Addressing Modes

Register Addressing – either source or destination is one of CPU register

```
MOV R0,A
MOV A,R7
ADD A,R4
ADD A,R7
MOV DPTR,#25F5H
MOV R5,DPL
MOV R,DPH
```

Note that MOV R4,R7 is incorrect

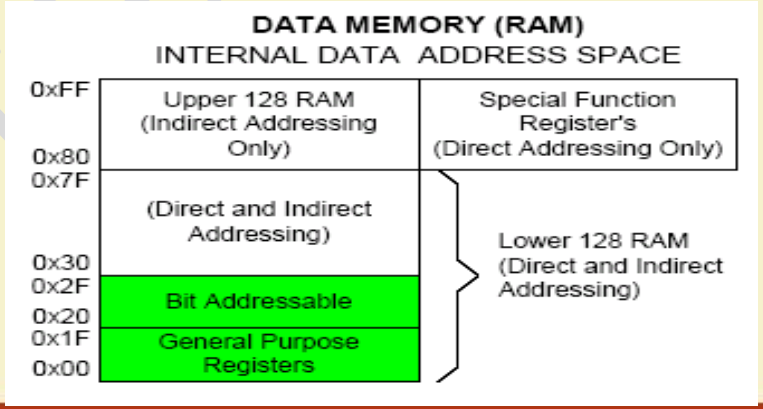
NPTEL

Addressing Modes

Direct Mode – specify data by its 8-bit address

Usually for 30h-7Fh of RAM

```
Mov a, 70h          ; copy contents of RAM at 70h to a
Mov R0,40h          ; copy contents of RAM at 40h to R0
Mov 56h,a           ; put contents of a at 56h
Mov 0D0h,a          ; put contents of a into PSW
```



NPTEL

Addressing Modes

Direct Mode – play with R0-R7 by direct address

```
MOV A,4   ≡   MOV A,R4

MOV A,7   ≡   MOV A,R7

MOV 7,2   ≡   MOV R7,R6

MOV R2,#5    ;Put 5 in R2
MOV R2,5     ;Put content of RAM at 5 in R2
```

NPTEL

Addressing Modes

Register Indirect – the address of the source or destination is specified in registers

Uses registers R0 or R1 for 8-bit address:

```
mov psw, #0              ; use register bank 0
mov r0, #0x3C
mov @r0, #3              ; memory at 3C gets #3
                         ; M[3C] ← 3
```

Uses DPTR register for 16-bit addresses:

```
mov dptr, #0x9000        ; dptr ← 9000h
movx a, @dptr            ; a ← M[9000]
```

Note that 9000 is an address in external memory

NPTEL

Addressing Modes

Register Indexed Mode – source or destination address is the sum of the base address and the accumulator(Index)

- Base address can be DPTR or PC

```
mov dptr, #4000h
mov a, #5
movc a, @a + dptr  ;a ← M[4005]
```

NPTEL

Addressing Modes

Register Indexed Mode continue

- Base address can be DPTR or PC

```
        ORG 1000h
        1000  mov a, #5
        1002  movc a, @a + PC    ;a ← M[1008]
PC  →   1003  Nop
```

- Table Lookup
- MOVC only can read internal code memory

NPTEL

Acc Register

- A register can be accessed by direct and register mode

- This 3 instruction has same function with different code

```
0703 E500        mov a,00h
0705 8500E0      mov acc,00h
0708 8500E0      mov 0e0h,00h
```

- Also this 3 instruction

```
070B E9          mov a,r1
070C 89E0        mov acc,r1
070E 89E0        mov 0e0h,r1
```

NPTEL

SFRs Address

- B – always direct mode - except in MUL & DIV

```
0703 8500F0          mov b,00h
0706 8500F0          mov 0f0h,00h

0709 8CF0            mov b,r4
070B 8CF0            mov 0f0h,r4
```

- P0~P3 – are direct address

```
0704 F580            mov p0,a
0706 F580            mov 80h,a
0708 859080          mov p0,p1
```

- Also other SFRs (pcon, tmod, psw,….)

NPTEL

SFRs Address

All SFRs such as

(ACC, B, PCON, TMOD, PSW, P0~P3, …)

are accessible by name and direct address

But

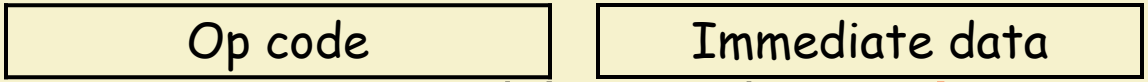
both of them

Must be coded as direct address

NPTEL

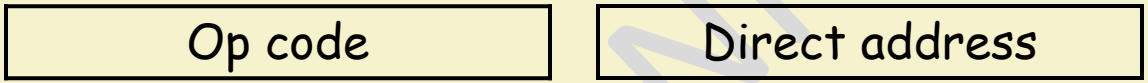
8051 Instruction Format

- immediate addressing




```
add a,#3dh      ;machine code=243d
```

- Direct addressing




```
mov r3,0E8h     ;machine code=ABE8
```

NPTEL

8051 Instruction Format

- Register addressing

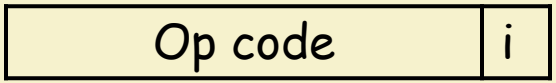
Op code	n	n	n

```
070D E8      mov a,r0      ;E8 = 1110 1000
070E E9      mov a,r1      ;E9 = 1110 1001
070F EA      mov a,r2      ;EA = 1110 1010
0710 ED      mov a,r5      ;ED = 1110 1101
0711 EF      mov a,r7      ;Ef = 1110 1111
0712 2F      add a,r7
0713 F8      mov r0,a
0714 F9      mov r1,a
0715 FA      mov r2,a
0716 FD      mov r5,a
0717 FD      mov r5,a
```

NPTEL

8051 Instruction Format

- Register indirect addressing



mov a, @Ri ; i = 0 or 1

```
070D E7      mov  a,@r1
070D 93      movc a,@a+dptr
070E 83      movc a,@a+pc
070F E0      movx a,@dptr
0710 F0      movx @dptr,a
0711 F2      movx @r0,a
0712 E3      movx a,@r1
```

NPTEL

8051 Instruction Format

- relative addressing

Op code	Relative address

```
here:  sjmp  here     ;machine code=80FE (FE=-2)
Range = (-128 ~ 127)
```

- Absolute addressing (limited in 2k current mem block)

A10-A8	Op code	A7-A0

```
0700                1       org 0700h
0700 E106           2       ajmp next    ;next=706h
0702 00             3       nop
0703 00             4       nop
0704 00             5       nop
0705 00             6       nop
                    7   next:
                    8       end
```

07FEh

8051 Instruction Format

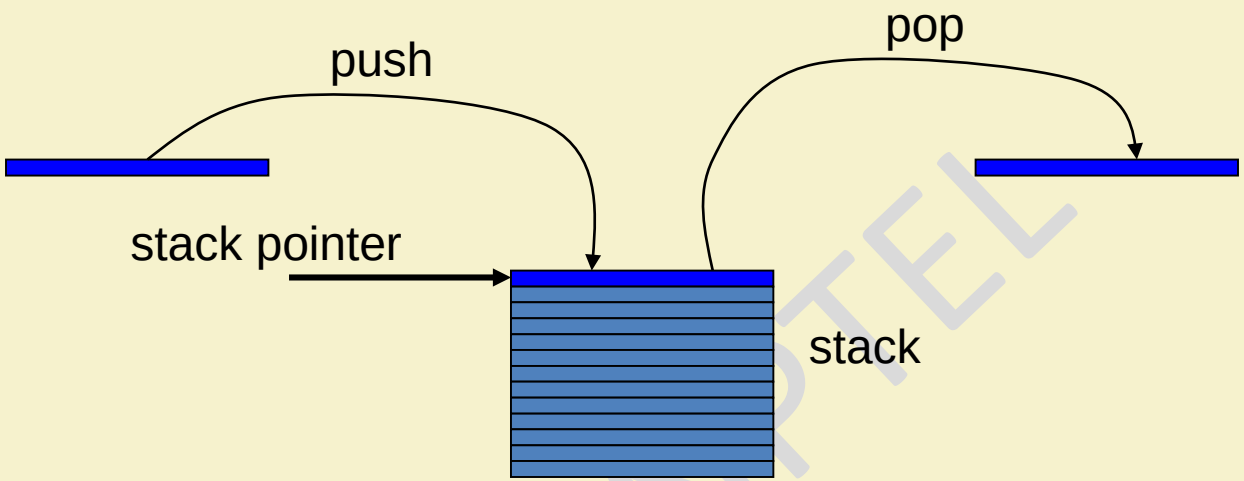
- Long distance address

Op code	A15-A8	A7-A0

Range = (0000h ~ FFFFh)

```
0700                1          org 0700h
0700 020707         2          ljmp next    ;next=0707h
0703 00             3          nop
0704 00             4          nop
0705 00             5          nop
0706 00             6          nop
                    7     next:
                    8          end
```

Stacks



NPTEL

Stack

- Stack-oriented data transfer
  - Only one operand (direct addressing)
  - SP is other operand – register indirect - implied
- Direct addressing mode must be used in Push and Pop

```
mov sp, #0x40   ; Initialize SP
push 0x55       ; SP ← SP+1, M[SP] ← M[55]
                ; M[41] ← M[55]
pop b           ; b ← M[55]
```

Note: can only specify RAM or SFRs (direct mode) to push or pop. Therefore, to push/pop the accumulator, must use acc, not a

NPTEL

Stack (push,pop)

- Therefore

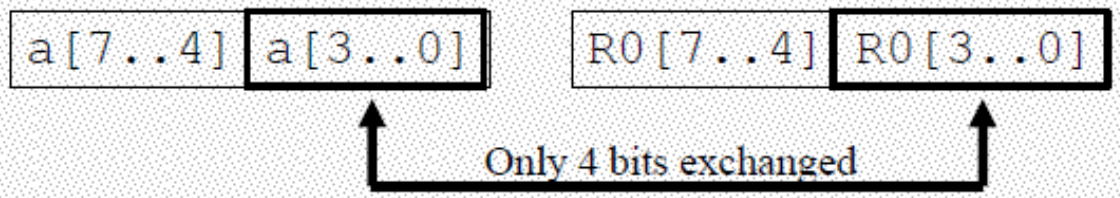
```
Push a     ;is invalid
Push r0    ;is invalid
Push r1    ;is invalid
push acc   ;is correct
Push psw   ;is correct
Push b     ;is correct
Push 13h
Push 0
Push 1
Pop  7
Pop  8
Push 0e0h   ;acc
Pop  0f0h   ;b
```

NPTEL

Exchange Instructions

two way data transfer

```
XCH a, 30h          ; a ↔ M[30]
XCH a, R0           ; a ↔ R0
XCH a, @R0          ; a ↔ M[R0]
XCHD a, R0          ; exchange "digit"
```

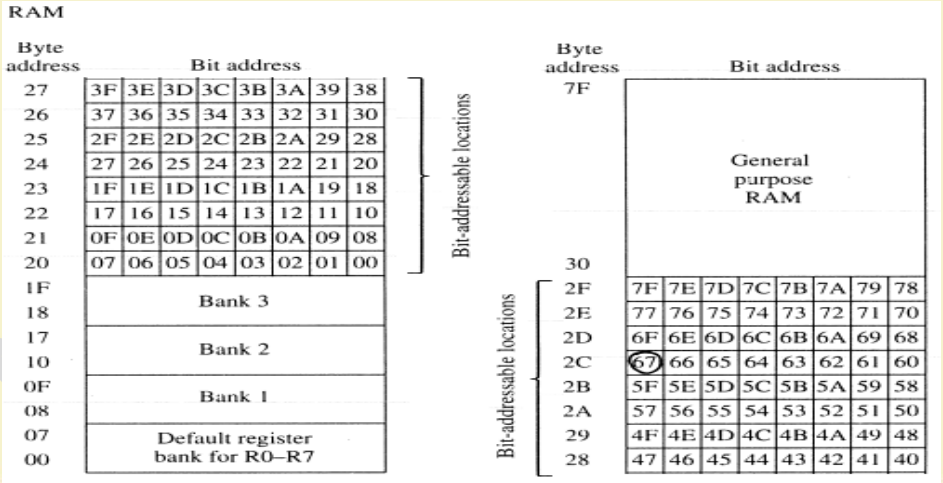


Bit-Oriented Data Transfer

- transfers between individual bits.
- Carry flag (C) (bit 7 in the PSW) is used as a single-bit accumulator
- RAM bits in addresses 20-2F are bit addressable

```
mov C, P0.0

mov C, 67h
mov C, 2ch.7
```



SFRs that are Bit Addressable

SFRs with addresses ending in 0 or 8 are bit-addressable.

(80, 88, 90, 98, etc)

Notice that all 4 parallel I/O ports are bit addressable.

Byte address	Bit address									Byte address	Bit address								
98	9F	9E	9D	9C	9B	9A	99	98	SCON	FF									
										F0	F7	F6	F5	F4	F3	F2	F1	F0	B
90	97	96	95	94	93	92	91	90	P1										
										E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
8D	not bit addressable								TH1										
8C	not bit addressable								TH0	D0	D7	D6	D5	D4	D3	D2	–	D0	PSW
8B	not bit addressable								TL1										
8A	not bit addressable								TL0	B8	–	–	–	BC	BB	BA	B9	B8	IP
89	not bit addressable								TMOD										
88	8F	8E	8D	8C	8B	8A	89	88	TCON	B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
87	not bit addressable								PCON										
										A8	AF	–	–	AC	AB	AA	A9	A8	IE
83	not bit addressable								DPH										
82	not bit addressable								DPL	A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
81	not bit addressable								SP										
80	87	86	85	84	83	82	81	80	P0	99	not bit addressable								SBUF

Data Processing Instructions

Arithmetic Instructions

Logic Instructions

NPTEL

Arithmetic Instructions

- Add
- Subtract
- Increment
- Decrement
- Multiply
- Divide
- Decimal adjust

NPTEL

Arithmetic Instructions

Mnemonic	Description
ADD A, byte	add A to byte, put result in A
ADDC A, byte	add with carry
SUBB A, byte	subtract with borrow
INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte
MUL AB	multiply accumulator by b register
DIV AB	divide accumulator by b register
DA A	decimal adjust the accumulator

NPTEL

ADD Instructions

```
add a, byte          ; a ← a + byte
addc a, byte               ; a ← a + byte + C
```

These instructions affect 3 bits in PSW:

C = 1 if result of add is greater than FF

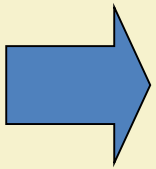
AC = 1 if there is a carry out of bit 3

OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.

Program Status Word (PSW)

Bit	7	6	5	4	3	2	1	0
Flag	CY	AC	F0	RS1	RS0	OV	F1	P
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflow flag	User Flag 1	Parity Bit

Instructions that Affect PSW bits



Instructions that Affect Flag Settings(1)

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

ADD Examples

```
mov a, #3Fh
add a, #D3h
```

```
  0011 1111
  1101 0011
  ---------
  0001 0010
```

- What is the value of the C, AC, OV flags after the second instruction is executed?

```
C = 1
AC = 1
OV = 0
```

NPTEL

Signed Addition and Overflow

2's complement:

0000 0000	00	0
…		
0111 1111	7F	127
1000 0000	80	-128
…		
1111 1111	FF	-1

```
 0111 1111  (positive 127)
 0111 0011  (positive 115)
 1111 0010  (overflow
cannot represent 242 in 8
bits 2's complement)

 1000 1111  (negative 113)
 1101 0011  (negative  45)
 0110 0010  (overflow)

0011 1111  (positive)
1101 0011  (negative)
0001 0010 (never overflows)
```

NPTEL

Addition Example

```
; Computes Z = X + Y
; Adds values at locations 78h and 79h and puts them in 7Ah
;-----------------------------------------------------------------
X               equ             78h
Y               equ             79h
Z               equ             7Ah
;-----------------------------------------------------------------
                org 00h
                ljmp Main
;-----------------------------------------------------------------
                org 100h
Main:
                mov a, X
                add a, Y
                mov Z, a
                end
```

NPTEL

The 16-bit ADD example

```
; Computes Z = X + Y     (X,Y,Z are 16 bit)
;-----------------------------------------------------------------
X           equ         78h
Y           equ         7Ah
Z           equ         7Ch
;-----------------------------------------------------------------
            org 00h
            ljmp Main
;-----------------------------------------------------------------
            org 100h
Main:
            mov a, X
            add a, Y
            mov Z, a
          mov a, X+1
            adc a, Y+1
            mov Z+1, a
            end
```

NPTEL

Subtract

SUBB A, byte	subtract with borrow

Example:

```
SUBB A, #0x4F     ;A ← A – 4F – C
```

Notice that
There is no subtraction WITHOUT borrow.
Therefore, if a subtraction without borrow is desired,
it is necessary to clear the C flag.

Example:

```
Clr  c
SUBB A, #0x4F     ;A ← A – 4F
```

NPTEL

Increment and Decrement

INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

NPTEL

Example: Increment 16-bit Word

- Assume 16-bit word in R3:R2

```
mov a, r2
add a, #1          ; use add rather than increment to affect C
mov r2, a
mov a, r3
addc a, #0         ; add C to most significant byte
mov r3, a
```

NPTEL

Multiply

When multiplying two 8-bit numbers, the size of the maximum product is 16-bits

FF x FF = FE01

(255 x 255 = 65025)

MUL AB ; BA ← A * B

Note : B gets the High byte
A gets the Low byte

NPTEL

Division

- Integer Division

```
DIV AB      ; divide A by B

A ← Quotient(A/B)
B ← Remainder(A/B)
```

OV - used to indicate a divide by zero condition.
C – set to zero

NPTEL

Decimal Adjust

```
DA a            ; decimal adjust a
```

Used to facilitate BCD addition.
Adds “6” to either high or low nibble after an addition to create a valid BCD number.

Example:

```
mov a, #23h
mov b, #29h
add a, b        ; a ← 23h + 29h = 4Ch (wanted 52)
DA a            ; a ← a + 6 = 52
```

NPTEL

Logic Instructions

- Bitwise logic operations
  - (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do NOT affect the flags in PSW

NPTEL

Bitwise Logic

ANL → AND

ORL → OR

XRL → XOR

CPL → Complement

Examples:

```
        00001111
ANL     10101100
        00001100

        00001111
ORL     10101100
        10101111

        00001111
XRL     10101100
        10100011

CPL     10101100
        01010011
```

NPTEL

Address Modes with Logic

ANL – AND

ORL – OR

XRL – eXclusive oR

a, byte

direct, reg. indirect, reg, immediate

byte, a

direct

byte, #constant

CPL – Complement

a ex: cpl a

NPTEL

Uses of Logic Instructions

- Force individual bits low, without affecting other bits.

```
anl PSW, #0xE7          ;PSW AND 11100111
```

- Force individual bits high.

```
orl PSW, #0x18          ;PSW OR 00011000
```

- Complement individual bits

```
xrl P1, #0x40                   ;P1 XRL 01000000
```

NPTEL

Other Logic Instructions

CLR - clear

RL – rotate left

RLC – rotate left through Carry

RR – rotate right

RRC – rotate right through Carry

SWAP – swap accumulator nibbles

NPTEL

CLR (Set all bits to 0)

CLR A

CLR byte (direct mode)

CLR Ri (register mode)

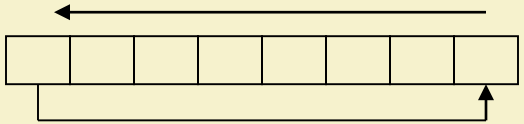
CLR @Ri (register indirect mode)

NPTEL

Rotate

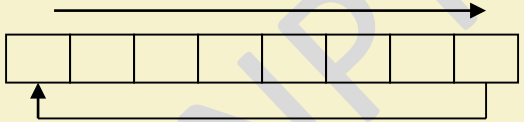
- Rotate instructions operate only on **a**

RL a



```
Mov a,#0xF0      ; a← 11110000
RR a             ; a← 11100001
```

RR a

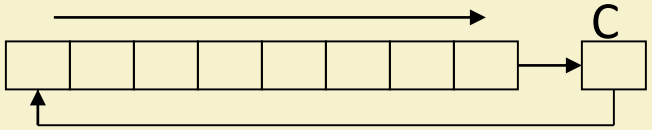


```
Mov a,#0xF0      ; a← 11110000
RR a             ; a← 01111000
```

NPTEL

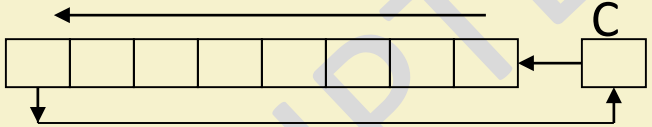
Rotate through Carry

RRC a




```
mov a, #0A9h        ; a ← A9
add a, #14h         ; a ← BD (10111101), C←0
rrc a               ; a ←  01011110, C←1
```

RLC a




```
mov a, #3ch         ; a ← 3ch(00111100)
setb c              ; c ← 1
rlc a               ; a ←  01111001, C←1
```

NPTEL

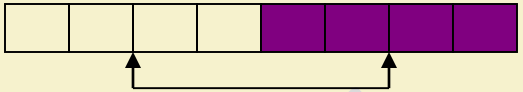
Rotate and Multiplication/Division

- Note that a shift left is the same as multiplying by 2, shift right is divide by 2

```
mov a, #3    ; A← 00000011 (3)
clr C        ; C← 0
rlc a        ; A← 00000110 (6)
rlc a        ; A← 00001100 (12)
rrc a        ; A← 00000110 (6)
```

Swap

SWAP a



```
mov a, #72h      ; a ← 72h
swap a           ; a ← 27h
```

NPTEL

Bit Logic Operations

- Some logic operations can be used with single bit operands

```
ANL C, bit
ORL C, bit
CLR C
CLR bit
CPL C
CPL bit
SETB C
SETB bit
```

- “bit” can be any of the bit-addressable RAM locations or SFRs.

NPTEL

Program Flow Control

- Unconditional jumps (“go to”)
- Conditional jumps
- Call and return

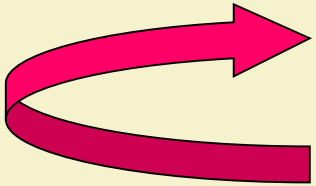
NPTEL

Unconditional Jumps

- **SJMP <rel addr>** ; Short jump, relative address is 8-bit 2's complement number, so jump can be up to 127 locations forward, or 128 locations back.
- **LJMP <address 16>** ; Long jump
- **AJMP <address 11>** ; Absolute jump to anywhere within 2K block of program memory
- **JMP @A + DPTR** ; Long indexed jump

NPTEL

Infinite Loops



```
Start: mov C, p3.7
       mov p1.6, C
       sjmp Start
```

Microcontroller application programs are almost always infinite loops!

NPTEL

Re-locatable Code

Memory specific NOT Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        ljmp Start
        end
```

Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        sjmp Start
        end
```

NPTEL

Jump table

```
                Mov dptr,#jump_table
                Mov a,#index_number
                Rl  a
                Jmp @a+dptr
                    ...
Jump_table:     ajmp case0
                ajmp case1
                ajmp case2
                ajmp case3
```

NPTEL

Conditional Jump

- These instructions cause a jump to occur only if a condition is true. Otherwise, program execution continues with the next instruction.

```
loop: mov a, P1
      jz  loop      ; if a=0, goto loop,
                    ; else goto next instruction
      mov b, a
```

- There is no zero flag (z)
- Content of A checked for zero on time

NPTEL

Conditional jumps

Mnemonic	Description
JZ <rel addr>	Jump if a = 0
JNZ <rel addr>	Jump if a != 0
JC <rel addr>	Jump if C = 1
JNC <rel addr>	Jump if C != 1
JB <bit>, <rel addr>	Jump if bit = 1
JNB <bit>,<rel addr>	Jump if bit != 1
JBC <bir>, <rel addr>	Jump if bit =1, &clear bit
CJNE A, direct, <rel addr>	Compare A and memory, jump if not equal

NPTEL

Example: Conditional Jumps

```
if (a = 0) is true
    send a 0 to LED
else
    send a 1 to LED
```

```
            jz led_off
            Setb P1.6
            sjmp skipover
 led_off:   clr P1.6
            mov A, P0
skipover:
```

NPTEL

More Conditional Jumps

Mnemonic	Description
CJNE A, #data <rel addr>	Compare A and data, jump if not equal
CJNE Rn, #data <rel addr>	Compare Rn and data, jump if not equal
CJNE @Rn, #data <rel addr>	Compare Rn and memory, jump if not equal
DJNZ Rn, <rel addr>	Decrement Rn and then jump if not zero
DJNZ direct, <rel addr>	Decrement memory and then jump if not zero

NPTEL

Iterative Loops

For A = 0 to 4 do

{…}

```
      clr a
loop: ...
      ...
      inc a
      cjne a, #4, loop
```

For A = 4 to 0 do

{…}

```
      mov R0, #4
loop: ...
      ...
      djnz R0, loop
```

NPTEL

Call and Return

- Call is similar to a jump, but
  - Call pushes PC on stack before branching

```
acall <address ll>        ; stack ← PC
                            ; PC ← address 11 bit

lcall <address 16>        ; stack ← PC
                            ; PC ← address 16 bit
```

NPTEL

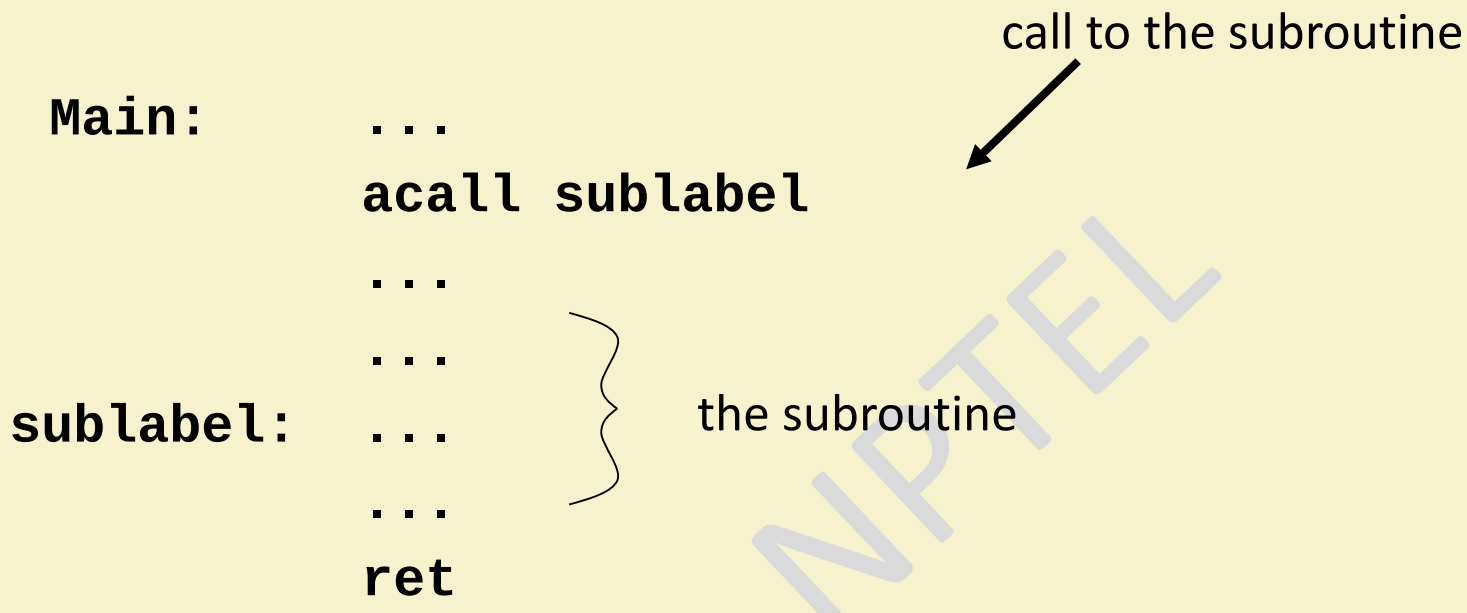
Return

- Return is also similar to a jump, but
  - Return instruction pops PC from stack to get address to jump to

```
ret                    ; PC ← stack
```

NPTEL

Subroutines



NPTEL

Initializing Stack Pointer

- SP is initialized to 07 after reset.(Same address as R7)
- With each push operation 1st , pc is increased
- When using subroutines, the stack will be used to store the PC, so it is very important to initialize the stack pointer. Location 2Fh is often used.

```
mov SP, #2Fh
```

NPTEL

Subroutine - Example

```
square:  push b
         mov  b,a
         mul  ab
         pop  b
         ret
```

- 8 byte and 11 machine cycle

```
square: inc a
         movc a,@a+pc
         ret
table:  db 0,1,4,9,16,25,36,49,64,81
```

- 13 byte and 5 machine cycle

NPTEL

Subroutine – another example

```
; Program to compute square root of value on Port 3
; (bits 3-0) and output on Port 1.
        org 0
        ljmp Main

Main:   mov P3, #0xFF    ; Port 3 is an input
loop:   mov a, P3
        anl a, #0x0F     ; Clear bits 7..4 of A
        lcall sqrt
        mov P1, a
        sjmp loop

sqrt:   inc a
        movc a, @a + PC
        ret

Sqrs:   db  0,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3
        end
```

reset service

main program

subroutine

data

Why Subroutines?

- Subroutines allow us to have "structured" assembly language programs.
- This is useful for breaking a large design into manageable parts.
- It saves code space when subroutines can be called many times in the same program.

NPTEL

example of delay

```
      mov a,#0aah
Back1:mov p0,a
      lcall delay1
      cpl a
      sjmp back1
Delay1:mov r0,#0ffh;1cycle
Here: djnz r0,here ;2cycle
      ret          ;2cycle
      end
```

Delay=1+255*2+2=513 cycle

```
Delay2:
      mov r6,#0ffh
back1: mov r7,#0ffh ;1cycle
Here:  djnz r7,here ;2cycle
       djnz r6,back1;2cycle
       ret         ;2cycle
       end
```

Delay=1+(1+255*2+2)*255+2
=130818 machine cycle

NPTEL

Long delay Example

```
GREEN_LED:      equ   P1.6
                org   ooh
                ljmp  Main

                org   100h
Main:           clr   GREEN_LED
Again:          acall Delay
                cpl   GREEN_LED
                sjmp  Again


Delay:          mov   R7, #02
Loop1:          mov   R6, #00h
Loop0:          mov   R5, #00h
                djnz  R5, $
                djnz  R6, Loop0
                djnz  R7, Loop1
                ret
                END
```

reset service

main program

subroutine

Example

```
; Move string from code memory to RAM
                        org 0
                        mov dptr,#string
                        mov r0,#10h
loop1:                  clr a
                        movc a,@a+dptr
                        jz stop
                        mov @r0,a
                        inc dptr
                        inc r0
                        sjmp loop1
stop:                   sjmp stop
; on-chip code memory used for string
                        org 18h
String:     db 'this is a string',0
                        end
```

NPTEL

Example

```
; p0:input   p1:output
            mov a,#0ffh
            mov p0,a
back:       mov a,p0
            mov p1,a
            sjmp back
```

Example

```
; duty cycle 50%
back:          cpl p1.2
               acall delay
               sjmp back
```

```
back:          setb p1.2
               acall delay
               Clr p1.2
               acall delay
               sjmp back
```

NPTEL

Example

```
; duty cycle 66%
back:    setb p1.2
         acall delay
         acall delay
         Clr p1.2
         acall delay
         sjmp back
```

NPTEL

8051
timer/counter

NPTEL

Timers /Counters Programming

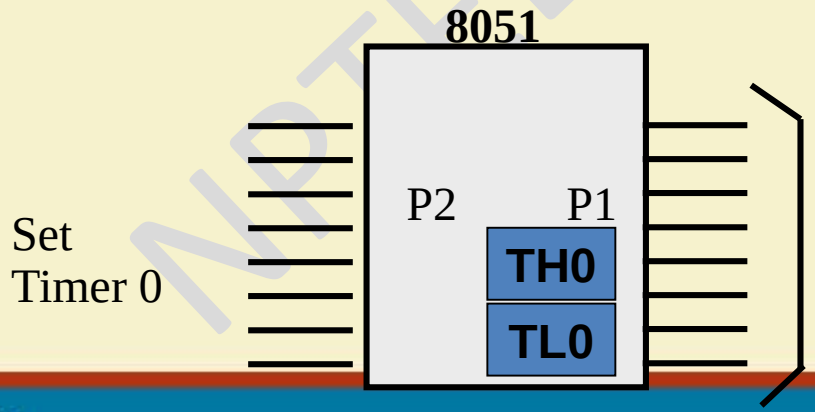
- The 8051 has 2 timers/counters: timer/counter 0 and timer/counter 1. They can be used as

1. The **timer** is used as a time delay generator.
   - The clock source is the **internal** crystal frequency of the 8051.
2. An event **counter**.
   - **External input** from input pin to count the number of events on registers.
   - These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

NPTEL

Timer

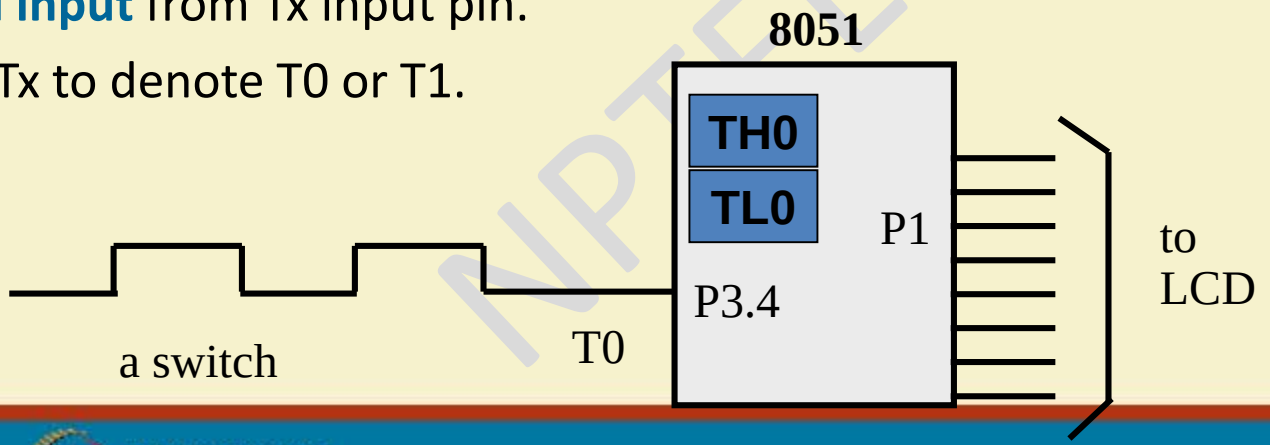
- Set the initial value of registers
- Start the timer and then the 8051 counts up.
- Input from internal system clock (machine cycle)
- When the registers equal to 0 and the 8051 sets a bit to denote time out



NPTEL

Counter

- Count the number of events
  - Show the number of events on registers
  - External input from T0 input pin (P3.4) for Counter 0
  - External input from T1 input pin (P3.5) for Counter 1
  - **External input** from Tx input pin.
  - We use Tx to denote T0 or T1.



NPTEL

Registers Used in Timer/Counter

- TH0, TL0, TH1, TL1
- TMOD (Timer mode register)
- TCON (Timer control register)
- Since 8052 has 3 timers/counters, the formats of these control registers are different.
  - T2CON (Timer 2 control register), TH2 and TL2 used for 8052 only.

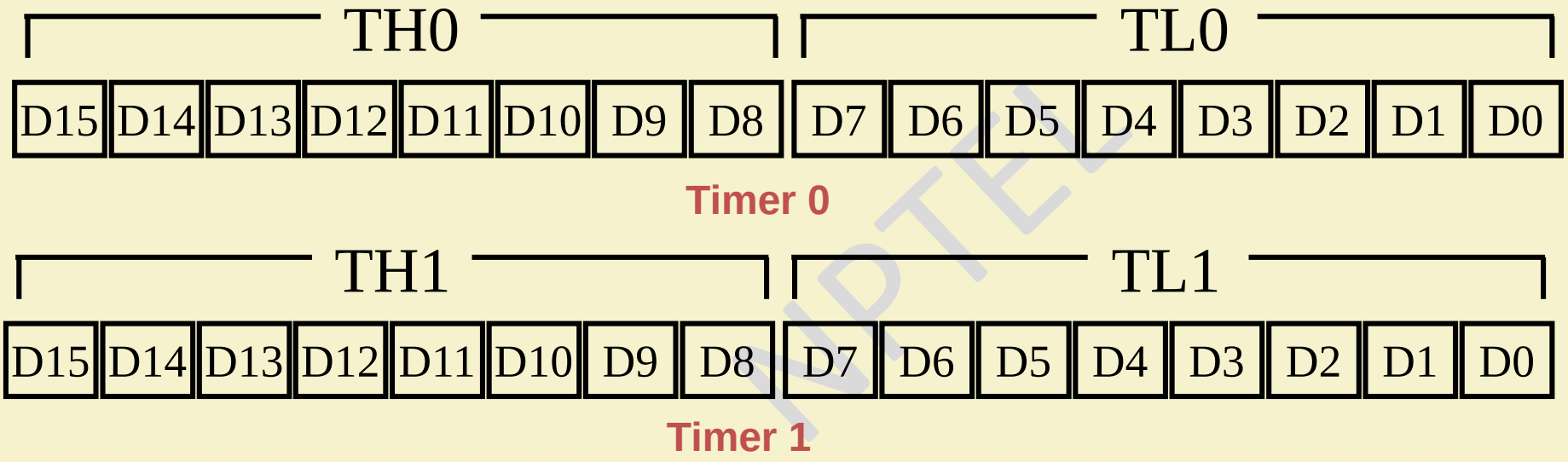
NPTEL

Basic Registers of the Timer

- Both timer 0 and timer 1 are 16 bits wide.
  - These registers stores
    - the time delay as a timer
    - the number of events as a counter
  - Timer 0: **TH0** & **TL0**
    - Timer 0 high byte, timer 0 low byte
  - Timer 1: **TH1** & **TL1**
    - Timer 1 high byte, timer 1 low byte
  - Each 16-bit timer can be accessed as two separate registers of low byte and high byte.

NPTEL

Timer Registers



NPTEL

TMOD Register

- Timer mode register: **TMOD**

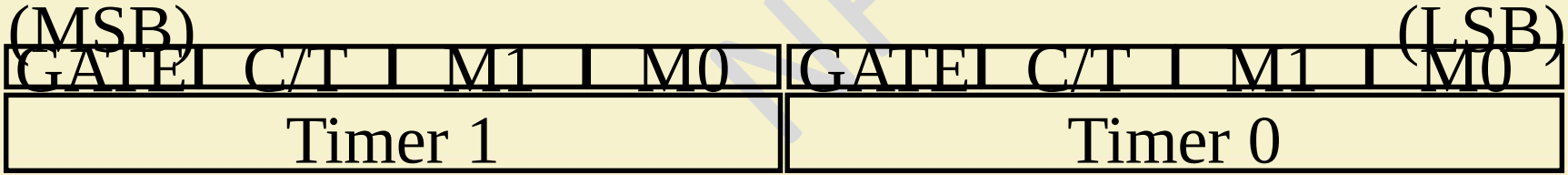
  MOV TMOD,#21H

  – An 8-bit register

  – Set the usage mode for two timers

    - Set lower 4 bits for Timer 0 (Set to 0000 if not used)
    - Set upper 4 bits for Timer 1 (Set to 0000 if not used)

  – Not bit-addressable



NPTEL

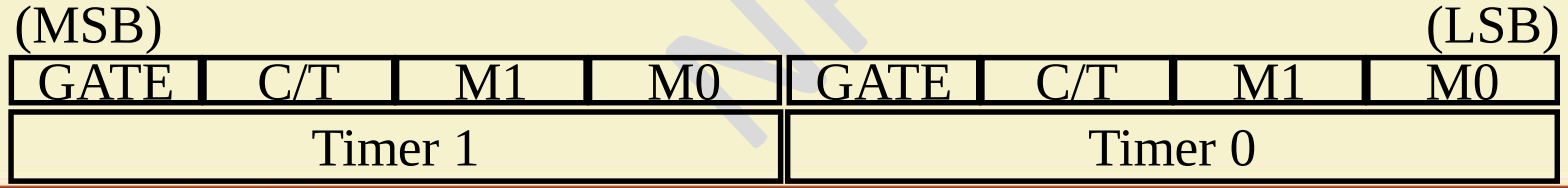
TMOD Register

GATE Gating control when set. Timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0



NPTEL

C/T (Clock/Timer)

- This bit is used to decide whether the timer is used as a delay generator or an event counter.
- C/T = 0 : timer
- C/T = 1 : counter

NPTEL

Gate

- Every timer has a mean of starting and stopping.
  - GATE=0
    - **Internal** control
    - The start and stop of the timer are controlled by way of **software.**
    - Set/clear the TR for start/stop timer.
  - GATE=1
    - **External** control
    - The hardware way of starting and stopping the timer by **software** and **an external source**.
    - Timer/counter is enabled only while the INT pin is high and the TR control pin is set (TR).

NPTEL

M1, M0

- M0 and M1 select the timer mode for timers 0 & 1.

M1	M0	Mode	Operating Mode
0	0	0	**13-bit timer** mode 8-bit THx + 5-bit TLx (x= 0 or 1)
0	1	1	**16-bit timer** mode 8-bit THx + 8-bit TLx
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

NPTEL