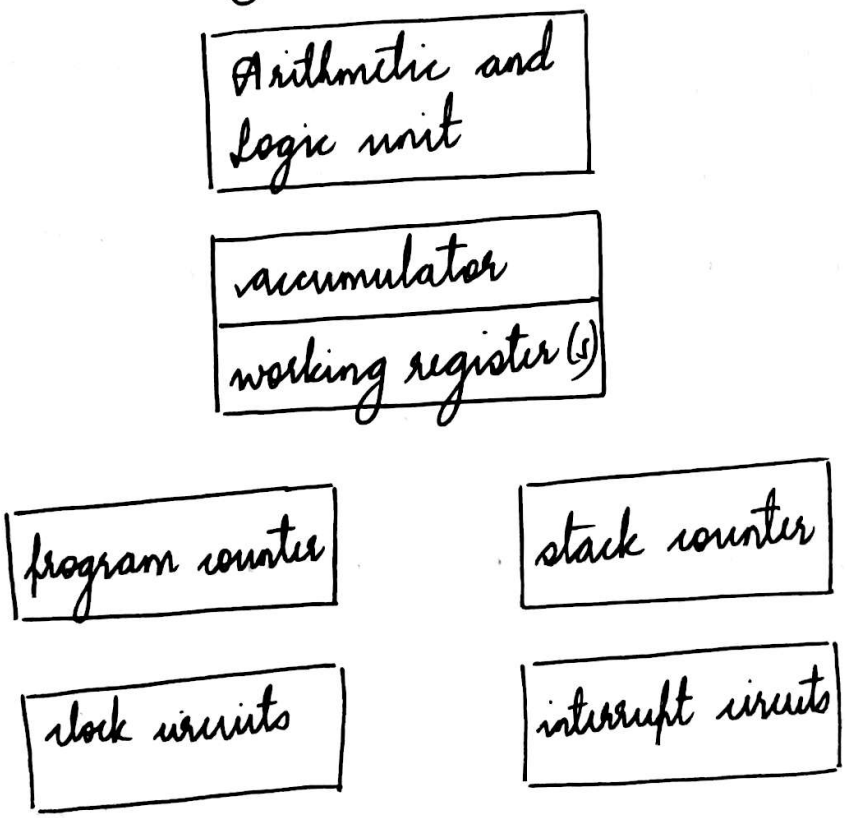
MICROPROCESSOR

A microprocessor, is a general-purpose digital computer central processing unit (CPU). Although popularly known as a "computer on a chip". The prime use of microprocessor is to read data, perform extensive calculations on that data, and store those calculations in a mass storage device or display the results for human use.

Block diagram of Microprocessor

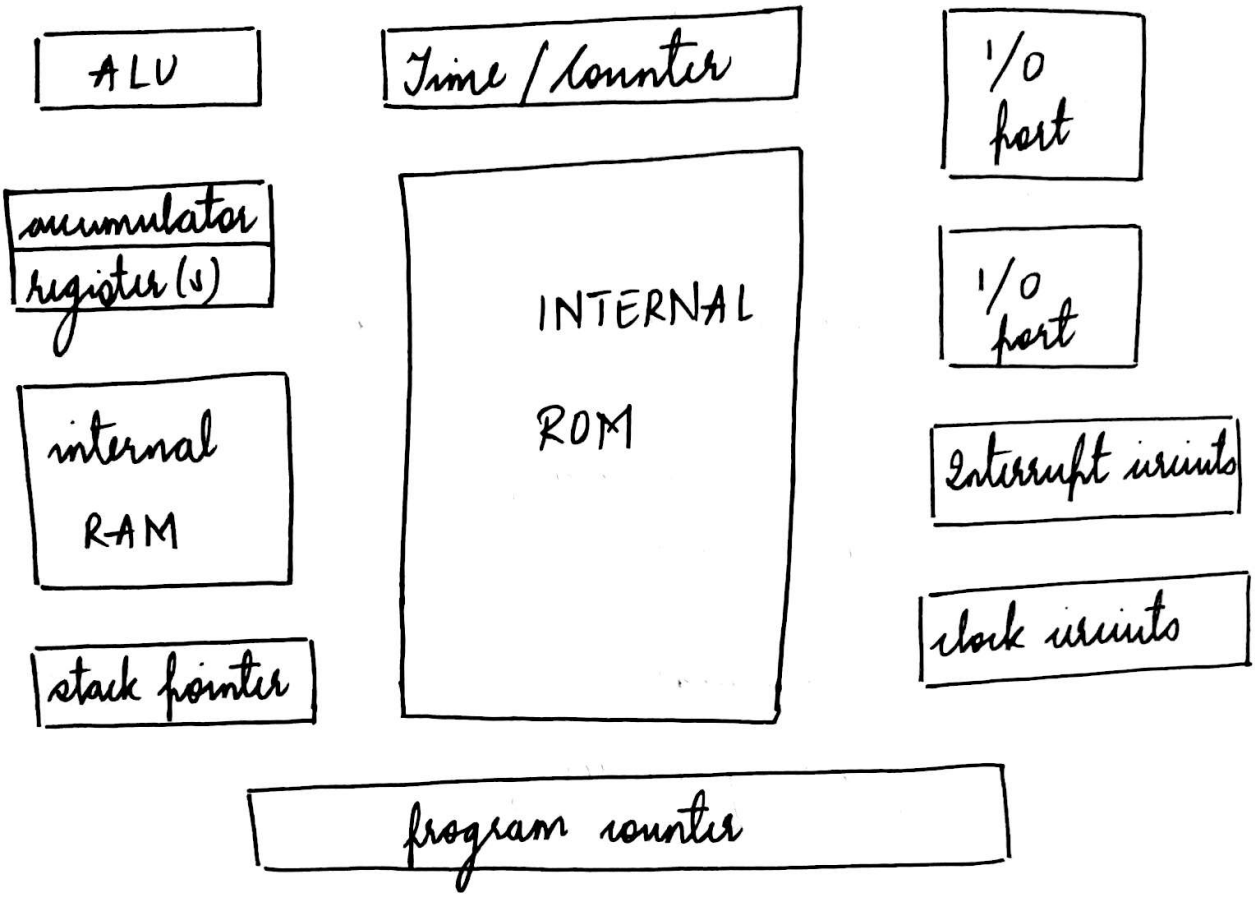


MICROCONTROLLERS

A microcontroller is a general- purpose device, but one that is meant to read data, perform limited calculations on that data, perform limited calculations on that data and control its environment based on those calculations its main use is

to control the operation of a machine using a fixed program that is stored in ROM and that doesn't change over the lifetime of the system.

MICROCONTROLLER (Block diagram)



Comparision of microprocessor and Microcontrollers

Microprocessors	Microcontrollers
i) They have many operational codes for moving data from internal memory to the CPU.	They have one or two o/p codes for moving data from internal memory to CPU.
ii) They have one or two bit handling instructions.	They have many.

iii) It is concerned with rapid movement of code and data from external addresses to the chip.

It is concerned with rapid movements of bits within the chip.

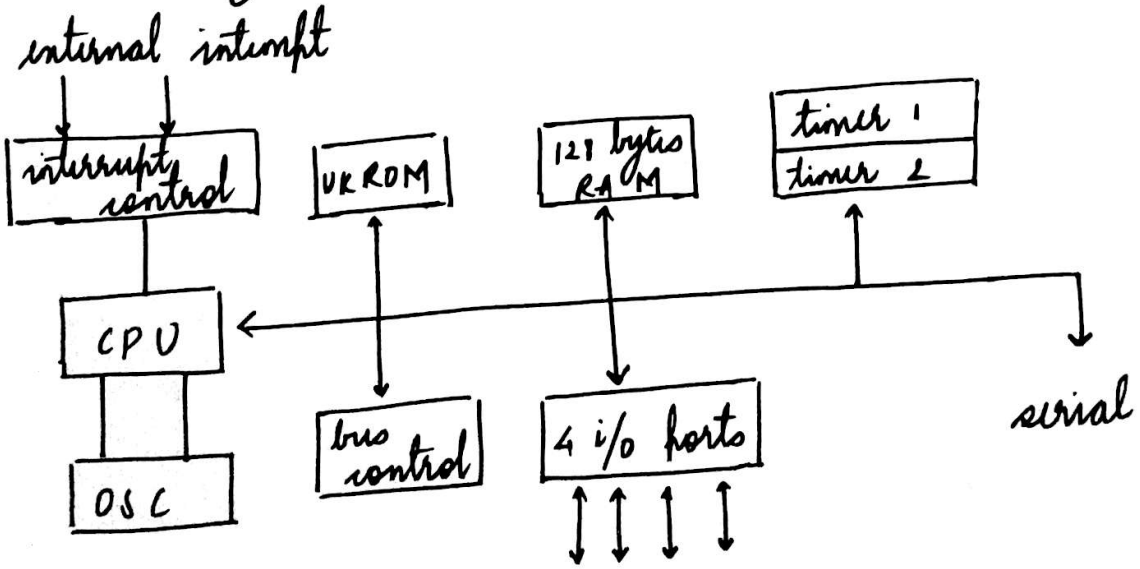
iv) It can function as a computer with addition of no external digital parts.

It must have many additional parts to be operational.

8051 Basic Components

- 4KB ROM
- 128 B RAM
- four 8-BIT I/O ports
- 2 16-BIT timers/counters
- One serial interface
- 6 interrupt sources

Block diagram



8051 flag bits and PSW register

CY	PSW.0	carry flag
AC	PSW.1	ALX carry flag
F0	PSW.2	general purpose
RS1	PSW.3	register bank selector bit 1
RS0	PSW.4	register bank selector bit 0
OV	PSW.5	overflow flag
-	PSW.6	user defined
P	PSW.7	parity bit.

Basic operation of processor

i) Fetch - Bring the next instruction from memory into the processor.

ii) Decode - To translate data from a code into the original language or form.

iii) Execute - Do it.

Microcontroller architecture and programming codes

Microcontrollers

Bits	memory		instructions	memory architecture		
4 8 16 32	embedded	external	CISC	RISC	princeton	Harvard

family

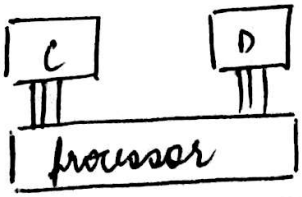
8051 motorola PIC texas national ARM others

Intel Atmel Dallas Phillips siemens

RISC - Reduced instruction set computer architecture.

CISC - Complexed instruction set computer architecture.

Harvard - There is a different block of memory for code and data



Princeton → Have common block for code and data

In case of Harvard, 2 things can be done parallely in princeton one has to wait for the completion of the other. Harvard required more space on ROM but princeton requires less space.

MCS-51 family of MC

→ 8 bit processor (8051) → basic version.
→ 4k bytes of on-chip ROM instruction memory.
→ Data storage and the stack
→ 2 times, one serial port, four 8 bit, parallel I/O ports
→ speeds (frequency) starting from 12MHz.

Architectural needs of a microcontroller

→ Lets consider what architectural features could be needed in a microcontroller.

- What are expected applications?

→ sensing the environment
- I/P

→ The response may be displayed
- Timer / counter

→ prioritized response
- Interrupts (disturbs processing)

these are used to disturb the processes which went wrong.

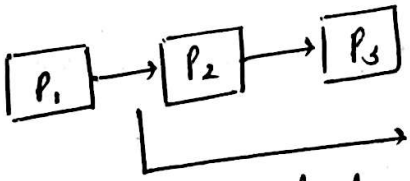
- software to control the process
  - non-volatile memory (ROM)
- Temporary data
  - RAM

Register → A proper formatted memory. It has different capacity but for a particular microcontroller they are fixed.

The processor

Registers

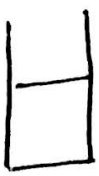
- General purpose → used as programmer requires
- special purpose → can only be used for specific function
  - stack pointer, - accumulator - pointers



interrupt by storing some address of pervious process in stack in it. Jumps to process (interrupted)

stack - Data structures following mechanisms of, st in last out

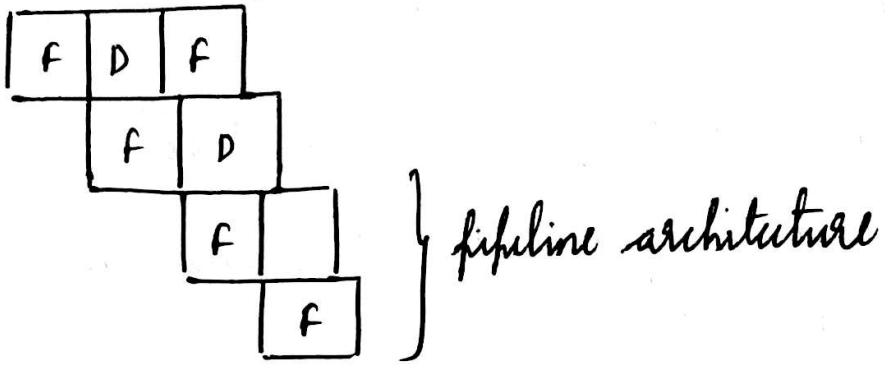
stack is used for arithmetic operators.



untill this it shows error.

search in stack is impossible.

pointers - pointing



} pipeline architecture

ALU

→ Made up of functional blocks that perform different arithmetic and logical operations on the operand.

→ flag → indicate something
eg- sign, carry, zero.

→ PSW - (program status word)
flags register are collectively known

Memory

RAM - Dynamic and static temporary memory.

Dynamic - set of box in home when switched off it again sets up (same channel will not play).

ROM - Erasable :- EPROM hard disc
Electrically erasable - EEPROM

MCS - 51 variants

More no. of interrupts, better is controller.

Read about flash memory

- It is an electronic non-volatile computer storage medium that can be electrically erased and reprogrammed.

when pointer is pointing at first address stack will be empty.

GUI - graphical user interface. 1 bit at a time can be sent through "serial interface".

RISC

Reduced Instruction set computing / architecture has a state set of attributes that allows it to leave a lower cycles per instruction that CISC.

CISC

Complex instruction set computing / architecture is a processor design where single instructions can execute several low-level operations load from memory, an arithmetic or are capable of multi-step operations or a addressing modes within single instructions.

RISC	CISC
i) Reduced instruction set architecture	Complex instruction set architecture
ii) Only few instructions are present	Many instructions are present.

iii)

iv)

v)

vi)

iii) Only few instruction addresses memory.	Most of the instructions address the memory.
iv) fixed length instructions	variable length instructions.
v) highly pipelined	less pipelined
vi) many register sets are present	few register sets are present

Harward Architecture	Princeton architecture
i) Has seperate data and instruction busses allowing transfers to be performed simultaneously on both busses	Has only one bus which is used for both data transfers and instruction fetches must be scheduled.
ii) It is possible to have two seperate memory systems.	They can't be performed at the same time.

Processor has controlling unit.

Note on pins of 8051

→ 8051 consists of I/O ports (P_0-P_3)
→ Port 0 (pins 32-39) : P_0 ($P_{0.0}$-$P_{0.7}$)
→ 8 bit R/W general purpose I/O.
→ OR acts as a multiplexed (same pin can acts as address, data and general purpose) low byte address, and data bus for external memory design.

→ Part 1 (pins 1-8) : P1 (P1.0 - P1.7)
→ only 8 bit R/W - general purpose I/O.
→ Part 2 (pins 21-28) : P2 (P2.0 - P2.7)
→ 8 bits R/W - general purpose I/O or high bytes of address bus for internal memory design.
→ Part 3 (pins 10-17) : P3 (P3.00 - P3.7)
  → General purpose I/O
  → Each port can be used as i/p or output. (bi - directional)
  → Its not using only one of the internal peripherals (timer) or external interrupts.

On chip RAM

→ There are 4 BANK's (are group of registers)
If there are 4 banks, then each bank has 8 resistors, 32 resistors totally in 4 banks.
locations → addresses
16 locations × 8 bits = 128 bits

Accumulator - ports the value of our operation.
2-16 bit register.

1) Program counter → points to the next instruction in ROM.
2) DPH → data pointer higherbyte
   DPL → data pointer lowerbyte
Data pointer contain address of internal and external

→ Hardware can exist without software.

ACC and B registers → 8 bit each stack pointer → 8 bit

→ Nibble - 4 bit

carry generated from 4th bit to 5th bit is called auxiliary carry.

→ If both RS0, RS0 are present then bank "zero" is selected.

→ parity :- Pair of one's.

F is represented in 4 bits.

∴ FF is 8 bits

Buffers - to communicate these are necessary.

"128 bytes" bytes RAM consists

Microcontroller is CISC architecture

We can connect "External ROM"

"RAM" has 128 bytes.

There are no instructions which has 4 bytes, we have only "1, 2, 3". 8051 microcontroller can use both internal and external memory. All addresses are hexadecimal in nature.

software is embedded in hardware or vice versa.

"Counter" makes "Timers".

We can design counter and timer.

8051 Instruction set

→ 8051 has 255 instructions

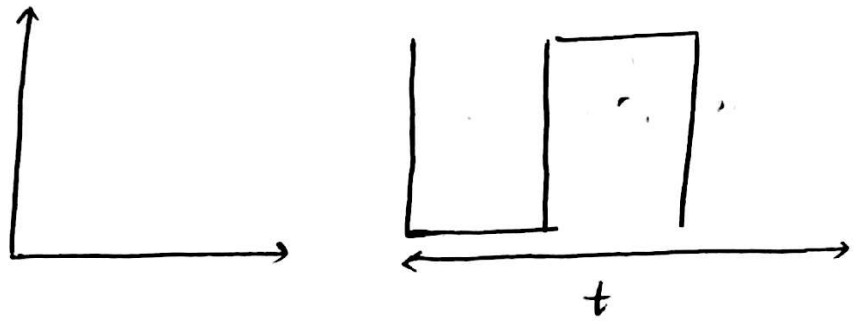
→ It is optimized for 8-bit control applications.

→ It is having 1 bit op-codes.

It is having 8 bit of-codes.

039 → 1 byte, 92 → 2 byte and 24 → 8 byte.

→ 2 times together form a state



$T = 1/f$

if $f = 11.0592$ MHz of crystal oscillator

$$T = \frac{1}{11.0592} = 0.094 \mu s$$

→ 1 state is as good as $2 \times T = 2 \times 0.0904 \mu s$

$= 0.18 \mu s$

1 Machine cycle (1 MC) $= 6 \times$ state

$= 6 \times 0.18 \mu$

1MC $= 1.085 \mu s$

How many machine cycle are there in 1 sec

Soln $\left(\frac{11.0592}{12}\right)$ MHz $= 0.921 \times 10^6$

$T = \frac{C \times 12}{\text{frequency}}$

8051 ADDRESSING MODES

i) Five addressing modes are available :-

→ Immediate
→ Register
→ Direct
→ Indirect
→ Indexed

There are three more modes :-

→ Relative
→ absolute
→ long

They are used with calls, branches, and jumps and are handled automatically by the assembly.

What is overflow ? when does it occur ?

→ Overflow occurs when there are insufficient bits in a binary number representations

Exceptions - Can't move one register to another.

Not all combinations are valid.

Ex. MOV R_2, R_1 (invalid)

→ only one bank can be accessed at a time controllable through bit R_{50} and R_{51} of the PSW.

MOV. PSW, # 00011000B

set R_{50}: R_{51} to 11, therefore accessing register bank 3.

Direct Addressing

Either the source and destination is direct address value.
- It can access any on-chip hardware register by their address or access any memory location by its address.
- All on chip memory locations and registers have 8-bit address.

Can we use the 8-bit address in the instruction ?

MOV A, 4H A ← mem [04H] (memory location)

Note :- No '#' sign here.
"04H" coz we need to know the actual address to transfer data

Yes, hexadecimal can be represented using 3 bits.
We can write address on distination also.

Indirect Addressing

- R_0 and R_1 may be used as pointer (we use to point to other location) registers where their contents indicate an address in internal RAM where the data is to be read and written.
- Indirect addressing is represented by a commercial 'at' sign (@) preceding R_0 or R_1.
- MOV R_1, #40H, 40H value is passing to R_1
- MOV A, @R_0,
- MOV @R_0, R_1

Limitations

We can't use other than R_0 and R_1 to point to other registers.

MOVx is used for external memory.

MOV is for internal memory.

Write a program to copy value, 40H in register R_0, R_1, memory address 3FH or @ location 3FH using min instructions and using all memory address

Immediate addressing

```
MOV R0, #40H
MOV R1, #40H
```

Register addressing

```
MOV A, 40H
MOV R0, A
MOV R1, A
```

Direct addressing

```
MOV R0, 40H
MOV R1, 40H
```

16 bit register points program counter is not present in ROM because it is the only register which doesn't have address/ location.

Write the previous code in a single instruction with all addressing modes.

```
MOV R0, #40H,
MOV A, R0
MOV R1, A
MOV 3FH, @R1
```

Can we use address location in distination?

Yes

Ex:- MOV @R0, R1 (This allows (shows) address can be distination)

We use op-code to perform.
Operations on operands — are registers
To perform operation, we need op-codes and operands.

Why do we need address?

Immediate Addressing

MOVX A, @R1; Move contents to external memory not internal

Instruction types

The instructions are grouped into 5 groups
- arithmetic
- logic
- Data transfer
- Boolean
- Branching

Steps :-

Kiel Mvision 3 → double click
IDE → Integrated Development Environment
Project → give name → save → choose target device → atmel (+ sign) → AT89C51ED2 → OK → NO → file → text → textfile opens (start program).

Directors → ORG, end
└→ program has to start from is memory location

Program :-

```
org 0000h
mov a, #12h
mov r0, a
end
```

Now go to file → save → file name movd1.asm → save (after saving it, come in dark black or bold letters). Right click on source group → add files to group source group 1 → change file type to asm → click on movd1 → click only once or add not double click.

To check for error :-

right click on file movd1.asm → click on build target. If there are errors it shows a blue arrow mark, near the error.

To execute

Debug → start debug session → OK.

Yellow arrow is PC (program counter)

To run - F11

NOP → no operation

Window is called disassembly window

To step → debugging → press step

To view memory window :-

view → memory window (where we can see any memory location)

Write an ALP to transfer a block of data from internal memory location 30h to 34h to 50h - 54h.

```
org 0000h
mov 30h, #0aah
mov 31h, #0bbh
mov 32h, #0cch
mov 33h, #0ddh
mov 34h, #0eeh
mov 30h, 40h
mov 31h, 41h
mov 32h, 42h
mov 33h, 43h
mov 34h, 44h
end
```

dptr is a register which is used to point to the external register
(16 bit)

```
mov dptr, #4050h
mov a, #12h
movx @dptr, a
movx a, @dptr
mov R7, a
end.
```

Write an ALP to transfer a block of data from external memory location 4000h to 4004h to internal memory 30h to 34h

```
org 0000h
mov dptr, #4000h
mov a, #0aah
movx @dptr, a
movx a, @dptr
mov 30h, a
mov dptr, #4001h
mov a, #0bbh
mov @dptr, a
mov a, @dptr
mov 31h, a
mov dptr, #4002h
mov a, #0cch
mov @dptr, a
mov a, @dptr
mov 32h, a
```

```
mov dptr, #4003h
mov a, #ddh
movx @dptr, a
movx a, @dptr
mov 33h, a
mov dptr, #4004h
mov a, #0cch
movx @dptr, a
movx a, @dptr
mov 34h, a
end
```

Write an ALP to exchange a block of data from 4000 h to 4004 h to internal memory location internal memory location

```
org 0000h
mov dptr, #4000h
movx a, @dptr
mov r0, a
mov a, 50h
mov 50h, r0
movx @dptr, a
mov dptr, #4001h
movx a, @dptr
mov r0, a
mov a, 51h
mov 51h, r0
movx @dptr, a
```

```
org 0000h
mov dptr, #4000h
movx a, @dptr
mov r0, a
mov 50h, r0
movx @dptr, a
```

```
mov dptr, #4002h
mov a, @dptr
mov r0, a
mov a, 52h
mov 52h, r0
movx @dptr, a
mov dptr, #4003h
mov a, @dptr
mov r0, a
mov a, 53h
mov 53h, r0
movx @dptr, a
mov dptr, #4004h
mov a, @dptr
mov r0, a
mov a, 54h
mov 54h, r0
movx @dptr, a
end
```

Write an ALP to add two 8 bit numbers

```
org 0000h
mov a, #0FBh
mov r0, #022h
ADD a, r0.
end
```

Write an ALP to add 16 bit numbers

```
org 0000h
mov R0, #0FBh
mov a, #022h
ADD a, R0
mov 51h, a
mov R2, #0AAh
mov a, #0BBh
ADDC a, R2
mov 50h, a
end
```

Write an ALP to multiply and divide two 8 bit numbers

```
org 0000h
mov a, #12h
mov b, #11h
mul ab
end
```

```
org 0000h
mov a, #20h
mov b, #30h
div ab
end
```

Write an ALP to add 5 numbers present in 50-54 memory location

```
org 0000h
mov a, #50h
mov R0, #51h
add a, R0
```

```
org 0000h
mov 50h, #11h
mov 51h, #22h
mov 52h, #33h
mov 53h, #44h
mov 54h, #55h
```

```
mov a, #10h
add a, 50h
add a, 51h
add a, 52h
add a, 53h
add a, 54h
end
```

Addition of 32-bit numbers

```
org 0000h
mov r0, #13h
mov a, #16h
addc a, r0
mov 50h, a
mov r0, #12h
mov a, #14h
addc a, r0
mov 9h, a
mov r0, #11h
mov a, #12h
addc a, r0
mov 48h, a
mov a, #11h
addc a, r0
mov 47h, a
end
```

Subtract two 16 bit numbers (let no.s be ab17 & 103a

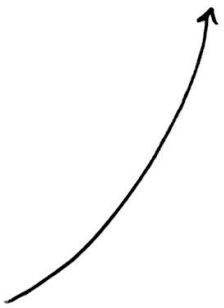
```
org 0000h
mov a, #17h
mov r0, #03ah
subb a, r0
mov 51h, a
mov a, #0abh
mov r0, #012h
subb a, r0
mov 50h, a
end
```

Block transfer using increment function

```
org 0000h
mov 50h, #0aah
mov 51h, #0bbh
mov 52h, #0cch
mov 53h, #0ddh
mov 54h, #0eeh
mov 60h, 50h
mov a, 60h
mov r0, 50h
inc a
inc r0
mov a, r0
inc a
inc r0
mov a, r0
inc r0
mov a, r0
inc a
inc r0
mov a, r0
end
```

Block transfer data using document operation

```
org 0000h
mov 54h, #0aah
mov 53h, #0bbh
mov 52h, #0cch
mov 51h, #0ddh
mov 50h, #0eeh
mov a, 54h
mov r0, 64h
mov r0, a
dec a
dec r0
mov r0, a
dec a
dec r0
mov r0, a
dec a
dec r0
mov r0, a
dec a
dec r0
mov r0, a
end.
```



subtraction

54h and 35h and operation logical operation on 8 bit.

54h
35h

2	54
2	27 - 0
2	13 - 1
2	6 - 1
2	3 - 0
	1 - 1

54h = 110110 b

2	35 h
2	17 - 1
2	8 - 1
2	4 - 0
2	2 - 0
	1 0

35h = 100011 b

```
  1 1 0 1 1 0 h
  1 0 0 0 1 1 h
---------------
  1 0 0 0 1 0
```

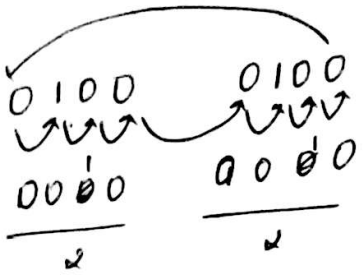
XRL - xor
ANL - and
ORL - or
CLR - clear
CPL A - compliment
RL A — How many time you rotate left it changes by 2^n when n - times, it will rotate left.

RLC A — Rotate left through carry

MJB - 0 +ve
MJB - 1 -ve

RRA

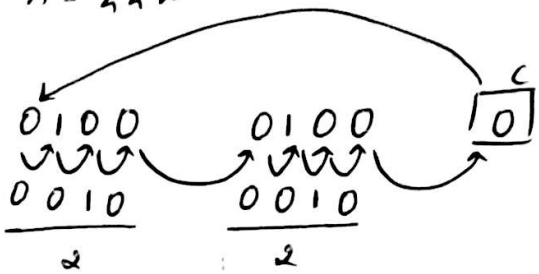
A = 44h



22h

RRC

A = 44h



LSB = 1 ~~odd~~

LSB = 0 even

XCH exchange 8 bit of memory

XCH a, Rn

XCH A, direct

XCH A, @ Ri

XCHD - exchange lower nibbles of two values

Boolean algebra

OR, XOR operate on single bit

SETB C (make C-1)

SETB bit (make bit-1)

jump → (change the flow of instruction)

conditional jump — non-conditional jump

Bit level — Byte level

JC (jump if carry is true)

By giving label to instruction, it jumps

carry is 1, it jumps down

JNC - jump on no carry.

JB - jump of bit

JB ___, back — TB=1 (true condition)
FB=0 (false condition)

JNB ___, back — TB=0
FB=1

JBC ___, back — it jumps if the condition is true, bfor jumping it clears anything.

JBC (jump on bit, clear)

Byte level

DJNZ R0, back

Dec | not

jump zero

Write an ALP to add 5 8 bit numbers stored in 20-24h.

If moved by 10h, we will get 16 numbers

simple one

```
org 0000h
mov a, 20h
add a, 21h
add a, 22h
add a, 23h
add a, 24h
end
```

```
org 0000h
mov r5, #05h
CLR C
mov r0, #20h
back: ADDC A, @r0
      INC r0
      DJNZ r5, back
      end
```

1) Write an ALP to find average of n numbers
2) Write an ALP to find GCD and LCM.
3) to find count no. of positive no in an array of 10 numbers
4) no. of negative no.s
5) no. of even no.s
6) no. of odd no.s
7) no. of zero in a given 8-bit number
8) no. of ones in a given 8 bit number
9) to find average of all +ve no.s
10) " " " all -ve no.s
11) " " " all even "
" " " all odd "

12) To check greater no among two no.s

13) " smaller " " " "

14) To check the greatest no in an array of 5 no.s

15) To " smallest no. " "

16) To arrange array in ascending

17) descending

Write an ALP to find factorial of 5 numbers

```
org 0000h
mov r0, #20h
clr c
mov b, r0
mov r5, #05h
back: mul ab
INC r0
DJNZ r5, back
end
```

```
org 0000h
mov clr c
mov r0, #05h
mov a, #01h
x: mov b, r0
mul ab
djnz r0, x
end
```

1)
```
org 00ch
mov r0, #40h
mov r4, #44h
clr c
go: add a, @r0
inc r0
DJNZ r4, go
mov b, #05h
div ab
end
```

2)
```
org 0000h
mov r0, #40h
mov r1, #10h
mov r2, #00h
go: RLC a
inc r.
mov a, r0
JNC next
inc r2
next: DJNZ r1, go
end
```