6 Assembly Language Programming:

I Addressing Modes is 8051 Mc

1. Immediate Addressing mode

Data is given in the instructions → 8-bit Reg

Immediate → Data in instruct' → eg: mov A, #12 (A ← 12)

#

mov DPTR, #3000H (DPTR ← 3000H

↳ 16-bit Reg.

2. Register Addressing mode : Data is given in register.

Register → data in Reg → eg: mov A, R0; A ← R0

mov R1, A; R1 ← A

mov R1, R0 is not allowed

3. Direct Addressing mode

Address is given in instruction. Address in instruction.

Direct → eg: mov A, 25H

Only for int. RAM & SFR

A ← [25H]

8 bit address

eg: mov 30H, A [30H] ← A

mov 20H, #20H; [20H] ← 20H

mov 01H, 00H; [01H] ← [00H]

↳ not in dp

↳ only in dir

mov A, 80H; A ← P0 {SFR?.

4. Indirect

1. Internal RAM - 8 bit address (R0/R1)
2. External RAM 16 bit address (DPTR)
3. External RAM - 8 bit address (R0/R1)

* Int. RAM restricted in R0 & R1 reg)

1. Int. RAM 8-bit address (R0/R1)

eg: mov A, @R0, A ← [R0]

A get the contents of memory location R0.

mov @R1, A ; [R1] ← A

eg: Access data from series of locations

Block transfer pgm.

2. External RAM - 16 bit address (DPTR)

eg: movx A, @DPTR ; A ← [DPTR]

movx @DPTR, A, [DPTR] ← A

↓ External RAM)

3. External RAM - 8 bit address (R0/R1)

*Acess 8-bit address

* eg: assume higher bit is zero

eg: movx A, @R0, A ← [R0]

movx @R1, A, [R1] ← A

@R0 - ~~data inside~~ memory location of R0

Instruction Syntax

Concept of the opcode and operands in an instruction

LABEL: opcode operand; comment

LABEL

Label is a symbolic address for the instruction when the pgm is assembled the label will be given the specific address in which that instruction is stored

OPCODE: is the symbolic representation of the operation the assembler converts the opcode to a unique binary code (machine language)

OPERAND: while opcode specifies what operation to perform, operand specifies where to perform that action. The operand field generally contain the source and destination of the data. The operand will be either address or data

8051 addressing modes

1. Immediate
2. Register
3. Direct
4. Indirect ad
5. Indexed addressing
6. Absolute addr
7. Long address
8. Relative address
9. Bit Inherent
10. Bit direct

Indexed A

In Indexed addressing either PC / DPTR is used to hold the base address, & A is used hold the offset address. Adding the value of Base address

to the value offset address from the effective address.
Indexed addressing is used with Jump / MOVC
lookup tables are easily implemented with the help of
Indexed addressing

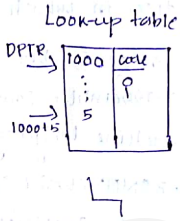
Indexed ——————)
address ← sum of
A+PC or
A + DPTR (only for ROM)

eg: MOVC A, @A+DPTR; A ← [A+DPTR]
MOVC A, @A+PC; A ← [A+PC]

* used for program memory

eg: Seven segment to display

g h e d c b a i/p code
0 0 0 0 1 1 0 01 06H



eq: mov DPTR, #1000
mov A, #05
mov A, @DPTR

Relative addressing

eg: SJMP loop1 (Short jump)
JC back (Jump carry)

Absolute addressing

AJMP loop1 (Absolute jump)
ACALL loop2 (Absolute call)

long addressing

LJMP loop1
LJMP loop2
Bit inherent

CLR C; Clear carry flag

Bit direct

CLR 07H
SETB 0H

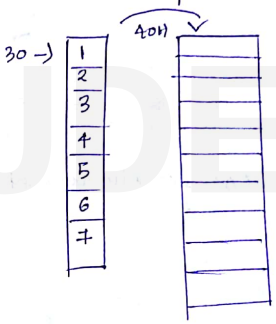
Assembly Language

mov A, R0
ADD A, R1
ADDC A, R2
mov B, R3
MUL A, B
MOV R4, A
MOV R5, B

$x = (a+b+c) * d$

Assume
$R_0 = a$
$R_1 = b$
$R_2 = c$
$R_3 = d$
R_4 & $R_5 = x$

Block transfer.



Q: Write an ALP add two number in location 4000H & 4001H and store the result at 4002H

A

Program written in Assembly language

			Executing steps
	ASM	ORG 3000	
	MOV	DPTR, #4000H	ES:
Indirect inter. RAM:	MOVX	A, @DPTR (16 bit)	M
	MOV	R1, A "	4000 : 5

```
MOV DPTR, # 4001H      4001: 3
MOVX A, @DPTR          ESC
ADD A, R1                 G 3000
MOV DPTR, #4002H       RESET → M
MOV @DPTR, A              4002
300E: JUMP 300E
```

```
LOOP:
mov R0, #30
mov R1, #40
mov A, @R;  A ← [R0]
mov @R1, A   [R1] ←
inc R0
inc R1
Jmp loop
```

Assuming PC to be 1000H transfer the code at 1000H to the internal at 70H

```
MOV A, #00
MOVC A, @A+PC
MOV 70H, A
```

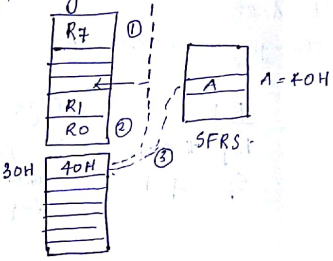
Instruction set in 8051

* Data transfer instructions
* Boolean variable manipulation instructions
* Arithmetic instructions.
* Program flow control instructions.

* Interrupt flow control instructions.
* logic instructions
* Data transfer instructions

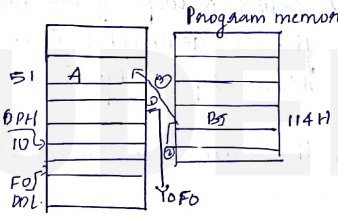
MOV
MOVX
MOVC

1) eg: MOV @R1, A

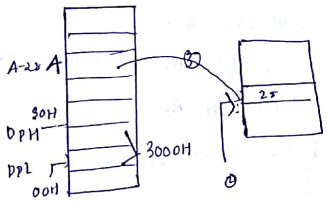


• Inte RAM - normal Special function regi.

2) MOVC A, @A + DPTR • code memory



3) MOVX @DPTR, A • External RAM memory



Exchange Swapping.

XCH A, @R
XCH A, Rn
XCH A, direct
XCHD A, @R
b) changing lower nipple only.

Push
SP → SPH
Push # data

Pop
Pop # data
SP ← SP - 1

Prgm flow instru.
- Loop, jump & call instr.
- Condition & unconditional jump instruct.

LOOP:

Repeating a sequence of instructions a certain number of times is called a loop eg: ~~DJNZ~~ DJNZ Reg, label
In this instruction the register is decremented if it is not 0. it jump to the target address offered by the label

eg: write a pgm a) clear the accumulator then add the value 3 ten times.

Pgm

```
mov A, #0 ;  A=0, Clear Acc
MOV R2, #10 ; Load counter R2=10
ADD A, #03 ; Add 03 to acc
DJNZ R2, Again ; Repeat until R2=0 (10 times)
MOV R5, A ; Save A in R5
```

* loop can be repeated maximum 256 times.

8051 Conditional Jump Instructions

JZ : jump if A=0 check accumulator is zero or not
JNZ; Jump if A≠0
DJNZ; decrement and jump if
CJNE A byte; jump if A ≠ byte (compare)
CJNE reg, #data;
JC; Jump if CY = 1 (carry)
JNC; Jump if CY = 0
JB Jump if bit = 1
JNB Jump if bit = 0
JBC jump if bit = 1 and clear bit

eg: progaam: write ALP to determine if R5 contains the value 0. if so put 55H in H

```
MOV A, R5; copy R5 to A
JNZ Next; jump if A is not zero.
MOV R5, # 55H
Next: ....
      ....
```

eg:2 Find the sum of the value 79H, F5, E2H
Put the sum in register R0 and R5

```
MOV A, #0
MOV R5, A
ADD A, #79H
JNC l1
INC R5
l1: ADD A, #0F5H
```

```
    JNC L2
    INC R5
L2: ADD A, #0E2H
    JNC over
    INC R5
```

Another control transfer instruction is the instruction call which is used to call subroutines are often used to perform task that need to be performed frequently

1. Long call (LCall)
2. Absolute call (ACALL)

LCALL:-

* range: call subroutine located anywhere within 64K
* 3 byte instruction (1 byte opcode, 2 byte address)

ABSOLUTE CALL

* range: within 2K byte
* 2- byte instruction (1 byte opcode, 1byte address).

Jump Instructions

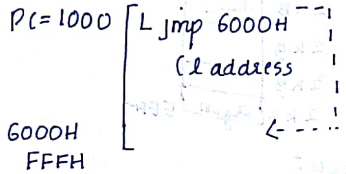
1. Long jump
2. Short jump
3. Absolute jump

1. Long jump.

Syntax: L jump address.

Range: full memory 64 KB (longest)

Pgm memory



1. Fetch the instruction in PC = 1000
2. PC ← L address thus jump.
3. fetch the instruction from L address.

PC ← 6000H
(L address)

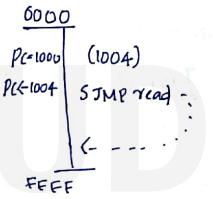
* 3 byte instruction

LJMP L address (3B)

2. Short jump.

Syntax: SJMP address (address → relative address).

Distance is very small



* PC ← PC + address
* 2-byte instruction
* Reel is calculated from next inst

eg: SJMP read (2B) eg: SJMP 04H (2B)

Range: -128 ..... +127.

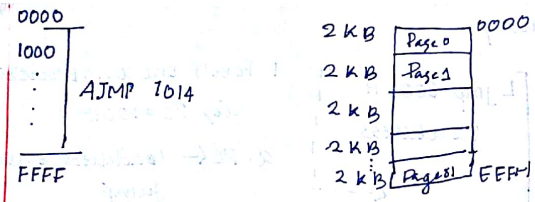
all conditional jump are short jumps

eg: JC, JNC, JZ, JNZ, JE, DJNZ, JB etc...

3. ABSOLUTE JUMP

- Increase the range
- Any jump in same page.
- Max range is 2KB.

Syntax: AJMP address.



Average of two numbers:-

Read proced
CX, 5
MOV n, CX
AX, Num 1
DEC CX
CMP CX, 00
Je 1500
acid sum, ax.
Jmp.
div bx, y

21/6/17 Programmable Interrupt Timer (8253/8254)

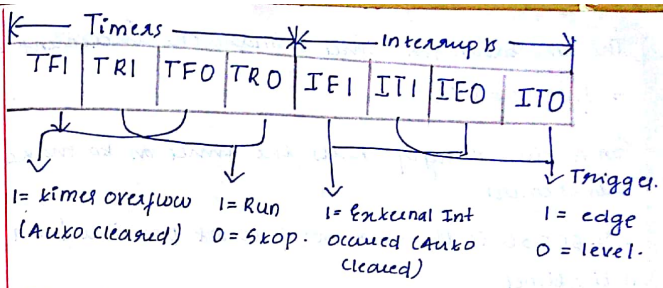
x Timer → To produce delay
→ Fixed frequency

x Counter → Variable frequency
→ to count

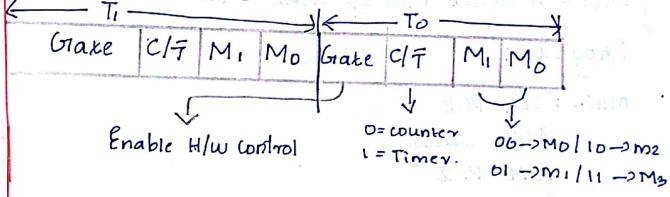
TIMERS

Two timers in 8051 → T_0 (16-bit) → TH0 (8-bit), TL0 (8-bit)
T_1 (16-bit) → TH1 (8-bit), TL1 (8-bit)

TCON (TIMER CONTROL) * 8-bit register (SFR)



TMOD (Timer modes



Count = maximum − need + 1
count

eg:
If timer act as counter ~~counter~~ to count currency

TCON - bit addressible, 8-bit, Special funct^n Register.
It consist of timer and ~~register~~ Interrupts.
If $TR_0 = 1$ timer starts else stop.

External Interrupt and internal interrupt will be
for. Up " a timer " " counter.

C/T = 0 ⇒ Timer
= 1 ⇒ counter.

DELAY PROGM

- Assume the processor is clocked by a 12 MHz crystal
ie, timer clock i/p will be 12 MH ÷ 12 ie 1 MHz.

The time taken for timer to make one increment is

$= \frac{1}{1}$ MHz = 1 μs

For a time delay of n μs the timer as to make n increments

$2^{16} = 65536$ is the max. no: of count possible for a 16 bit timer

THDL = Hexadecimal of max count − next count + 1

Program:

```
main : CLR  P1.2
       ACALL delay
       SETB  P1.2
       ACALL DELAY
       SJMP  main.

Delay: MOV TMOD, #01H   mode 1
       MOV TL0, #0CH  } count
       MOV TH0, #FEH  }
       MOV TCON, #10H  Mode - Mode1 Timer Start

Count: JNB TF0, wait [first overflow flag = 1, automatically 0]
       CLR TR0
       CLR TF0
       RET.
```

Delay count = 500

eg: 65536 − 500 + 1 = 65036 = $(FE0C)_H$.

Procedure

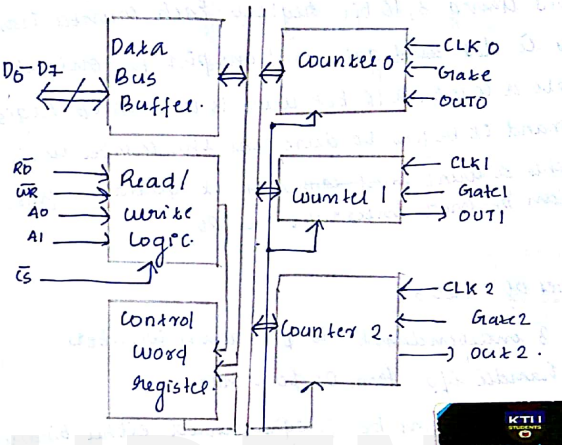
1: Set the mode.

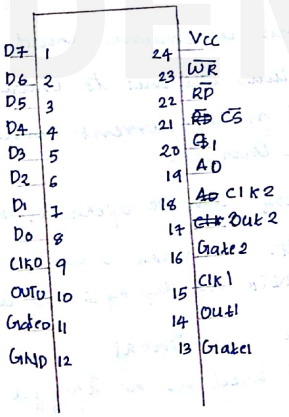
2: Give the count

3: Start the timer

8253/8254 Programmable Interval Timer.

Architecture:





A	A1	
0	0	Counter 0
0	1	Counter 1
1	1	Control word Reg.
1	0	Counter 2
X	X	No Selection

The INTEL 8253 and 8254 are programmable interval Timers designed for up to perform timing & counting functions using 3, 16 bit registers. Each counter has 2 i/p pin i.e clock and gate and one pin for output. To operate a counter a 16 bit count is load in its register on command it begins to decrement the counter until it reaches a count ~~that can~~ then it generate a pulse that can be used interrupt the CPU.

Features of 8253

It has 3 independant 16 bit down counters
It can handle i/p from DC to 10MHz.
These 3 counters can be programmed either binary/BCD. count. It is compatible with almost all up & µc
8254 has a powerful command called READ BACK command which allows the user to check the count value, programmed mode, current mode & the current status of the counter.

8253/8254 can be operated in 6 diff. mods.

mod 0: (Interrupt on terminal count) It is used to generate an INTR to up after a certain inter

mod 1: programmable one shot
The gate i/p is used as a trigger i/p in this mode.

Mod 2: Write generator
whenever count = 0 another low pulse is generated at the o/p and the counter will be reloaded

mod 3 Square wave gene

mod 4: Slw triggered mode (In this mode the o/p will remain high untill timer has counted to zero. at which point the o/p will pulse low and then go high.

mod 5: Hlw triggered mode this mode is similar to mod 4 except that counting is initiated by the gate i/p which means it is H/w triggered instead of Slw triggered.

Logical Operation Instruction Set in 8051

8051/mnuemonic Boolean operators

ANL ——→ AND (RESET)
ORL ——→ OR (SET)
XRL ——→ XOR (CLEAR)
CPL ——→ NOT

eg: ANL A, #n [AND each bit of A with the same bit of immediate number n, Put the result in A].

BYTE LEVEL LOGICAL.

ANL A, Rn
ORL A, #n
ORL A, address
ORL A, @DPTR
CLR A
CPL A
XRL A, #n.

Number Conversions - AND.

Bit level logical operations

ANL C, b AND C and the address bit put the result in C.

Bit level instruction

A	P_0	PSW
B	P_1	TCON
IE	P_2	SCON
IP	P_3	

Rotate Instruction.

RL A, Rotate the A register one bit position to left

RLC A
RR A (Right Rotation)
RRC A → (carry)
Swap A

Arithmetic Instruction.

ADD / ADDC dest, src
SUB dest, src.
MUL AB
DIV AB
DA A [Decimal adjust the A register]