System clock for device Connected with the Mp

RESET IN : Reset the Mp by setting the Program counter to zero.

RESET OUT : Reset all devices which are connected with Mp.

HOLD : Another master requesting to use the address & data bus.

HLDA :
* HOLD Acknowledge
* Indicates the CPU has received HOLD Request & it will relinquish the bus in next clock cycle.
* HLDA set to Low after HOLD signal removed.

SOD :
* Serial output data lines
* set/Reset Specified by SIM Instruction.

SID :
* Serial Input data lines.
* Data is Loaded into Accumulator whenever RIM instruction executed

Vcc & Vss : Vcc → +5V
Vss → Ground

Status codes of 8085 :

S1	S0	Operations
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

Registers of 8085 :

* Used for temporary storage and manipulation of data & instruction
* Data remain in the register till sent to I/O devices or memory.

1) Accumulator (A) :
* 8 bit Register associated with ALU
* Hold one operand of ALU operation.

2) General Purpose Registers :
* Contains 6 general purpose registers (B, C, D, E, H & L)
* combination of two 8 bit register called register pair. (Holds 16 bit data).
(B-C, D-E & H-L)

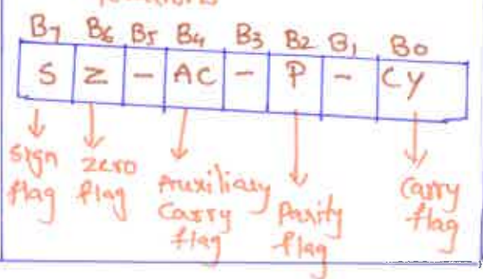
3) Program Counter :-
* 16 bit special purpose register.
* Holds address of memory of next instruction to be executed.
* Track the instruction in a program when executed.
* Increment the content of next Program Counter (PC) during execution.

4) Stack Pointer (SP):
* 16 bit special purpose register used as memory Pointer.
* portion of RAM (controls the address of stack).

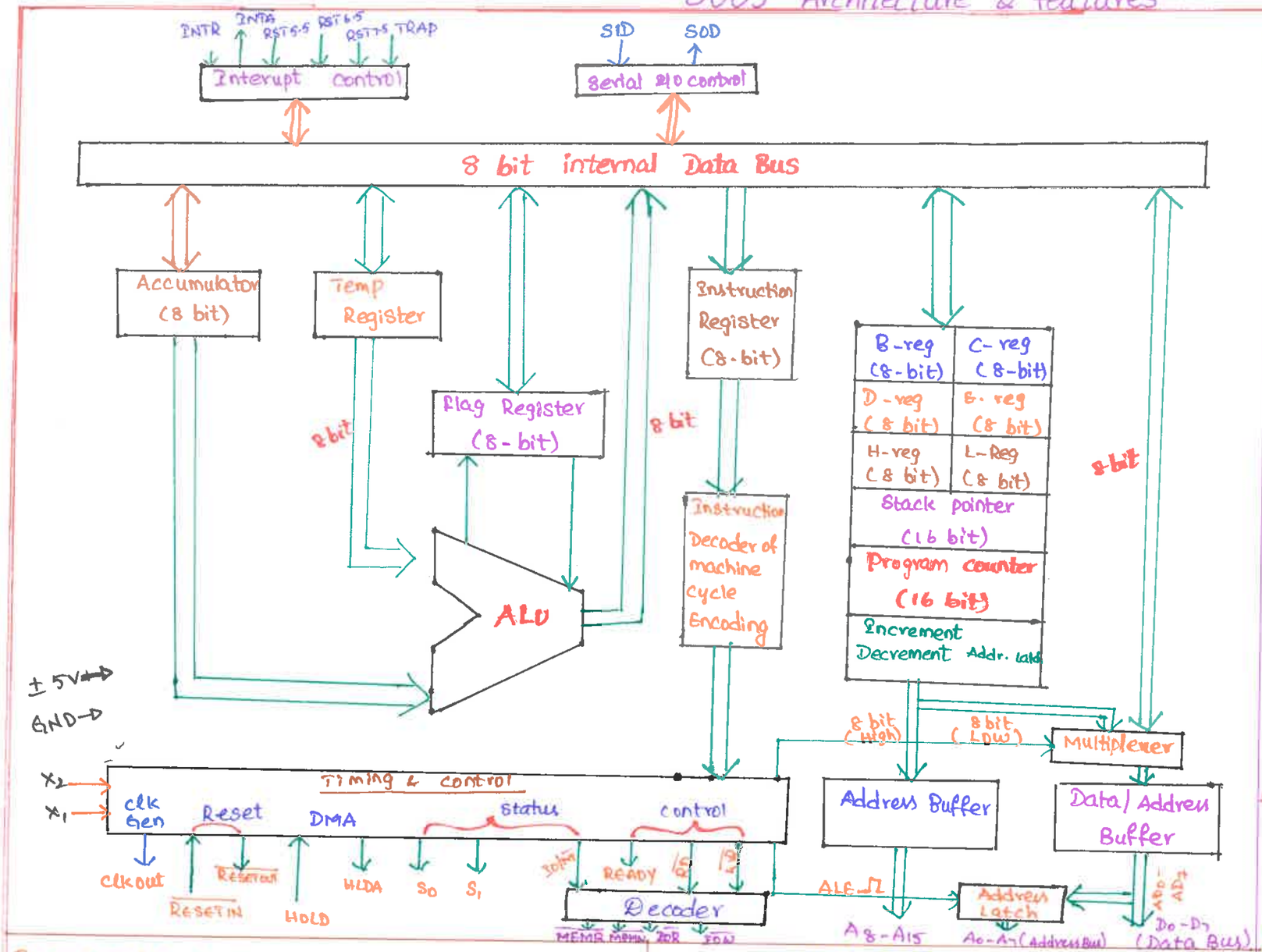
5) Instruction Register (IR):
* Holds opcode of the instruction which is being decoded & executed.

6) Temporary Register : (8 bit register)
* Holds data during ALU operation.
* Not accessible to Programmer.

Flag Register :
* Five filpflops to serve as status flag.
* Set/Reset according to ALU operations



8085 Architecture & features



Architecture :-

* Interrupt control unit consists of
  a) INTR
  b) INTA
  c) RST 5.5
  d) RST 6.5
  e) RST 7.5
  f) TRAP
* Arithmetic & logic unit perform operations such as
  1) Addition
  2) Subtraction
  3) Logical AND
  4) Logical OR
  5) Logical EXOR
  6) Logical NOT
  7) Increment
  8) Decrement
  9) shift, etc
* Timing & control unit consists of
  1) clock generation
     a) X_1, X_2
     b) CLK.OUT
  2) RESET
     a) $\overline{\text{RESET IN}}$
     b) RESET OUT
  3) Control
     a) Ready
     b) $\overline{RD}$
     c) $\overline{WR}$
  4) Status
     a) S_0, S_1
     b) $IO/\overline{M}$
  5) DMA
     a) HOLD
     b) HLDA

Flag Register

eg:-

```
        AC
    1 0 1 1 1 0 1 0
  + 0 1 1 0 1 0 0 1
  ---------------
  1 0 0 1 0 0 0 1
  cy
```

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	X	1	X	0	X	1
S	Z	-	Ac	-	P	-	cy

Features of 8085 Hp:

* 40 pin IC package fabricated on a single LSI chip
* uses a single ± 5 vd.c
* clock speed about 3MHz, clock cycle of 320ns
* 8 bit Data bus
* 16-bit Address Bus (address upto 64 kB)
* 16 bit stack pointer
* 16 bit program Counter (pc)
* six 8 bit registers (B, C, D, E, H, L)
  Register pair (B-C, D-E, H-L)
* consists of 80 basic instruction sets & 246 opcodes

Application: Mobile phones, Microwave ovens, etc.

Status of different control signal for different operation:

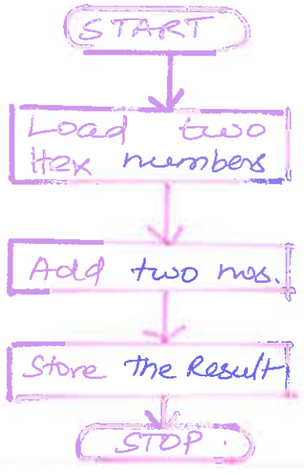
Machine cycle	Status			Control		
	$IO/\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch (OF)	0	-	-	0	1	1
Memory Read ($\overline{MEMR}$)	0	-	-	0	1	1
Memory write ($\overline{MEMW}$)	0	0	1	1	0	1
I/O Read ($\overline{IOR}$)	1	-	-	0	1	1
I/O write ($\overline{IOW}$)	1	0	1	1	0	1

WRITE AND EXECUTE ASSEMBLY LANGUAGE PROGRAMMING OF 8085.

Write, assemble and executing Assembly language programming of 8085 to add two numbers. The three tasks are involved in this program

* **Load two hex numbers**
* **Add two numbers**
* **Store the result in the memory**

These tasks for write and executing assembly language program of 8085 is represented by flow chart.



TASK of INSTRUCTIONS.

Task 1 Instructions:

```
MVI A, 20H ; Load 20H
MVI B, 40H ; Load 40H
```

Task 2 Instructions:

```
ADD B ; Add two numbers and Save in Areg.
```

Task 3 Instructions:

```
STA 220H ; Store the result in memory 220H. and Stop.
```

Data 1: 23; Data 2: 35; output: 58.

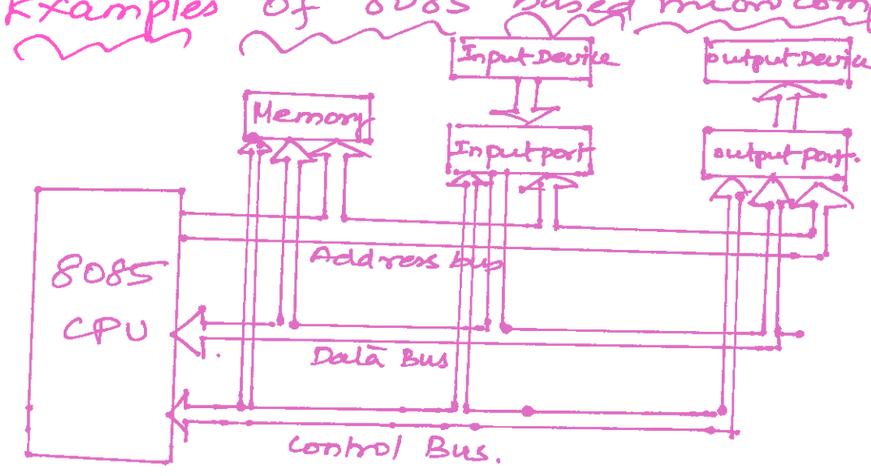
Assembly language program to machine language program.

Memory Address	Opcodes	Mnemonics
4100	3A	LDA 4150
4101	50	
4102	41	
4103	47	MOV B,A
4104	3A	LDA 4151
4105	51	
4106	41	
4107	80	ADD B
4108	32	STA 4152
4109	52	
410A	41	
410B	76	HLT.

Execution Steps:

1. Reset the microprocessor system by Pressing the RESET key.
2. Enter into store mode by pressing SET key.
3. Enter the address of the memory where the first hex code is to be Stored using hex keys.
4. Enterep the hex code using hex keys.
5. Increment the memory address by 1 using INC key.
6. Repeat steps 4 and 5 until the last hex code.

Examples of 8085 based microcomputer.



8085 MPU

Compulir Languages:

1) Machine language — low level language — first developed in first generation compulir — written in binary codes. (0 and 1).
2) Assembly language — second generation programming language — using english words, names and symbols — necessary for any processor.
3) High level language — Programming language ex: PASCAL, FORTAN, C++ etc.

8085 Instruction Set

Instruction Format:

* One Byte Instructions
  Eg: RRC, CMA, MOV B, C
* Two Byte Instructions
  Eg: MVI B, 45H
* Three Byte Instructions
  Eg: CALL, JMP

Instruction set:

1) Data transfer Instruction Set:

* No flags are affected

Instruction set	Explanation	Addressing	Example
MOV r_1, r_2 1→Machine Cycle (MC)	$r_1 \leftarrow r_2$	Register	MOV A, B
MOV r, m 2→MC	r ← [HL] (memory)	Register Indirect	MOV B, M
MOV M, r 2←MC	[HL] ← r	Register Indirect	MOV M, C
MVI r, Data 3←MC	r ← data	Immediate	MVI B, 45H
LXI rp, Data 3←MC	rp ← 16 bit data rh ← 8 bit MSB rL ← 8 bit LSB	Immediate	LXI H, 2000H
LDA addr 4←MC	A ← [addr]	Direct	LDA 2000H
STA addr 4←MC	[addr] ← A	Direct	STA 3000H
LHLD addr SHLD addr 5←MC	[L] ← [addr] [H] ← [addr+1] [addr] ← L [addr+1] ← H	Direct Direct	LHLD 2500H SHLD 2600H
XCHG 1←MC	[HL] ↔ [DE]	Register	XCHG

* Similar to LDA & STA, LDAX for 16 bit notation (2→MC)

2) Arithmetic Instruction:-

Instruction Set	Explanation	Flags	Addressing	Example
ADD r 1←MC	A ← A + r	ALL	Register	ADD B
ADD M 2←MC	A ← A + [H-L]	ALL	Register indirect	ADD M
ACC r 1←MC	A ← A + r + CY	ALL	Register	ACC B
ADC M 2←MC	A ← A + [HL] + CY	ALL	Register indirect	ADC M
ADI data 2←MC	A ← A + Data	ALL	Immediate	ADI 45H
ACI data 2←MC	A ← A + data + CY	ALL	Immediate	ACI 50H
DAD rp 3←MC	H-L ← HL + [rp]	CY	Register	DAD B

Similar to Addition, Subtraction can be expressed as SUB r, SUB M, SBB M SUI data, SBI data

INR r K←MC	r ← r+1	All except CY	Register	INR B
INR M 3←MC	[HL] ← [HL]+1	All except CY	Register indirect	INR M
INX rp 1←MC	rp ← rp+1	None	Register	INX H
DAA	Decimal Adjust Acc	–	–	DAA

Similar to increment, Decrement instruct is expressed as DCR r, DCR M, DCX rp.

3) Logical Instruction:

Instruction set	Explanation	Flags	Addressing	Example
ANA r 1←MC	A ← (A) ∧ (r) ↓ AND	All	Register	ANA C
ANA M 2←MC	A ← (A) ∧ (M) (HL)	All	Register indirect	ANA M
ANI data 2←MC	A ← (A) ∧ (data)	All	Immediate	ANI 22H

Similar AND operation, OR and EXOR operations can expressed as

1) ORA r, ORA M, ORI data eg: (A ← (A) ∨ (r) ∨ (M)
   A ← (A) ∨ (M)
2) XRA r, XRA M, XRI data eg: A ← (A) ⊕ (r)
   A ← (A) ⊕ (M)

CMA CMC 1←MC	A ← $\bar{A}$ CY ← $\bar{CY}$	None CY	Implicit Implicit	CMA CMC
STC 1←MC	CY ← 1 Set Carry	CY	Implicit	STC
CMP r 1←MC	Compare A and r-register (A>B, A<B, A=B)	All	Register	CMP B
CMP M 2←MC CPI data 2←MC	CMP A & M CMP A & data	All All	Indirect Immediate	CMP M CPI 22H
RLC	$A_{N+1} \leftarrow A_N$ $A_0 \leftarrow A_7$ CY ← A_7	CY	Implicit	RLC
RRC	Reverse of RLC $A_N \leftarrow A_{N+1}$ $A_7 \leftarrow A_0$ $A_7 \leftarrow$ CY	CY	Implicit	RRC
RAL	$A_{N+1} \leftarrow A_N$ CY ← A_7 A_0 ← CY	CY	Implicit	RAL

* Similarly RAR with Reverse steps of RAL

BRANCHING LOOPING, COUNTING and INDEXING OPERATIONS

JUMP:

CONDITIONAL JUMP INSTRUCTIONS:-

Instruction	Explanation	Example
JC	Jump on Carry if Carry flag is '1'	JC 2000
JNC	Jump on No Carry If Carry flag is '0'	JNC 2050
JZ	Jump on Zero if Zero flag is '1'	JZ 2000
JNZ	Jump on NO Zero, If Zero flag is '0'	JNZ 2050
JPE	Jump on Parity Even ; PF is '1'	JPE 2000
JPO	Jump on Parity ODD ; PF is '0'	JPO 2050
JM	Jump on Sign 'MINUS' If SF is '1'	JM 2000
JP	Jump on Sign 'plus' If SF is '0'	JP 2050

CALL:

CONDITIONAL CALL INSTRUCTIONS:-

Instruction	Explanation	Example.
CC	Call on Carry flag CF: 1	CC 2000
CNC	Call on No Carry flag CF: 0	CNC 2050
CZ	Call on Zero flag ZF: 1	CZ 2000
CNZ	Call on No Zero flag ZF: 0	CNZ 2050
CPE	Call on Parity Even PF: 1	CPE 2000
CPO	Call on Parity ODD PE: 0	CPO 2050
CM	Call on Minus SF: 1	CM 2000
CP	Call on Plus SF: 0	CP 2050

Three Types of Branching :-

① JUMP (Conditional & unConditional)
② CALL (Conditional & unConditional)
③ RET (Conditional & unconditional).

UNCONDITIONAL JUMP :-

JMP address ; Jump to the address

Ex: JMP 2000.

UN CONDITIONAL CALL :-

CALL address ; Call to the address

UN CONDITIONAL RETURN :-

RET address ; Return to the address

Ex: CALL 2000

Ex: RET

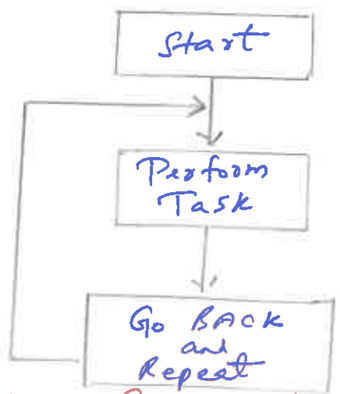
Looping in 8085 :-

→ Instructs MicroProcessor to repeat tasks.

① Continous Loops
② Conditional Loops.

Continous Loops :- Repeats a task Continously.

Conditional Loop :- Repeats a task if Some conditions are Satisfied



Flowchart for Continous loop

RETURN:-

CONDITIONAL RET INSTRUCTIONS

Instruction	Explanation	Example
RC	Return on Carry CF: '1'	RC
RNC	Return on No Carry CF: '0'	RNC
RZ	Return on Zero ZF: '1'	RZ
RNZ	Return on No zero ZF: '0'	RNZ
RPE	Return on Parity EVEN PF: '1'	RPE
RPO	Return on Parity ODD PF: 0	RPO
RM	Return on Minus SignFlag: '1'	RM
RP	Return on plus signflag: '0'	RP

INPUT- OUTPUT & STACK Instructions :-

* IN Port address ; Input to accumulator from I/o Port
* OUT Port address ; output from accumulator to I/o Port
* PUSH rp ; Push Content rp to stack.
* PUSH PSW ; Push processor status word.
* POP rp ; PoP Content from Stack.
* POP PSW ; PoP Processor Status Word.

MACHINE CONTROL INSTRUCTIONS:-

* HLT ; Halt
* XTHL ; Exchange stack top with H-L.
* SPHL ; HL-pair to stack pointer.
* EI ; Enable Interrupt
* DI ; Disable Interrupt
* SIM ; Set Interrupt Mask
* RIM ; Read Interrupt Mask
* NOP ; No operation.

ADDRESSING MODES OF 8085

Various ways of accessing data are called Addressing modes

Types of Addressing Modes Supported by 8085 Microprocessor

* Immediate addressing
* Register addressing
* Direct addressing
* Indirect addressing
* Implied addressing

Immediate add

* Data is written as part of Instruction

Ex :- MVI C, 8FH

C
[8F] ⇐ 8FH

2. LXI H, 472CH

H L
[47 | 2C] ⇐ 472CH

3. SUI 46H

A ⇐ A − 46

4. XRI 7D

A ⇐ A XOR 7D

Register Addressing

* Data is given by a register

Examples:

1. ADD D

A ⇐ A + D

2. DAD B

HL ⇐ HL + BC

3. MOV A, E

A ⇐ E

Direct Addressing

* Address of a data is specified in an Instruction.

Examples:

1. STA 4500H

4500 ⇐ [A]

2. LHLD A2F0H

L ⇐ A2F0 □

H ⇐ A2F1 □

3. IN 86H

A ⇐ 86H (Input Device)

4. OUT C8

C8 (output device) ⇐ A

Indirect Addressing

* Address of a data is given by a register

Examples:

1. MOV A, M

M - Memory

A ⇐ M(HL)

2. CMP M

A & M Compared and Flag updated.

Implied or Implicit Addressing

* Instruction itself Specified the data to be operated

Examples:

1. CMA

A ⇐ $\bar{A}$

2. RAR

Rotate Accumulator right through Carry

3. RLC

Rotate Accumulator Left once

4. STC

CY ⇐ 1

5. CMC

CY ⇐ $\overline{CY}$

* Add two 8-bit numbers 2A and B8 to demonstrate various addressing modes using-

Immediate -ALP

```
MVI A, 2AH
ADI B8H
STA 5750H      Output
HLT            5750 : E2
```

Register

```
MVI A, B8H
MVI E, 2AH
ADD E          output
STA 5750H      5750 : E2
HLT
```

Indirect

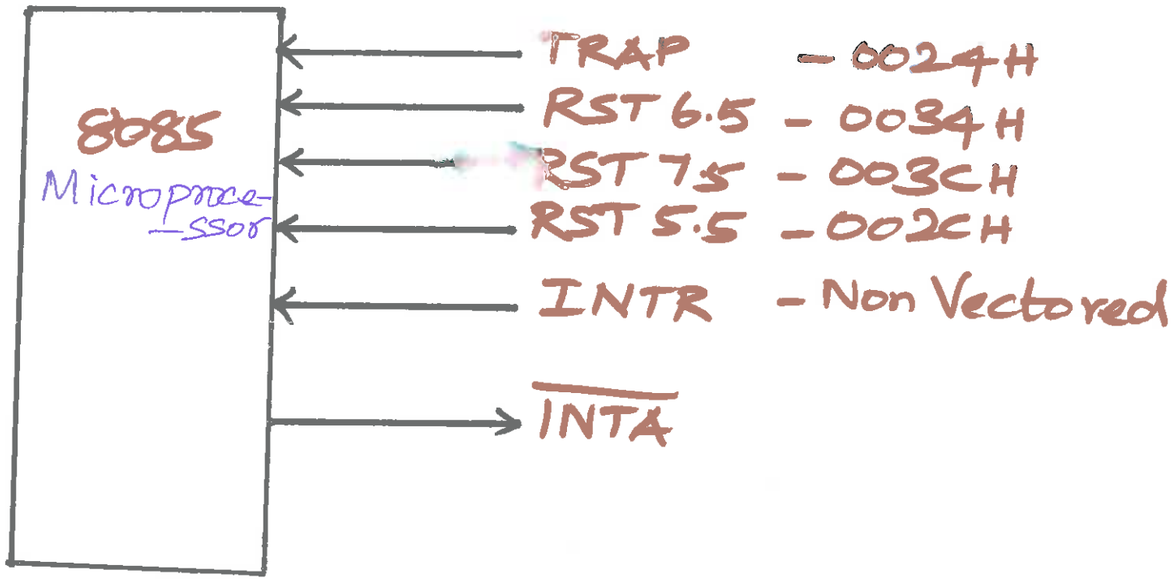
```
LXI H, 4500H
MOV A, M       I/P  | 4500 : 2A
INX H               | 4501 : B8
ADD M          O/P  | 4502 : E2
INX H
MOV M, A
HLT
```

```
    1 1
2A ⇒ 0 0 1 0 1 0 1 0
B8 ⇒ 1 0 1 1 1 0 0 0 +
-------------------
  1 1 1 0 0 0 1 0
  E     2
```

Interrupts in 8085 Microprocessor

* Effective way of Serivicing Input/Output devicess is called Interrupt. It is classified into Hardware interrupts and Software Interrupts.

Hardware Interrupts & Signals



TRAP: It is a non maskable Interrupt. It is Edge and level triggered. TRAP has highest Priority.

RST 7.5: It is a maskable Interrupt. It is positive edge triggered interrupt.

RST 6.5 & RST 5.5: These are level triggered interrupts. The RST 6.5 has third priority and 5.5 has fourth priority. These are maskable interrupts.

INTR: It is a maskable Interrupt. It is not the vector interrupt. It has lowerst priority interrupt.

* Software Interrupts

* When Microprocessor exicutes certain instructions, control is transfered to predefined ISR in the Memory.
* There are Eight Software instructions as interrupts in 8085 Microprocessor

S No	Instruction	Op Code	Vector address
1	RST 0	C7	0000H
2	RST 1	CF	0008H
3	RST 2	D7	0010H
4	RST 3	DF	0018H
5	RST 4	E7	0020H
6	RST 5	EF	0028H
7	RST 6	F7	0030H
8	RST 7	FF	0038H

ISR:

Interrupt Service Routine...

It is a program to Service an interrupting device, which is located on the vectored address.

* **SIM**: Set Interrupt Mask. It is used to disable the Maskable interrupts in 8085 Microprocessor. If mask bit = 1 then it is disabled.
* **RIM**: Read Interrupt Mask. After exicuting this instruction, Accumulator will give the Status of the interrupts masking in 8085 Microprocessor

8086 MICROPROCESSOR

FEATURES

→ Design : Intel - 1976 → Updated Version of 8085 Microprocessor

→ 16 bit, N-Channel High Speed Metal Oxide Semiconductor

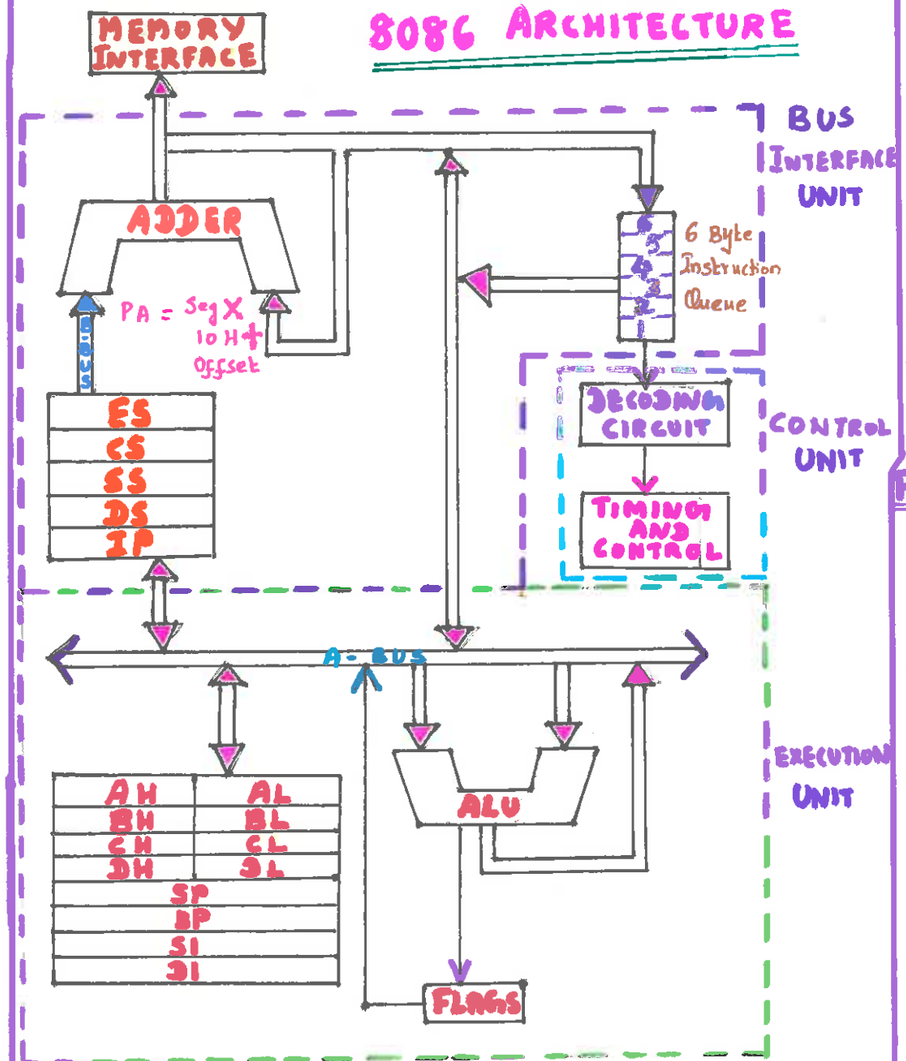
→ Built on Single semi conductor chip and packed in 40 pin IC pack (Dual Inline Package)

→ Uses 20 Address lines & 16 Data lines

→ Addresses upto 2^{20} = 1 Mbyte Memory

→ Size of I/O : 2^{16} = 64 kB

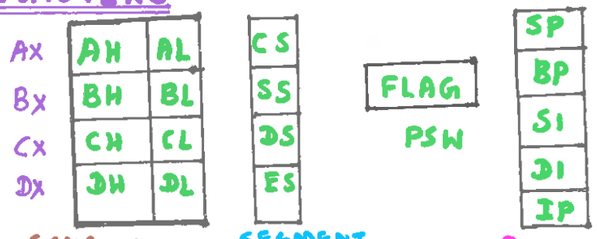
8086 ARCHITECTURE



→ 8086 Microprocessor consists of two units

1. Bus Interface Unit
   - ↳ Contains Segment Registers, Instruction Pointer and 6 Byte Instruction Queue
2. Execution Unit
   - ↳ Contains Programmable Registers, Index Registers, Pointer Register and Flag Registers. & ALU

REGISTERS



AX - Accumulator
Bx - Base
Cx - Counter
Dx - Data

CS - Code Segment
SS - Stack Segment
Ds - Data Segment
ES - Extra Segment

SP - Stack Pointer
BP - Base Pointer
SI - Source Index
DI - Destination Index
IP - Instruction Pointer

Two Operating modes : Maximum mode / Minimum mode

FLAGS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	A_C	X	P	X	C_y

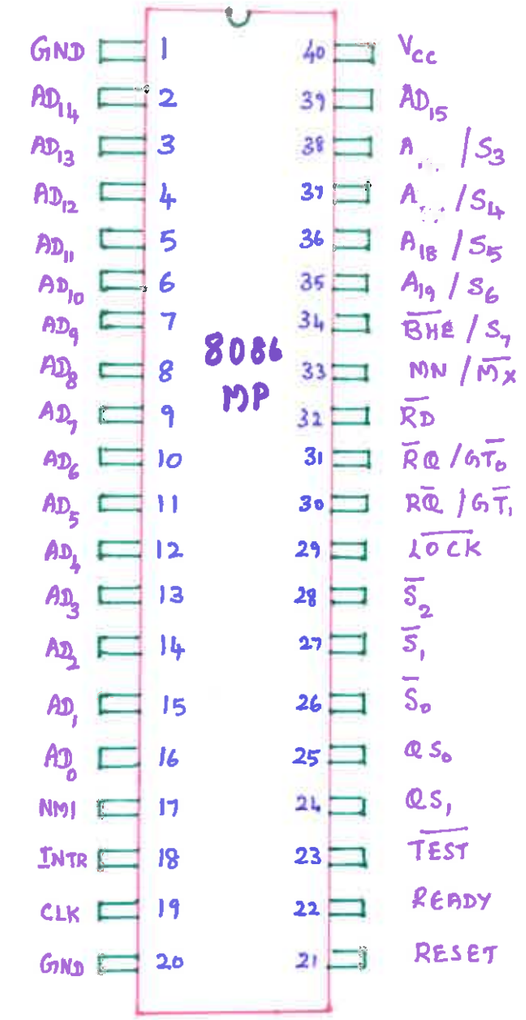
Conditional Flags

* Carry Flag (C_y)
* Auxiliary Flag (A_C)
* Parity Flag (P)
* Zero Flag (Z)
* Sign Flag (S)
* Overflow Flag (O)

Control Flags

* Trap Flag (T)
* Interrupt Flag (I)
* Direction Flag (D)
  - Clear Direction Flag
  - Set Direction Flag

PIN CONFIGURATION



* V_{cc} -> Power Supply => +5 V
* $AD_0 - AD_{15}$ -> A 16 Bit Address Data Bus
* $A_{16} - A_{19}$ -> Higher Order Address lines & Multiplexed with status signals
* $\overline{BHE}$ / S_7 -> Bus High Enable / Status
* $\overline{RD}$ -> Read , GND - Ground
* RESET -> System Reset
* CLK (i/p) -> Clock 5, 8, 10 MHz
* INTR -> Interrupt Request
* NMI -> Non Maskable Interrupt Request

8086 Addressing Modes

Types of Addressing modes :-

Group I
1) Register Addressing
2) Immediate Addressing

Group II for memory
3) Direct Addressing
4) Register Indirect Addressing
5) Based Addressing
6) Indexed Addressing
7) Based Index Addressing
8) String Addressing

Group III
9) Direct I/O Port Addressing
10) Indirect I/O Port Addressing

Group IV 11) Relative Addressing

Group V 12) Implied Addressing.

1) Register Addressing :-

The instruction will specify the name of the register which holds the data to be operated by the instruction.

Eg :- MOV CL, DH
(CL) ← (DH)

2) Immediate Addressing :-

An 8bit or 16bit data is specified as part of the instruction.

Eg :- MOV DL, 08H ⇒ (DL) ← 08_H
MOV AX, 0A9FH
⇒ (AX) ← $0A9F_H$

3) Direct Addressing :-

* The effective address of the memory location at which the data operand is stored.
* Effective Address will be 16bit number.

Eg :- MOV BX [13545] ; BX ← [1354]
MOV BL, [0800H] BL ← [00]

4) Register Indirect Addressing :-

* Name of the register which holds the effective address [EA] will be specified in the instruction.
* Register used to hold EA are
BX, BP, DI, SI

Eg : MOV CX, [BX]

Operation :-
EA = (BX)
BA = (DS) x 16_{10}
MA = BA + EA
(CX) ← (MA)
(or)
(CL) ← (MA)
(CH) ← (MA+1)

5) Based Addressing :-

* BX or BP is used to hold the base value for effective address and a signed 8 bit or unsigned 16-bit displacement will be specified.
* In case of 8 bit displacement, signed extend to 16 bit before adding base value.
* BX holds base value of EA, 20 bit PA is calculated from BX & DS
* BP holds base value of EA, BP & SS is used.

Eg : MOV AX, [BX + 08H]

Operation :-
0008_H ← 08_H
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA
AX ← (MA)

6) Indexed Addressing :-

* SI or DI register is used to hold an Index value for memory data and a signed 8 bit or unsigned 16 bit displacement will be specified.

Eg : MOV CX, [SI + 0A2H]

Operation :-
$FFA2_H$ ← $0A2_H$
EA = (SI) + $FFA2_H$

7) Based Index Addressing :-

* EA is computed from the sum of a base register (BX or BP), an index register (SI or DI) and displacement.

Eg :- MOV DX, [BX + SI + 0AH]

Operation :-
$000AH$ ← $0AH$
EA = (BX) + (SI) + $000AH$

8) String Addressing :-

→ Employed in String operation to operate on string data.
→ EA of source data is store in SI & EA of destination data is stored in DI.

Eg :- MOVS BYTE

Operation :-
Calculation of Source memory :-
EA = (SI) BA = (DS) x 16_{10} MA = EA + BA
Calculation of destination memory :-
EA_E = (DI) BA_E = (ES) x 16_{10}
MA_E = BA_E + EA_E

9) Direct I/O Port Addressing :-

* used to access data from standard I/O mapped devices on Port.

Eg :- IN AL, [09H]

Operation :- $Port_{addr}$ = 09H
(AL) ← ($Port_{addr}$)

10) Indirect I/O Port Addressing :-

11) Relative Addressing :-

→ EA of program instruction is specified relative to IP by 8bit displacement.

Eg : JZ 0AH

12) Implied Addressing :-

→ have no operands

Eg :- CLC

Identify the Addressing Modes of following Instr

ADD CX, [2100H]
DIV BX
IN AL, [1000H]
OUT ~~AX~~, ~~DX~~
AND CX, [BX+SI]
SUB CX, 4567H
XOR CL, [BX+8]
ADD AL, [DI+6]
SUB CH, [BX+DI]
AND AH, [BX+SI+7]
JNZ A1
SCANSB

8086 INSTRUCTION SET

INSTRUCTION SETS OF 8086 MICROPROCESSOR

TYPES OF INSTRUCTION SETS

* DATA TRANSFER
* ARITHMETIC
* LOGICAL
* STRING MANIPULATIONS
* CONTROL TRANSFERS
* PROCESSOR CONTROL

* DATA TRANSFER

MOV = MOVE
Register/Memory to/from Register
Immediate to Register/Memory
Immediate to Register
Memory to Accumulator
Accumulator to Memory
Register/Memory to Segment Register
Segment Register to Register/Memory
PUSH = Push:
Register/Memory
Register
Segment Register
POP = POP:
Register/Memory:
Register
Segment Register
XCHG = Exchange
Register/Memory with Register
Register with Accumulator
IN = Input from:
Fixed port
Variable port
XLAT = Translate Byte to AL
LEA = Load EA to Register
LDS = Load pointer to DS
LAHF = Load AH with Flags
LES = Load pointer to ES
SAHF = Store AH into Flags
PUSHF = push Flags
POPF = POP flags

* ARITHMETIC

ADD = Add:
Reg/Memory with Register to Either
Immediate to Register/Memory
ADC = Add with carry:
Reg/Memory with Register to Either
Immediate to Register/Memory
Immediate to Accumulator
INC = Increment:
Register/Memory
Register
AAA = ASCII Adjust for Addition
DAA = Decimal Adjust for Addition
SUB = Subtract
Reg/Memory and Register to Either
Immediate from Register/Memory
Immediate from Accumulator
SBB = Subtract with Borrow
Reg/Memory and Register to Either
Immediate from Register/Memory
Immediate from Accumulator
DEC = Decrement:
Register/Memory
Register
NEG = Change sign
CMP = compare:
Register/Memory and Register
Immediate with Register/Memory
Immediate with Accumulator
AAS = ASCII Adjust for Subtract
DAS = Decimal Adjust for Subtract
MUL = Multiply (unsigned)
IMUL = Integer Multiply (signed)
AAM = ASCII Adjust Multiply
DIV = Divide (Unsigned)
IDIV = Integer Divide (signed)
AAD = ASCII Adjust for Divide
CBW = convert Byte to word
CWD = convert word to Doubleword

* LOGICAL

NOT = Invert
SHL/SAL = Shift Logical/Arithmetic Left
SHR = Shift Logical Right
SAR = Shift Arithmetic Right
ROL = Rotate Left
ROR = Rotate Right
RCL = Rotate through carry Flag. Left
RCR = Rotate through carry Right
AND = And:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator
TEST = And Function to Flags, No Result:
Register/Memory and Register
Immediate Data and Register/Memory
Immediate Data and Accumulator
OR = OR:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator
XOR = Exclusive OR:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator

* STRING MANIPULATIONS

REP = Repeat
MOVS = Move Byte/word
CMPS = compare Byte/word
SCAS = scan Byte/word
LODS = Load Byte/wd to AL/AX
STOS = Stor Byte/wd from AL/AX

BRANCHING AND LOOPING OF 8086

INSTRUCTION SETS OF 8086 MICROPROCESSOR

* CONTROL TRANSFER

CALL = call :
Direct within Segment
Indirect within Segment
Direct Intersegment
Indirect Intersegment
JMP = unconditional Jump :
Direct within Segment
Direct within Segment -short
Indirect within Segment
Direct Intersegment
Indirect Intersegment
RET = Return from CALL :
within Segment
within Seg Adding Immediate to SP
Intersegment
Intersegment Adding Immediate to SP
JE/JZ = Jump on Equal / Zero
JL/JNGE = Jump on Less / Not Greater or Equal
JLE/JNG = Jump on Less or equal / Not Greater
JB/JNAE = Jump on Below / Not Above or Equal
JBE/JNA = Jump on Below or equal / Not above
JP/JPE = Jump on parity / Parity Even
JO = Jump on Overflow
JS = Jump on Sign
JNE/JNZ = Jump on Not Equal / Not Zero.
JNL/JGE = Jump on Not Less / Greater or Equal
JNLE/JG = Jump on Not Less or equal / Greater
JNB/JAE = Jump on Not Below / Above or Equal
JNBE/JA = Jump on Not Below or equal / Above
JNP/JPO = Jump on Not par / par odd
JNO = Jump on Not overflow
JNS = Jump on Not sign
LOOP = LOOP CX Times
LOOPZ/LOOPE = LOOP while Zero / Equal
LOOPNZ/LOOPNE = LOOP while Not Zero / Equal
JCXZ = Jump on CX Zero
INT = Interrupt
Type Specified
Type 3
INTO = Interrupt on overflow
IRET = Interrupt Return
Eg: INT 02H → Type 02 Interrupt

* PROCESSOR CONTROL

CLC = clear carry
CMC = complement carry
STC = set carry
CLD = clear Direction
STD = Set Direction
CLI = clear Interrupt
STI = set Interrupt
HLT = Halt
WAIT = wait
ESC = Escape (to external Device)
LOCK = Bus Lock prefix

Operation :
* STC → To set carry flag to 1
* CLC → clear / reset carry flag to '0'.
* CMC → put complement at the stat of CF
* STD & CLD
* Set Direction flag to '1'
→ Clear / resent Direction flag to '0'.
* Directing the CPU Execution Sequence in an order
* Controlling CPU

* CONDITIONAL BRANCH INSTRUCTION

1. JZ/JE → Zero
2. JNZ/JNE → NOT Zero / E
3. JS → Sign
4. JNS → NOT Sign
5. JO → over-flow
6. JNO → NOT over-flow
7. JP/JPE → parity / parity even
8. JNP → NOT parity
9. JB/JNAE/JC → Below, Not Above (or) Equal
10. JNB/JAE/JNC → Not Above, Above (or) equal
11. JBE/JNA → ~~Above~~ Below / equal
12. JNBE/JA → Above
13. JL/JNGE → Less, Not greater (or) equal
14. JNL/JGE → Not less, Greater (or) equal
15. JLE/JNC → Less / equal
16. JNLE/JE → Not less / equal

UNCONTITIONAL CALL JUMP, RETURNE :

Eg: JMP 2050H
RET
CALL 2000H

ALP 1: Write an ALP to initialise Port 'A' of 8255 as input port.

* Write the ALP using 8086µP
* Interface 8255 with 8086µP
* Initialise Port 'A' of 8255 as input port.
* Get the output at 1500H.

Program:-

```
MOV SI, 1500H
MOV AL, 90
OUT C6, AL
IN AL, C0
MOV [SI], AL
HLT.
```

INPUT : Port A SPDT SWITCHES
0000 1111

OUTPUT: Source Index Register pointed at 1500H → 0FH

ALP 2: Write an ALP to Perform One's Complement of 1234H and Store the result in 1400H.

* Write the ALP using 8086µP
* Store the input Word 1234H in AX Register.
* Perform 1st Complement for the AX data.

Program:-

```
MOV AX, 1234H
NOT AX
MOV [1400], AX
HLT
```

INPUT:- (AX) => 1234H
0001 0010 0011 0100

OUTPUT:- [1400] => EDCB
1110 1101 1101 1011

ALP 3: Write an ALP to perform 16-bit addition of two numbers.

* Write the ALP using 8086µP
* Add two numbers in location's 1100 & 1102.
* Store the result at 1200H.

Program:-

```
MOV AX, [1100]
ADD AX, [1102]
MOV [1200], AX
HLT
```

INPUT:- [1100] = 1234H
[1102] = 5678H

OUTPUT:- [1200] = 68ACH

AX => AX + [1102]
AX <= [1200H]

ALP 4:- Write an ALP to move a byte string from source to destination.

* ALP code using 8086µP
* String operation (MOVE)
* Source → S_ARRAY
* Destination → S_ARRAY

Program:-

```
MOV SI, OFFSET S_ARRAY
MOV DI, OFFSET D_ARRAY
MOV CX, 00FFH
CLD
MOVE : MOVSB
LOOP MOVE
HLT
```

OUTPUT:- fill the location from 2000H to 20FFH
S_ARRAY = 2000 → 41H
D_ARRAY = 20FF → 41H

ALP 5:- Write an ALP to obtain Sum of 'N' numbers in a Word Array.

* ALP Codes using 8086µP
* Point the location where the Array starts.
* Sum 16-bit Array in Memory.
* Use SI Register to store the Result.

Program:-

```
MOV CX, 0005H
MOV AX, 0000H
MOV SI, AX
Loop1 : MOV AX, START [SI]
ADD SI, 2
LOOP Loop1
MOV [SUM], AX
HLT
```

INPUT:-
Array Starts @
1100 : 0FFF
1102 : 0FFF
1103 : 0FFF
1104 : 0FFF
1108 : 0FFF

OUTPUT:-
[1200] = AFFB$_H$

ALP 6:- Write an ALP for Masking of Bits Selectively.

* ALP Code's Using 8086µP
* Load the data in memory location
* Logically AND with 0F0F.
* Result Stored in AX Register.

Program:-

```
MOV AX, [1200]
AND AX, 0F0F
MOV [1400], AX
HLT
```

INPUT:-
[1200] => FF
[1201] => FF

OUTPUT:-
[1400] => 0F
[1401] => 0F

ALP 7:- Write an ALP to perform division of 32-bit number by a 16-bit number.

* Load the first data in AX by.
* Load the second data
* Divide AX and BX.
* Store the Result in AX and DX.

Program:-

```
MOV AX, DATA1
MOV BX, DATA2
DIV BX
MOV [1100], AX
MOV [1102], DX
HLT
```

INPUT:-
Dividend :-
1100 = 0000
1102 = FFFF
Divisor :-
1104 = FFFF
Remainder :-
1202 : 0000

OUTPUT:-
Quotient :
1200 : 0001

ALP 8:- Write an ALP to display 'A' in the first digit.

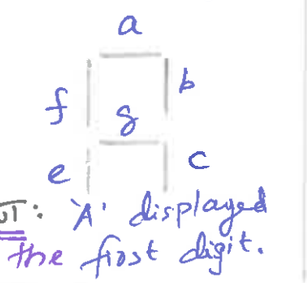
* Interface 8279 KDC with 8086µP
* Use Segment Definition of 'A'
* Control position and character of the display.

Program:-

```
MOV AL, 00
OUT AL, CL
MOV AL, CC
OUT C2, AL
MOV AL, 90
OUT C2, AL
MOV AL, 88
OUT C0, AL
MOV AL, FF
MOV CX, 0005
NEXT : OUT C0, AL
LOOP NEXT
HLT.
```

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0
d	c	b	a	dp	g	f	e
0	1	1	1	0	1	1	1

8 8



OUTPUT: 'A' displayed on the first digit.

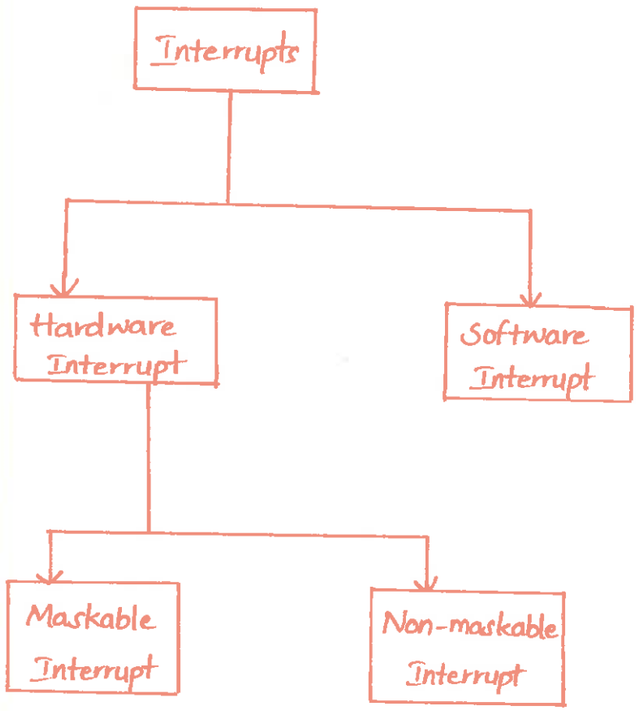
INTERRUPTS OF 8086

Interrupts - 8086

→ creating 'a' temporary halt during program execution.

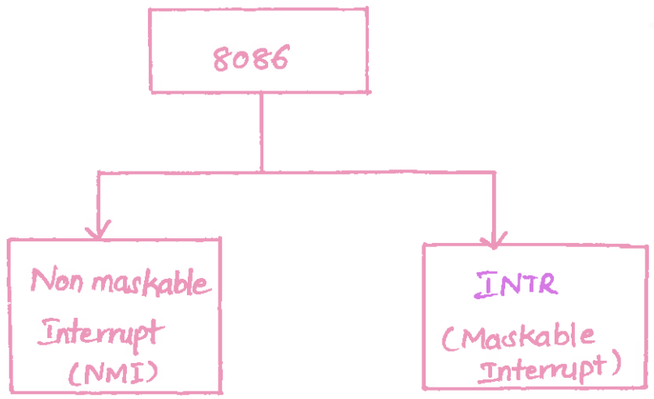
→ Allows peripheral devices to access the microprocessor.

Types :-



Hardware Interrupts

→ caused by any peripheral device by sending a signal through a particular pin to the microprocessor.



→ one more Interrupt - INTA (interrupt acknowledgment)

→ NMI & INTR - lower priority

Non-maskable Interrupt (NMI)

→ NMI pin - higher priority than Maskable Interrupt (INTR)

When Interrupt (NMI) is activated

→ complete the current instruction

→ Push the flag register values on to the stack.

→ push the cs (code segment) Value and Ip (instruction pointer) Value of the return address on the stack.

→ IP is loaded from the contents of the word location 00008H.

→ CS is loaded the contents of the next word location 0000AH.

→ Interrupt flag & trap flag are reset to 0.

INTR (Maskable Interrupt)

→ INTR enabled only set Interrupt + flag Instruction.

→ INTR Interrupt activated by an I/O port.

When Interrupt (INTR) is activated :

→ First complete the current Instruction.

→ Activate INTA output & receives the interrupt.

→ Flag register value, cs value of the return address and Ip value of the return address pushed on to the stack.

→ Ip value is loaded from the contents of word location.

→ cs loaded from the contents of the next word location.

→ Interrupt flags trap flag set to '0'

Software Interrupt

INT

- Type 0 - division by zero
- Type 1 - Single step executio
- Type 2 - Non maskable Interru
- Type 3 - Break point interrup
- Type 4 - over-flow Interrupt
- Type 5 - Type 31 - reservec for advanced microproc ssor
- Type 32 - To Type 255 - available hardwa & software inter rupt

INT3 - Break point Interrupt

→ one byte instruction

→ opcode - CCH

→ Instruction inserted into the program

→ During the execution stops the normal execution of program

→ follows the Break point

(INTO - overflow Inte)x

INTO - overflow Interrupt

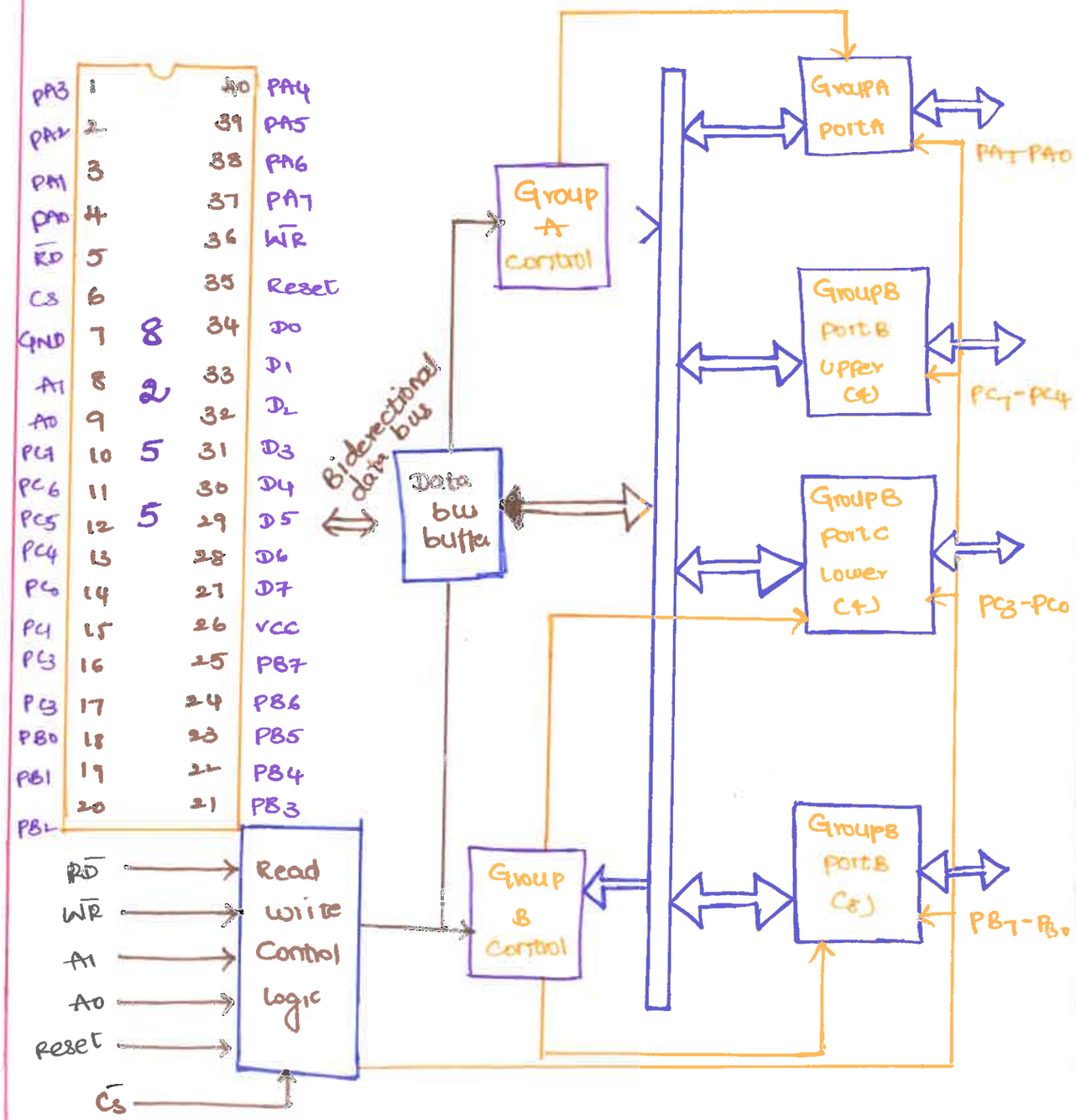
→ INTO & opcode - CEH

→ conditional interrupt

PROGRAMMABLE PERIPHERAL INTERFACE

8255 - programmable peripheral Interface [PPI]

The 8255 is a general purpose programmable I/O Device designed to Transfer the data from I/O devices the microprocessor and vice-versa.



Features of 8255

It has three I/O ports

* port A (PA7-PA0)
* port B (PB7-PB0)
* port C (PC7-PC0)

These three I/O ports are divided into two Groups i.e. Group A includes port A and upper PORT C Group B includes PORTB and lower PORTC.

Signals of 8255

D_7-D_0 → Bidirectional data bus

RESET → Reset Input

$\overline{CS}$ → Chip Select

$\overline{RD}$ → Read Input

$\overline{WR}$ → Write Input

A_1-A_0 → Port Address

PA_7-PA_0 → Port A

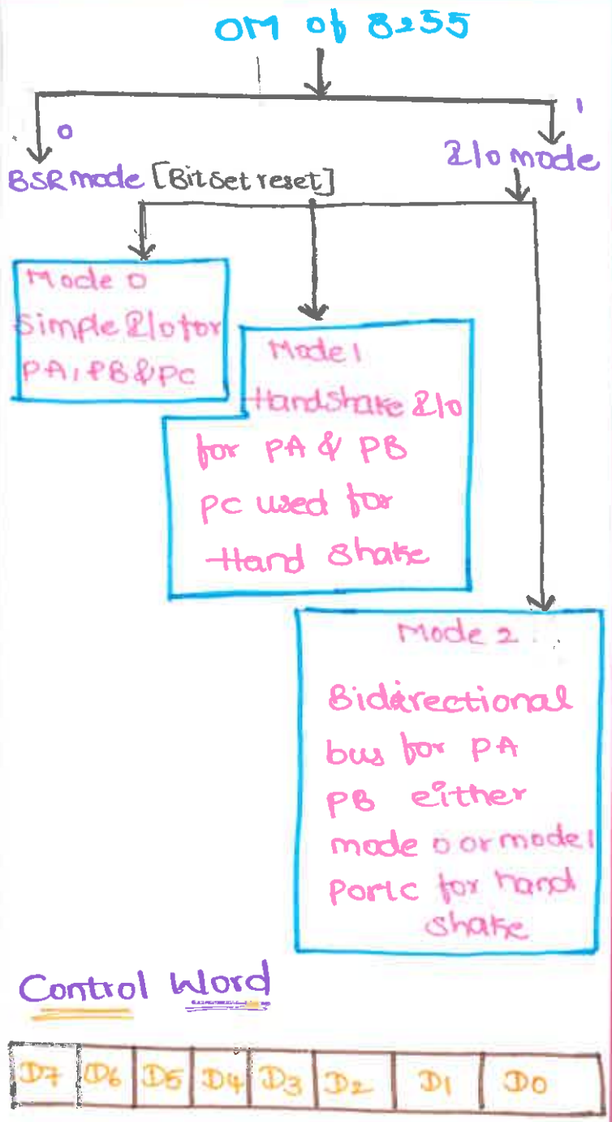
PB_7-PB_0 → Port B

PC_7-PC_0 → port C

VCC - 5 volts

GND - 0 volts.

operating Modes of 8255



Control Word

D7 - 1-I/O mode 0-BSR mode

D6, D5 - mode selection
D6 D5 mode

D4 - PA 1-Input, 0-output

D3 - PC upper (PC7-4) 1-I/P, 0-O/P

D2 - mode selection 1-mode 1 0-mode 0

D1 - PB 1-Input 0-output

D0 - PC lower (PC_3-0)
1-Input
0-output

INTEL 8279 KEYBOARD DISPLAY CONTROLLER

Features

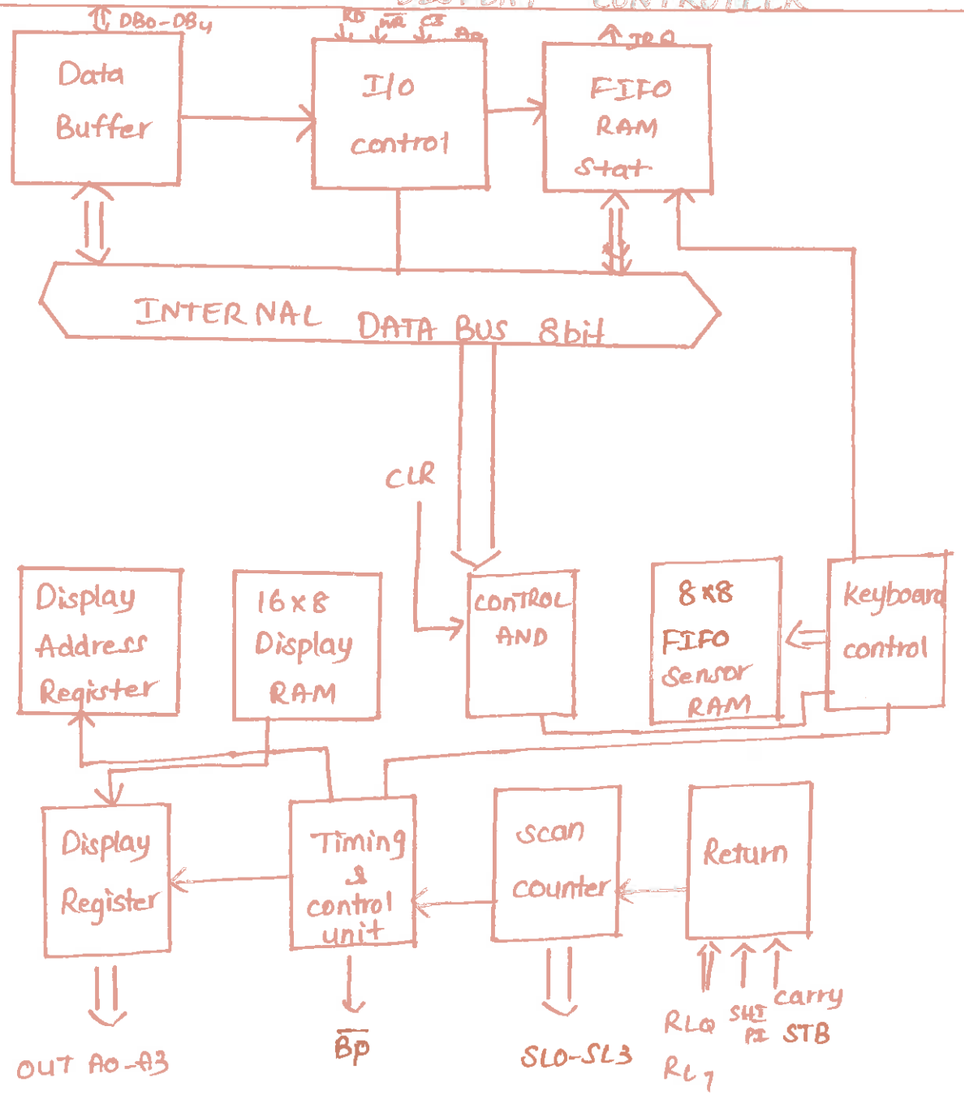
* 40 pin DIP
* Max 16 7-seg display
* Max 64 kbps
* 4 scan lines
* 8 bit Return lines
* 8 x 8 FIFO sensor RAM
* 16 x 8 Display RAM
* 8 O/P lines

PIN Details of 8279

DB0-DB7 - 8bit bidirectional data bus

$\overline{CS}$ → chip select

$\overline{RD}$ - Active low signal



INTERNAL ARCHITECTURE OF 8279

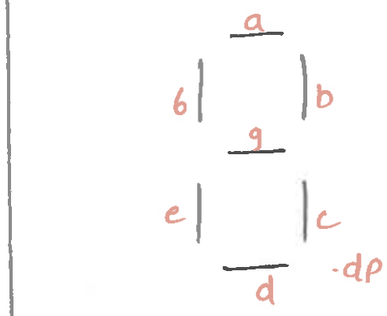
→ SL0-SL3 - Scan line → Scan the key board

→ RL0-RL7 - High → decode scan operation

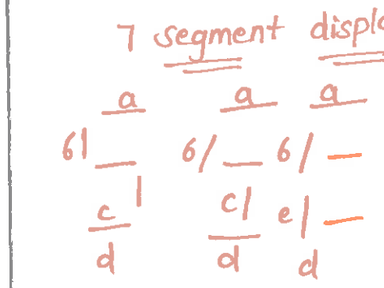
→ $\overline{BD}$ - Blank display

→ OUT A0-A2 & out B0-B2 (O/P)

Data bus	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
O/P	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0



7 segment display



d	c	b	a	dp	g	f	e	
0	1	1	0	1	1	0	1	20
0	0	1	0	1	1	0	1	20
0	1	1	0	0	1	0	0	6c

Loop up table

Applications of 8279

→ Right Entry display

→ Left Entry display

→ character display

→ Simultaneous key board display operation.

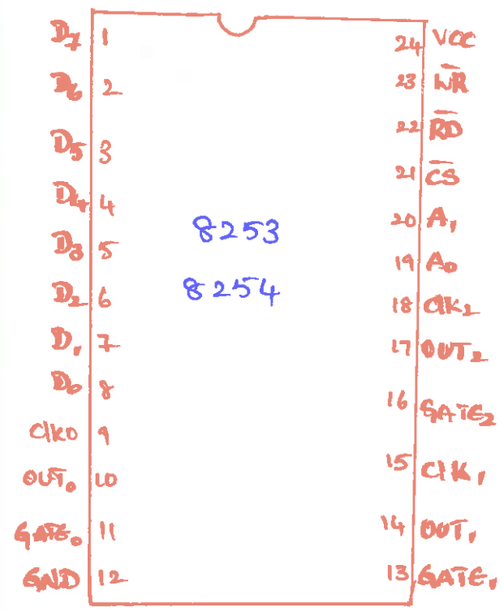
PROGRAMMABLE INTERVAL TIMER

8253/8254 Programmable Interval Timer

It is designed for microprocessors to perform timing and connecting functions using three 16-bit Registers

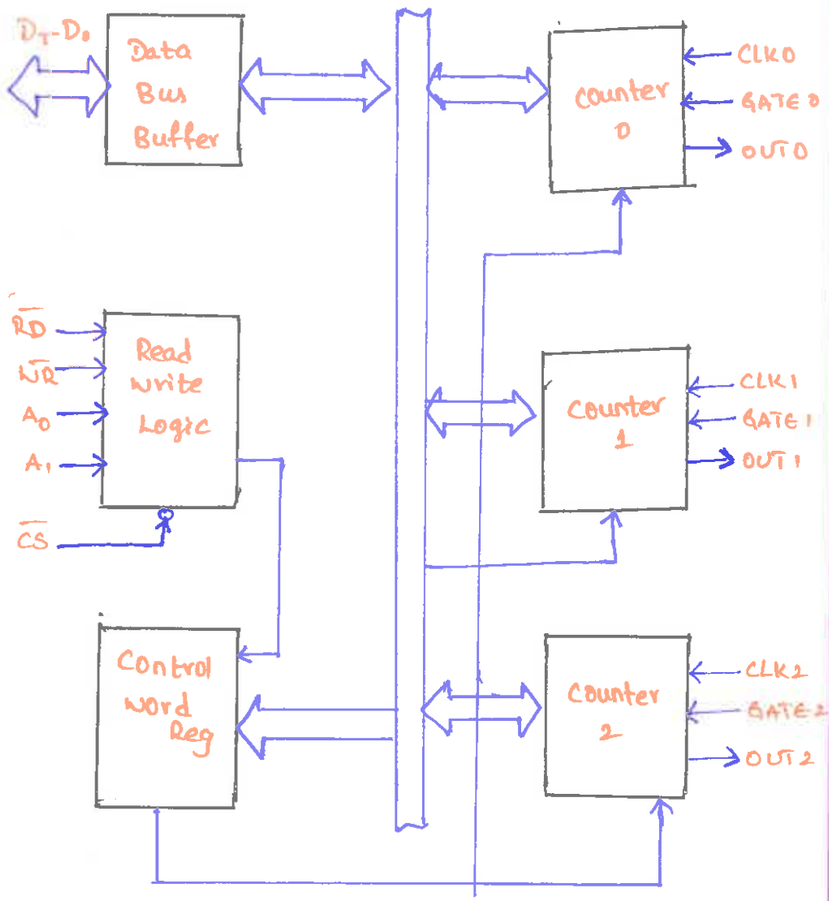
Features of 8253/8254

* It has three independent 16-bit down counters.
* It can count clock upto 10 MHz
* It supports for Both BCD & Binary counters
* It has Readback conditions Command



* It is a down Counter
* Generates output on terminal count
* It operates on 5V

Block Diagram of 8253/8254



Counter 0, 1, 2

All 16-bit independent counters

A_0, A_1 → 2 bit Address bus

D_0-D_7 - Bidirectional Databus

$\overline{CS}$ → Active low chip select

Data Bus Buffer

Tristate, bi-directional, 8-bit buffer for following function

* Programming the 8253/54
* Loading the count registers
* Reading the count values

Read / write Logic

$\overline{RD}$ - Read signal

$\overline{WR}$ - write signal

$\overline{CS}$ - chip select

A_1, A_0 - Address lines

A_1	A_0	Result
0	0	Count 0
0	1	Count 1
1	0	Count 2
1	1	Control word register

Working of Counter

* It counts every clock from its clk pin (Decrement timer register for every clk.
* Gate is used to stop, the counter
* When counter reaches '0', then out signal is generated

Operating Modes of 8253/54

* Mode 0 - Interrupt on terminal count
* Mode 1 - programmable one shot
* Mode 2 - Rate generator
* Mode 3 - Square wave rate generator
* Mode 4 - Software triggered stroke
* Mode 5 - Hardware triggered Stroke

* Desired freq of Squarewave is generated accurately

Control word

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD

1 - Binary
0 - BCD

SC_1	SC_0	Select counter
0	0	0
0	1	1
1	0	2
1	1	Illegal

Mode

0	0	0	0
0	0	1	1
x	1	0	2
x	1	1	3
1	0	0	4
1	0	1	5

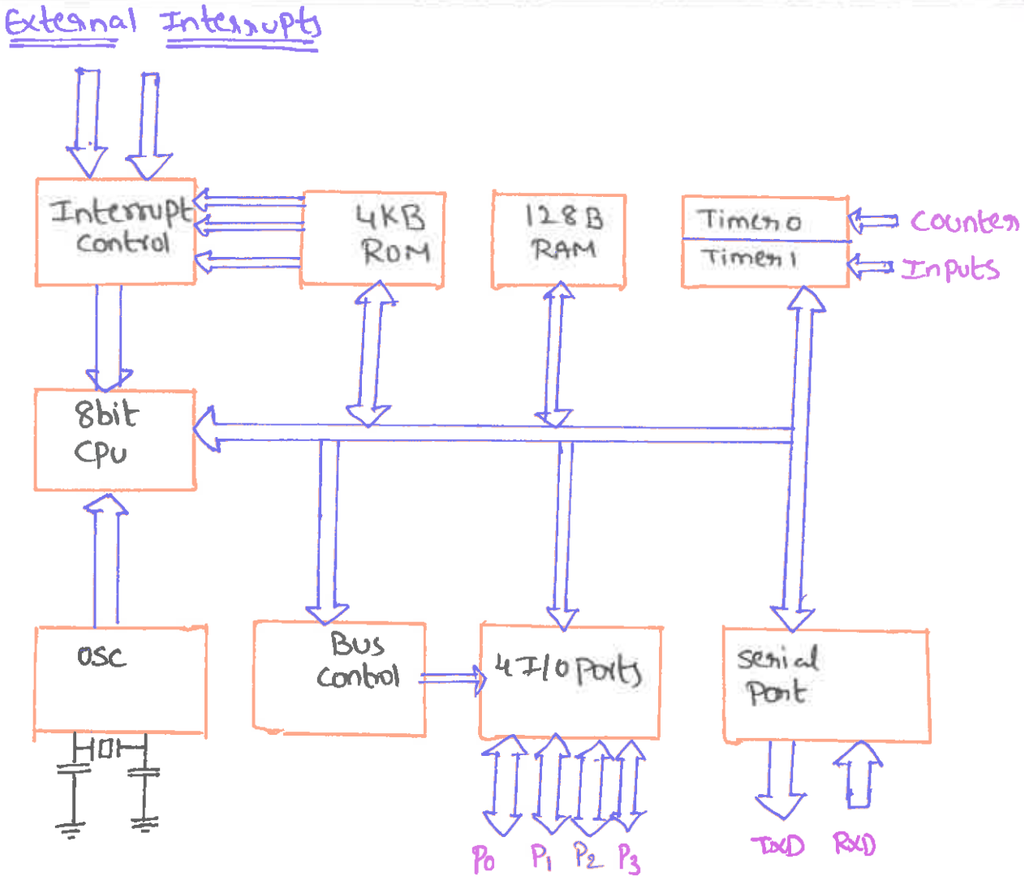
RL_1	RL_0	Function
0	0	Counter latching
0	1	Read/Load LSB only
1	0	Read/Load MSB only
1	1	Read/Load LSB first then MSB

Applications

* Used in Clock
* Rocket launch
* Time bomb
* Household Appliances

MICROCONTROLLER 8051 Architecture

Block Diagram of 8051



Criteria in choosing a microcontroller:-

* Speed
* Amount of RAM
* Amount of Rom
* No. of I/O Ports
* Timers
* Packaging Size
* Power Consumption
* Easy to upgrade.

Features of 8051 :-

* Harvard architecture
* Single chip microconboller
* Developed by INTEL in 1980
* 4kB Internal Rom
* 128 bytes internal RAM
* Four 8-bit I/o Ports ($P_0 - P_3$)
* Two 16 bit timer/counter
* 8 bit data bus & 16 bit address bus
* only 1 on Chip oscillator (external crystal)
* 6 interrupt sources (Reset, 2 external & 3 internal)

8051 Registers :-

* A (Accumulator) & B-Registers
* Psw (Program Status word)
* SP (Stack Pointer)
* Pc (Program Counter)
* DPTR (Data Pointer)
* I/o Port registers ($P_0 - P_3$)
* SBuf (Serial Data Buffer Registers)
* Timer Registers (TH_0, TH_1, TL_0 & TL_1)
* Power & Port Control (PCON & SCON)
* Interrupt Control Register (IP & IE)
* Timer control Register (TCON & TMOD)

Sample and Hold circuit and Multiplexers

* These are used in Data acquisition System
* Sample and Hold is part of Analog to Digital Conversion Techinque.
* Mux is used to select Multiple input into Single

Pin Diagram :-

Port 0 :-

* 8 bit R/w general purpose I/o ($P_{0.0} - P_{0.7}$)
* On multiplexed Low byte address & data bus.

Port 1 :-

only
* 8 bit R/w general purpose I/o ($P_{1.0} - P_{1.7}$)

Port 2 :-

* 8 bit R/w general purpose I/o ($P_{2.0} - P_{2.7}$)
* On High byte Address bus

Port 3 :-

* 8 bit R/w general purpose I/o ($P_{3.0} - P_{3.7}$)
* For communication with External Peripherals.

MEMORY ORGANIZATION AND SPECIAL FUNCTION REGISTERS

Special Function Registers in 8051 (SFR)

Each and every register has its own function and specific address

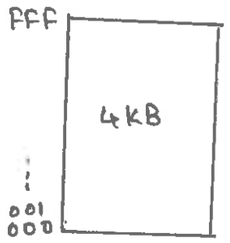
Symbol	Name	Address(H)
ACC	Accumulator	0E0
B*	B register	0F0
PSW*	Program status word	0D0
SP	stack pointer	81
DPTR	Data Pointer	
DPL	Low Byte	82
DPH	High Byte	83
P0*	Port 0	80
P1*	Port 1	90
P2*	Port 2	0A0
P3*	port 3	0B0
IP*	Interrupt Priority control	0B8
IE*	Interrupt Enable Control	0A8
TMOD	Timer Mode control	89
TCON*	TimerControl	88
TH0	Timer 0 High byte	8C
TL0	Timer 0 Low byte	8A
TH1	Timer 1 High byte	8D
TL1	Timer 1 Low byte	8B
SCON*	Serial Control	98 H
SBUF	Serial data Buffer	99
PCON	Power control	87

* Bit addressable

Memory organization and SFR of 8051

ROM (Read only Memory)

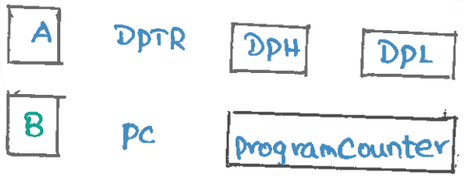
* 4KB of ROM (onchip)



RAM (Random Access memory)

* 128 bytes
  * Register Bank - 32 bytes
  * Bit addressable - 16 bytes
  * General purpose - 80 bytes

Registers in 8051



* Name of the registers

R0, R1, R2, R3, R4, R5 R6 & R7

7F	General purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default Register Bank0 R0 - R7							
00								

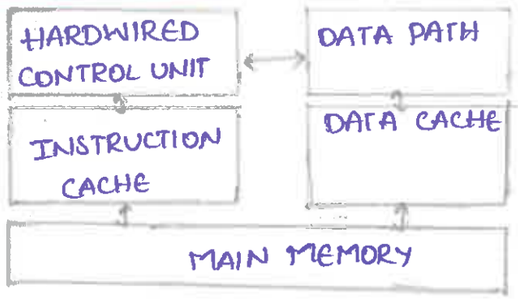
RISC ARCHITECTURE

RISC ARCHITECTURE

RISC - REDUCED INSTRUCTION SET COMPUTER

- SIMPLE INSTRUCTION SET & ADDRESSING MODES
- "ONE" WORD" in memory
- Faster excution & Take one clock cycle per Instruction.

ARCHITECTURE: BERKELEY RISC



- Implemented using Hardwired control unit [HCU]
- control signals of processors hardware.
- emphasizes on using Registers.
- Registers are placed on ALU.
- Control unit are placed on processor.
- Risc Instructions operate on operands present in processor's registers
- No control Memory - Due to simple & Execute one Instruction per cycle.
- No control store because Instruction are hardwired.

RISC INSTRUCTION SETS:

- Engages a single memory word.
- Operates on processor Register.
- Arithmetic & logical operation have their operand either in processor Register/Directer in the Instruction.

Operands in Registers Windows

ADD R_2, R_3

ADD R_2, R_3, R_4

Operands Mentioned directly in Instructions

ADD R_2, 100.

START OF EXECUTION OF PROGRAM:

All the operands are in memory.

To Access Memory IN RISC SYSTEM

RISC is has load & store

Load Instruction:

Loads the operand present in memory to the processor Register.

Load destination, SOURCE.

Load R_2, A // Memory to Register.

STORE INSTRUCTION: Store the operand Back to the memory

- Used to store Intermediate result

Store Source, Destination

Store R_2, A // Register to memory

SIMPLE ADDRESSING MODE:

1. IMMEDIATE ADDRESSING MODE:

Specifies the operand in the Instruction

Add R_4, R_2, #200

→ add 200 to the content of R_2 & Store the result in R_4.

2. Register Addressing mode: describes the Register holding the operand

→ Add R_3, R_3, R_4

→ Add the content of R_4 to the content of R_3 & store the Result in R_3

3) Absolut Addressing mode:

Describes name of some memory location in the Instruction. It is used to to declare global variables

→ Allocate memory to varible A, B, SUM.

4) REGISTER INDIRECT ADDRESSING MODE: Load R_2, (R_3)

5) INDEX ADDRESSING MODE:

Load R_2, 4(R_3)

→ load R_2 with content present at th location obtained by adding 4 to the content of Register R_3.

PIPELINING IN ARM:

- pipelining is the mechanism used by RISC
- To execute Instruction by speeding by fetching the Instruction.
- The other Instruction are being decoded & EXECUTED
- Allows memory system & processor to work continuously

WINDOWS AND PARAMETER PASSING & OVERFLOW

Fetch Decode Execute

TIME
clk 1 ADD →
clk 2 SUB → ADD →
clk 3 CMP → SUB → ADD

Stages Stage Stage3

- fetch loads an Instruction from memory.
- Decode identifies the Instruction to be executed
- Execut process the Instruction & write the result back to the Register.

CHARACTERISTICS:

- ARM pipeline doesn't process an Instruction unit it pass completly through the execution stage
- In the execution stage, PC Always Points to be Instruction address + 8 bytes.

RISC Revolution

ARM7: - 3 stage pipeline - 3 cycles
- process 32 Bit data

ARM-9 - 5 Stage pipeline - 5 cycles

Fetch Decode Execute Memory write

[LS1] [LS2]

Instruction from memory | decode Instruction | ALU | cond/ store | Edited (zero/ sign)

ARM 10 - SIX stage pipeline - 6 cycles

Fetch → Issue → Decode → Execute → write ← Memory

GENERAL CHARACTERISTICS:

- In Thumb state, PC Always points to the Instruction Add + 4 byte
- Executing branch Instruction PC Causes the ARM core to FLUSH
- On INTERRUPT - An Instruction in the execution stage will complete

Note: Increase in stages, Increase in efficiency

ARM ARCHITECTURE

ARM ARCHITECTURE

- IT IS A FAMILY OF REDUCED INSTRUCTION SET COMPUTING [RISC] ARCHITECTURES FOR COMPUTER PROCESSORS
- ARM - ADVANCED RISC MACHINES

WHY ARM? THIS PROCESSOR REQUIRES MINIMAL NUMBER OF INSTRUCTIONS R OPERATES ON VERY LOW POWER.

WHY ARM IS SO EFFICIENT? - DUE TO INCRESING IN INTERNAL DATA WIDTH [FROM 32 to 64 BITS] ADDITION OF EXTRA INTERNAL REGISTERS.

ARM FEATURES: LOAD/STORE - BASED ARCHITECTURE, SINGLE-CYCLE INSTRUCTION EXECUTION, 16X32 BIT REGISTER FILE, UNIK REGISTER, EASY DECODING & P. BELLMAN, POWER-INDEXED ADDRESSING MODES & FIXED 32-BIT INSTRUCTION SET.

NAMING ARM

ARM x y z TDMIEJFS

- x: Series
- y: MMU, CACHE
- z: THUMB
- TDMIEJFS: T - THUMB, D - DEBUG, M - MULTIPLIER, I - Embedded ICE, E, J, F, S

TYPES of ARM ARCHITECTURE:

ARM7TDMI →
- 3 PIPELINE STAGES [FETCH DECODE EXECUTE]
- HIGH CODE DENSITY
- LOW POWER CONSUMPTION

ARM9TDMI →
- COMPATIBLE WITH ARM7
- 5 PIPELINE STAGES [F|D|E|M|W]
- SEPERATE INSTRUCTION & DATA CACHE

ARM11 →
- 8 PIPELINE STAGES (8 DIFFERENT PROCESSING STAGES)
- FORWARDING: REDUCE DELAY
- PREDECTING: BRANCH PREDECTING

ARM DESIGN RULES:

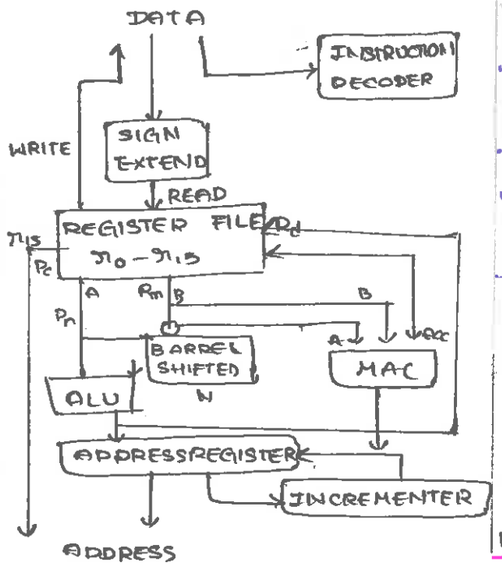
RISC: SIMPLE BUT POWERFUL INSTRUCTION THAT EXECUTE WITHIN A SINGLE CYCLE AT HIGH CLOCK SPEED.

INSTRUCTION: REDUCED SET | SINGLE CYCLE | FIXED LENGTH

PIPELINE: DECODE IN ONE STAGE | NO NEED FOR MICR CODE

REGISTER: A LARGE SET OF GENERAL PURPOSE REGISTER.

LOAD/STORAGE ARCHITECTURE: DATA PROCESSING INSTRUCTION APPLY TO REGISTERS ONLY; LOAD STORE TO TRANSFER DATA FROM MEMORY.



REGISTER BANK: LOAD/STORE ARCHITECTURE

- DATA PLACED IN REGISTRE FILE [32 bit]
- ARM CORE 32 BIT PROCESSOR. INSTRUCTION TREAT THE REGISTER AS HOLDING SIGNED/UNSIGNED 32 BIT.
- SIGN EXTEND: SINGLE LOAD INSTRUCTIONS

Eg: LDRSB & LDRSH.

REGISTER FILE CONTAINS:

- GENERAL PURPOSE REGISTER USED BY ALU/MAC
- REGISTER FILE CONNECTED TO PROCESSING UNIT THROUGH BUS A & B
- PROGRAM COUNTER [r_{15}] CONNECTED TO ADDRESS REGISTER

ADDRESS REGISTER: CONTAINS ADDRESS FROM WHICH DATA/INSTRUCTION NEEDS TO BE FETCHED.

- IT IS CONNECTED TO INCREMENTED UNIT.

REGISTERS: 16 REGISTERS FOR SPECIFIC MODE

A MODE COULD ACCESS:

- A PARTICULAR SET OF r_0-r_{12}
- r_{13} (SP - Stack pointer)
- r_{14} (l_r - link Register)
- r_{15} (P_c - program counter)
- CURRENT PROGRAM STATUS REGISTER [CPSR]
- THE USES OF r_0-r_{13} ARE ORTHOGONAL.

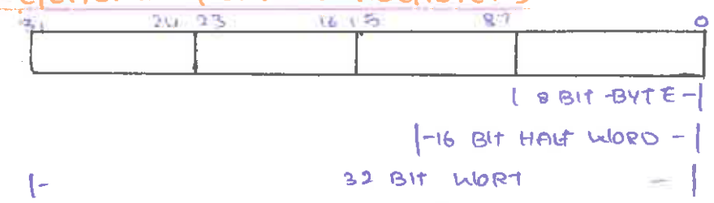
ARM ARCHITECTURE

- ARM7 - VAN NEUMAN ARCHITECTURE
- SAME BUS IS USED TO LOAD INSTRUCTION & DATA
- INPUT DATA BUS IS CONNECTED TO
  - DIRECT REGISTER BANK
  - SIGN EXTEND HARDWARE BLOCK
  - INSTRUCTION DECODE BLOCK.

BASIC PROCESSING UNITS

- ALU
- BARREL SHIFTER
- MAC [Multiply & accumulate unit]

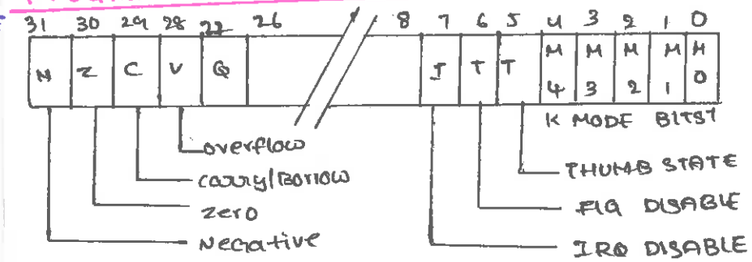
GENERAL PURPOSE REGISTERS



- 6 DATA TYPES [SIGNED/UNSIGNED]
- ALL ARM OPERATIONS ARE 32 BIT.

PROGRAM COUNTER:
- STORE THE ADDRESS of THE INSTRUCTION TO BE EXECUTED.
- ALL INSTRUCTIONS ARE 32 BIT WIDE & WORD ALLIGNED
- THE LAST TWO BITES of PC ARE UNDEFINED.

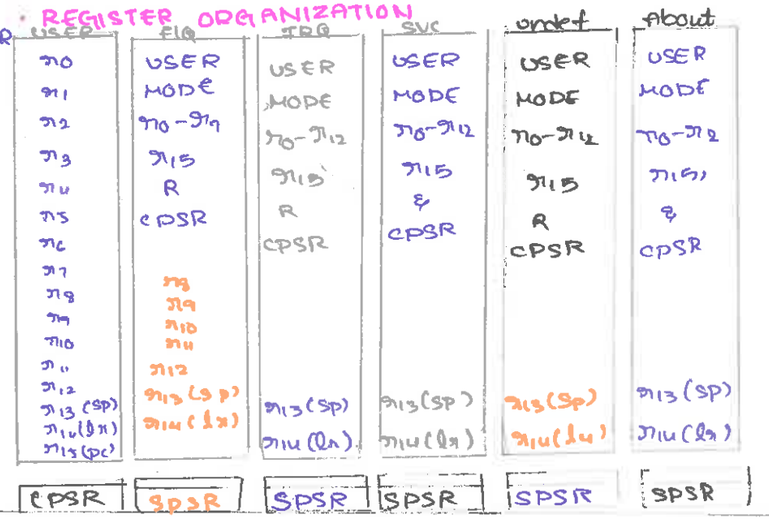
PROGRAM STATUS REGISTER [CPSR]



PROCESSOR MODES:

PROCESSOR MODE		DESCRIPTION
USER	USR	NORMAL PROGRAM EXECUTION MODE
FIQ	FIQ	HIGH SPEED DATA TRANSFER
IRQ	IRQ	GENERAL PURPOSE INTERRUPT HANDLING
SUPERVISOR	SVC	PROTECTED MODE FOR OS
ABORT	ABT	IMPLEMENTS VIRTUAL MEMORY
UNDEFINED	UND	SUPPORTS SOFTWARE EMULATION
SYSTEM	SYS	RUNS PRIVILEGED OS TASKS

REGISTER ORGANIZATION



ARM INSTRUCTION SET

ARM INSTRUCTION SET

PROGRAMMER MODEL: ARM IS DETERMIND BY THE CONTENT OF VISIBLE REGISTERS & MEMORY

USER MODE PROGRAM: USES 15 32 BIT GENERAL PURPOSE REGISTER[R_0-R_{14}], PC & CPSR

INSTRUCTION SET: DEFINES THE OPERATIONS THAT CAN CHAINGE THE STATE.

MEMORY SYSTEM:

0x00000000	00
0x00000001	10
0x00000002	20
0xFFFFFFFD	00
0xFFFFFFFE	00
0xFFFFFFFF	00

MEMORY REFERENCE INST

- IT IS A LINEAR ARRAY OF BYTES ADDRESSED FROM 0 TO $2^{32}-1$
- WORD, HALF-WORD, BYTES
- LITTLE ENDIAN

BYTE ORDERING:

1) BIG ENDIAN
   - least significant byte has highest order(address)
   - WORD ADDRESS 0x00000000
   - VALUE: 00102030
2) LITTLE ENDIAN
   - least significant BYTE HAS LOWEST ADDRESS WORD ADDRESS 0x00000000
   - VALUE: 30201000

0x00000000	00
0x00000001	10
0x00000002	20
0x00000003	30
0x00000004	FF
0x00000005	FF
0xFFFFFFFD	00
0xFFFFFFFE	00
0xFFFFFFFF	00

ARM PROGRAMMER MODEL:

R_0	R_1	R_2	R_3
R_4	R_5	R_6	R_7
R_8	R_9	R_{10}	R_{11}
R_{12}	R_{13}	R_{14}	PC

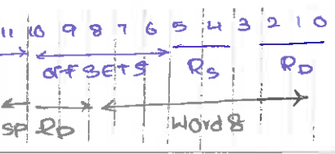
31	30							8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	//			I	F	T	M4	M3	M2	M1	M0	

0x00000000	00
0x00000001	10
0x00000002	20
0x00000003	30
0xFFFFFFFD	FF
0xFFFFFFFE	00
0xFFFFFFFF	00
	00

INSTRUCTION SET:

- ARM INSTRUCTION ARE ALL 32 BIT LONG [EXCEPT FOR THUMB MODE]
- 2^{32} POSSIBLE MACHINE INSTRUCTIONS

EXAMPLE:	FORMAT
MOVE SHIFTER REGISTER	01
load address	12



FEATURES:

LOAD STORE ARCHITECTURE, 3 ADDRESS INSTRUCTIONS, CONDITIONAL EXECUTION, LOAD/STORE MULTIPLE REG, & COMBING SHIFT & ALU OPERATIOIN IN SINGLE INS.

INSTRUCTION SET:
- DATA PROCESSING
- DATA MOVEMENT[MEMORY ACCESS]
- FLOW CONTROL

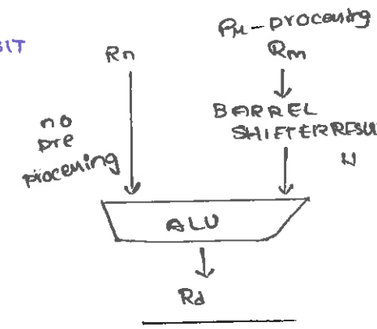
DATA PROCESSING:
- MOVE, ARITHMETIC, LOGICAL, COMPARISON R
- MULTIPLY INSTRUCTIONS
- PROCESS ONE OF THEIR OPERANDS USING BARREL SHIFTER

RULES:
- ALL OPERANDS ARE 32BIT
- 3 ADDRESS FORMAT

FORMAT: MOV<<cc><S>Rd <operand>

MOVes R_0, R_1 @ if carry is set @ the $R_0 := R_1$

MOVS R_0, #0 @ $R_0 := 0$ @ Z=1, N=0 @ C, V unaffected



CONDITIONAL EXECUTION: **SEQUENCE CONTROL**

MOV CS R_0, R_1 — CS – carry set; Mnemonic; condition

REGISTER MOVEMENT: **DATA MOVMENT**

SYNTAX:- <instruction>{<and>}(S) Rd, N

immediate, register, shift

MOV – Move 32 bit value into a register → $R_d = N$

MVN – Move the NOT of 32 bit value into a Reg → $R_d = \Omega N$

Eg: MOV R_0, R_2 @ $R_0 = R_2$; MVN R_0, R_2 @ $R_0 = \Omega R_2$

ADDRESSING MODES:

Register operands — ADD R_0, R_1, R_2

Immediate operand — ADD R_3, R_3, #1 @ $R_3 := R_3 + 1$

SHIFTER REGISTER OPERANDS

- FAST MULTIPLICATIONS
- MUL, CLZ, & ADD- DO not need bowel shifter

Eg: LSL → Logical shift left → x LSL4 x << 4

C ← Register ← 0

① MOV R_0, R_2, LSL #2 @ $R_0 := R_2 << 2$

② MOV R_0, R_2, LSR #2 @ $R_0 := R_2 >> 2$

③ MOV R_0, R_2, ASR #2 @ $R_0 := R_2 >> 2$

④ MOV R_0, R_2, ROR #2 @ $R_0 := R_2$ ROR @ R_2 – unchanged

RRX → Rotate Right extended.

MULTIPLICATION: MOV R_1, #35; MUL R_2, R_0, R_1

ARM Branch Instructions

* Flow control
* Sequence control

ARITHMETIC
- ADD → ADD R_0, R_1, R_2 @ $R_0 = R_1 + R_2$
- SUB → SUB R_0, R_1, R_2 @ $R_0 = R_1 - R_2$
- SBC → SBC R_0, R_1, R_2 @ $R_0 = R_1 - R_2 + C$

LOGICAL: AND, ORR, EOR, BIC [BIT CLEAR]

COMPARISON: <INS>{<COND>} R_n, N

BRANCH OPERATION FOLLOWS TO CHANGE

Eg: CMP R_1, R_2 @ SET CC ON $R_1 - R_2$

FLOW CONTROL INSTRUCTION: B{<cond>} label branch, Bloch with wive(B L), Branch exchang.

DATA TRANSFER INSTRUCTIONS:-

MOVE DATA BETWEEN REG & MEMORY

① SINGLE REG WADISTORC.

<LDR/STR>{<COND>}{B} Rd, addressing

ADDRESSING MODES:

MEMORY → BW REG & OFFSET → LDR R_0, {R_1} @ mem[R_1]

① Pre Index ② AUTO Index & ③ post Index

MULTIPLE LOAD/STORE REGISTER:

LDM, STM Eg: LDMIA R_0!, {R_1-R_3}

SWAP INSTRUCTION:

SWP{B}{<cond>} Rd, Rm, [Rn]

SWOP between memory & Reg.

LOAD CONSTANT: LDR Rd, =CONSTANT

ADR Rd, label.

Load 511 into R_0

MOV R_0, #256

add R_0, #255

- load from memory, declare L511 word 511

ldr R_0, L511 → ldr R_0[P_c, #0]

ASSEMBLER DECIDES

ldr R_0, =255 → MOV R_0, 255

ldr R_0, =511 → ldr R_0, [P_c, #4]

PC RELATIVE MODES:

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	0
COMD	0	I	#	P	U	B	W	L		Rn		Rd		OFFSET	

F D E

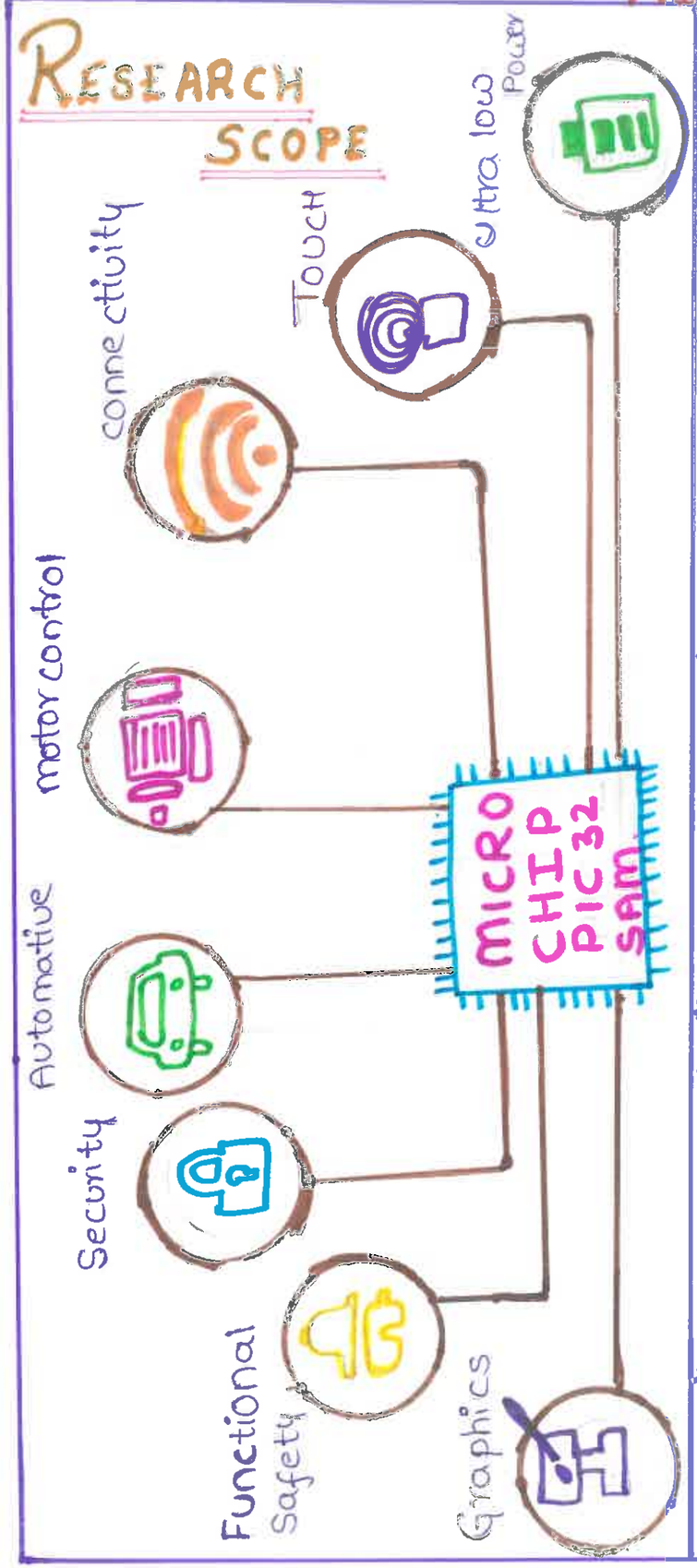
F D E

F D E

Eg: MOV R_0 #0; ADR R_1, @; STR R_0, [R_1]; PC → SWI # 11 @ word 100 : end.

Advances And Applications

Research Scope



Fields Where Microprocessors Are Applicable

* Household Devices

→ Programmable Thermostat

→ High-end Coffee Makers, Washing Machines, Radio Clocks, Ovens, Toasters, Televisions, VCRs, DVD Players & Stereo Systems.

→ Home Computers, Hand-held game devices, Video game systems and Home-lighting systems.

* In Medicals & Instrumentation

→ Insulin pump

→ Processing data from Bio Sensors, storing measurements and analyzing results.

→ function Generators, Frequency Counters, Frequency Synthesizers, Spectrum Analyzer

* Communications

→ Telephone Industry - Digital Telephones, Telephone Exchanges & Modem

→ Satellite Communication - Television, Tele Conferencing

→ Railway Reservation & Airline Reservation system

* Industries

→ Process Control, Control of Machine & Equipments

→ Measurement, Display & Control of current, Voltage, Frequency, Temperature, Stress & So on.

→ lift Control, Industrial tool Control & Robotics.

Practical Significance

* Home Automation System
* War field Spying Robot
* Embedded system for Vehicle Tracking
* Traffic Signal Control System
* Street light Control

Applications Of Microcontroller

* Computer Printers, Plotters, Fax Machines, Photocopiers and Automotive Engine Control Mechanisms.
* Electronic Instruments - Oscilloscopes, Multimeters & Ic Testers.
* Any Advanced Automobile has 25 or more microcontroller in different Control applications.
* Microcontrollers occupy about 80% of all CPU markets in the World.

Research Ideas

* https://shakti.org.in
* https://www.electronicshub.org/embedded-systems-projects-ideas/
* https://www.microchip.com/design-centers/microcontrollers

Programming Tools

* Emulator 8085/86
* Assembler 8086
* KEIL 8051
* Microchip PIC

SIMATS

SCHOOL OF ENGINEERING

Approved by AICTE | IET-UK Accreditation

Engineer to Excel

Saveetha Nagar, Thandalam, Chennai - 602 105, TamilNadu, India