CHANDIGARH
ENGINEERING COLLEGE
CGC, LANDRAN, MOHALI
Buiding Careers. Transforming lives.

Microcontroller 8051

8051
timer/counter

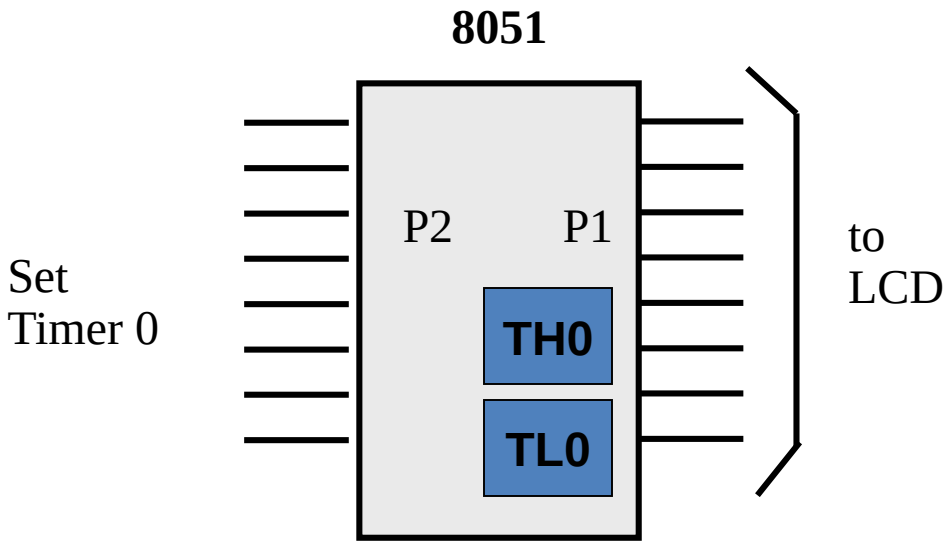
Timers /Counters Programming

- 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 cold 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.

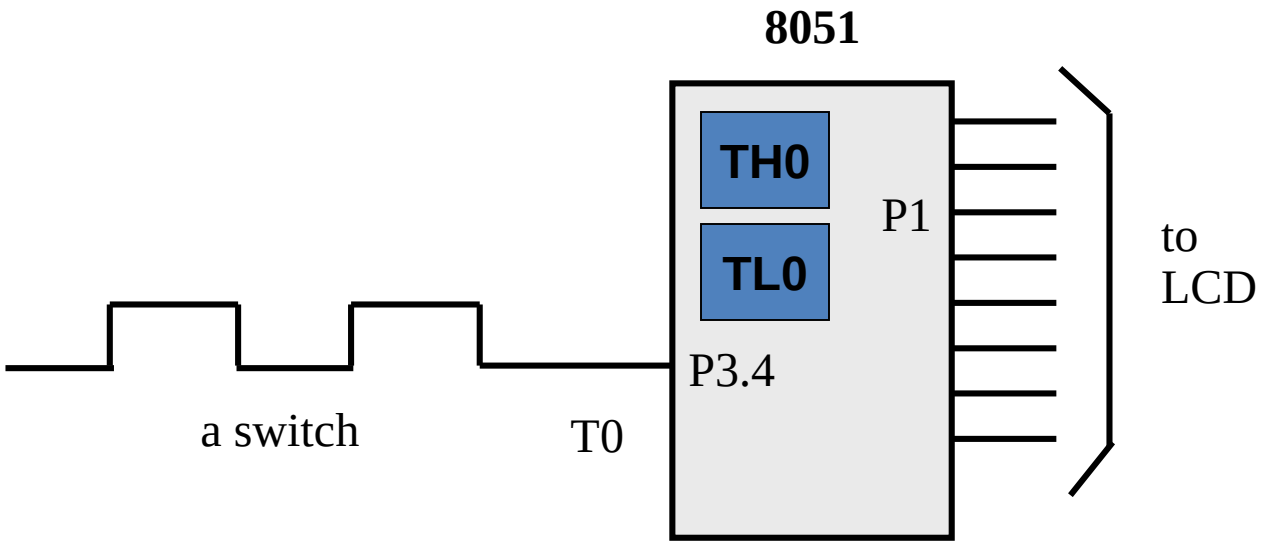
Timer

- 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



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.



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.

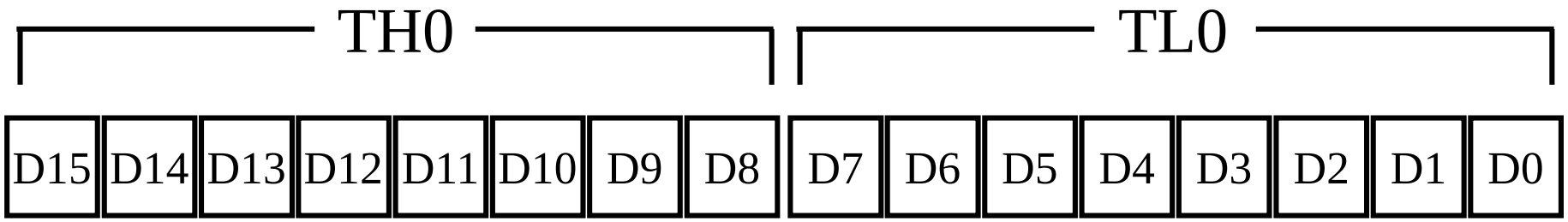
Basic Registers of the Timer

- 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.

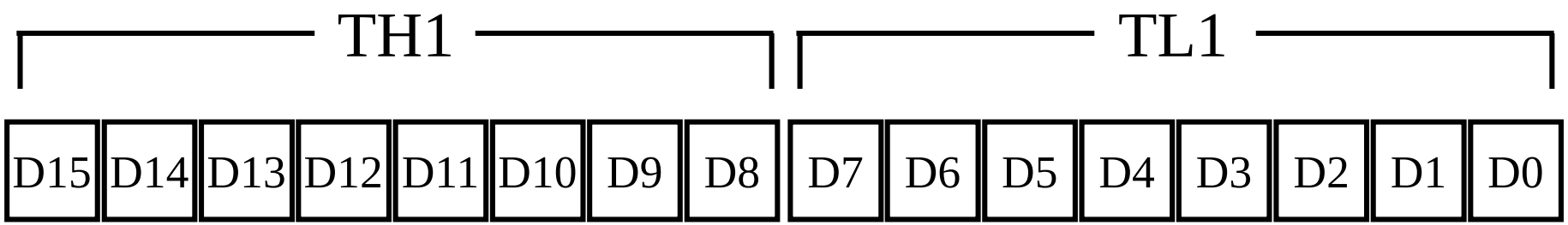
Basic Registers of the Timer

- 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.

Timer Registers



Timer 0



Timer 1

TMOD Register

- Timer mode register: **TMOD**

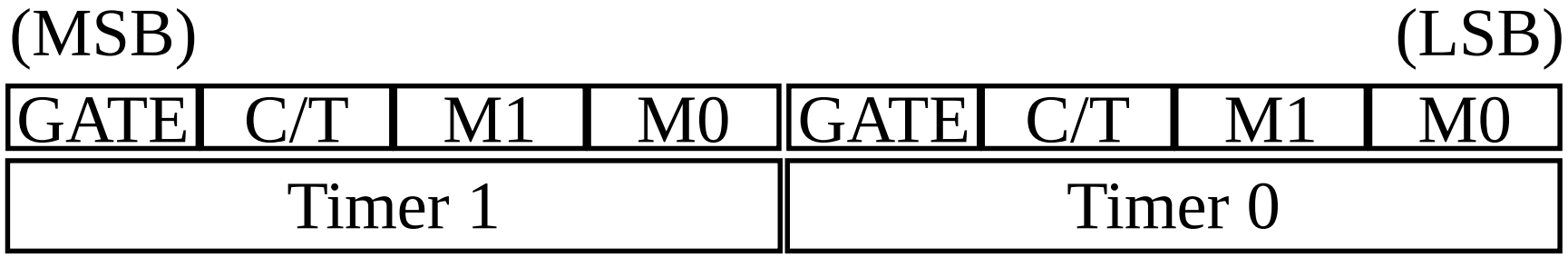
  `MOV TMOD,#21H`

  – 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



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

Gate

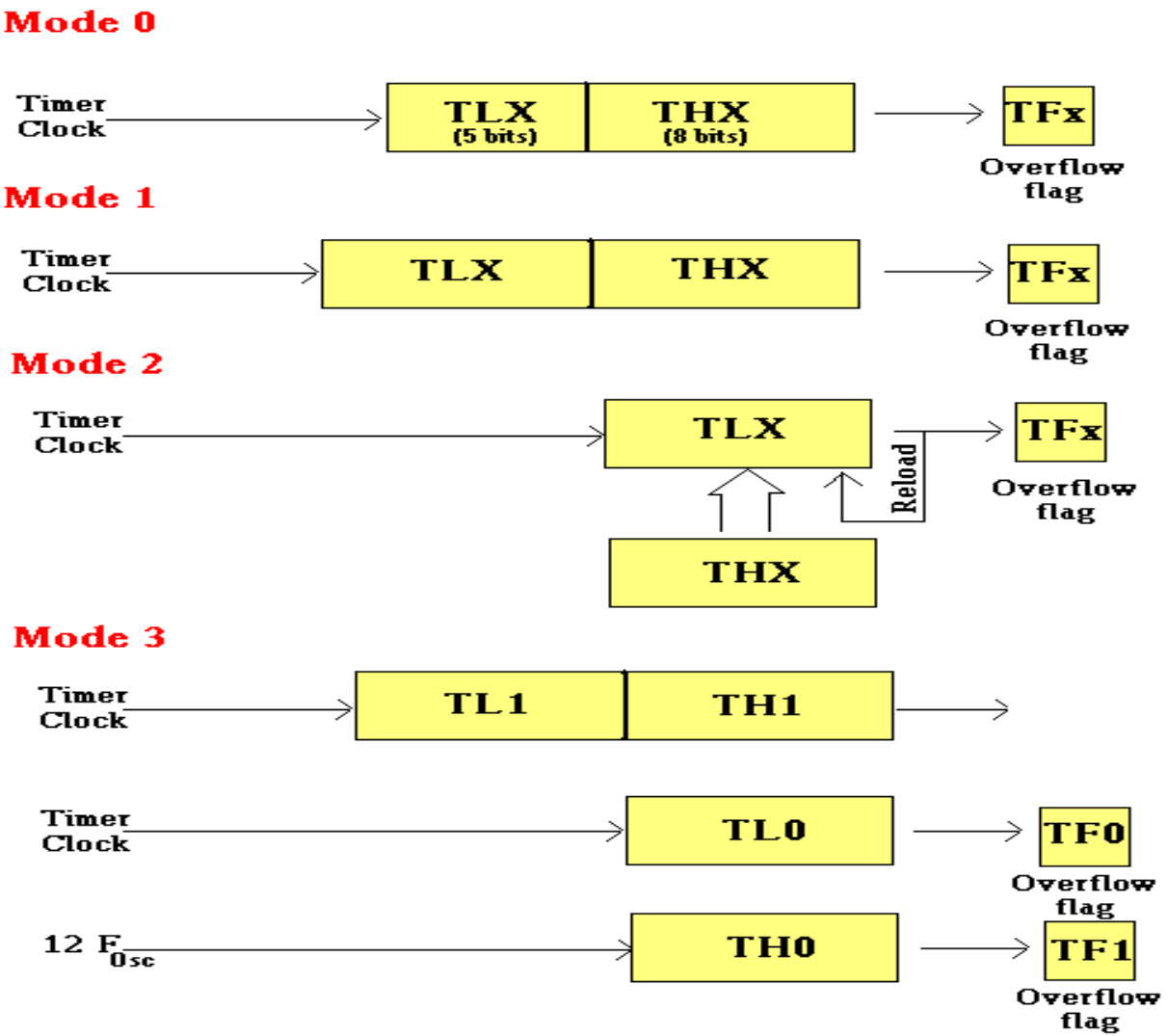
- Every timer has a mean of starting and stopping.
  - GATE=0
    - **Internal** control
    - The start and stop of the timer are controlled by way of **software.**
    - Set/clear the TR for start/stop timer.
  - GATE=1
    - **External** control
    - The hardware way of starting and stopping the 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).

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

Timer modes


Mode 0
Timer Clock
TLX (5 bits)
THX (8 bits)
TFx
Overflow flag
Mode 1
Timer Clock
TLX
THX
TFx
Overflow flag
Mode 2
Timer Clock
TLX
Reload
TFx
Overflow flag
THX
Mode 3
Timer Clock
TL1
TH1
Timer Clock
TL0
TF0
Overflow flag
12 F Osc
TH0
TF1
Overflow flag

TCON Register (1/2)

- Timer control register: **TMOD**
  - Upper nibble for timer/counter, lower nibble for interrupts
- **TR** (run control bit)
  - TR0 for Timer/counter 0; TR1 for Timer/counter 1.
  - TR is set by programmer to turn timer/counter on/off.
    - TR=0: off (stop)
    - TR=1: on (start)

Timer Mode 1

- In following, we all use timer 0 as an example.
- **16-bit** timer (TH0 and TL0)
- TH0-TL0 is incremented continuously when TR0 is set to 1. And the 8051 stops to increment TH0-TL0 when TR0 is cleared.
- The timer works with the internal system clock. In other words, the timer counts up each machine cycle.
- When the timer (TH0-TL0) reaches its maximum of FFFFH, it rolls over to 0000, and TF0 is raised.
- Programmer should check TF0 and stop the timer 0.

Data Processing Instructions

Arithmetic Instructions

Logic Instructions

Arithmetic Instructions

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

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

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

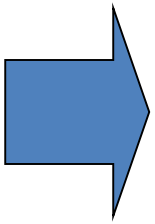
AC = 1 if there is a carry out of bit 3

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
```

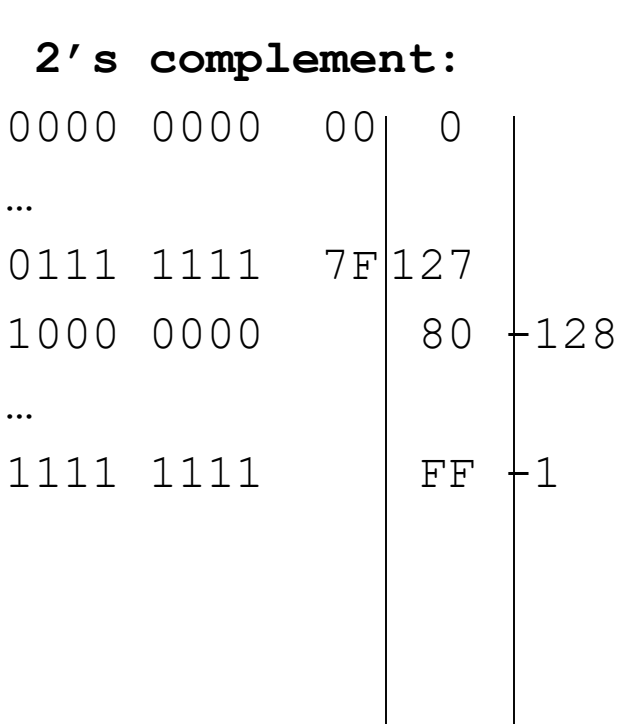
- What is the value of the C, AC, OV flags after the second instruction is executed?

C = 1

AC = 1

OV = 0

Signed Addition and Overflow




```
 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)
```

Data Processing Instructions

Arithmetic Instructions

Logic Instructions

Arithmetic Instructions

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

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

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

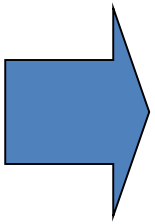
AC = 1 if there is a carry out of bit 3

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
```

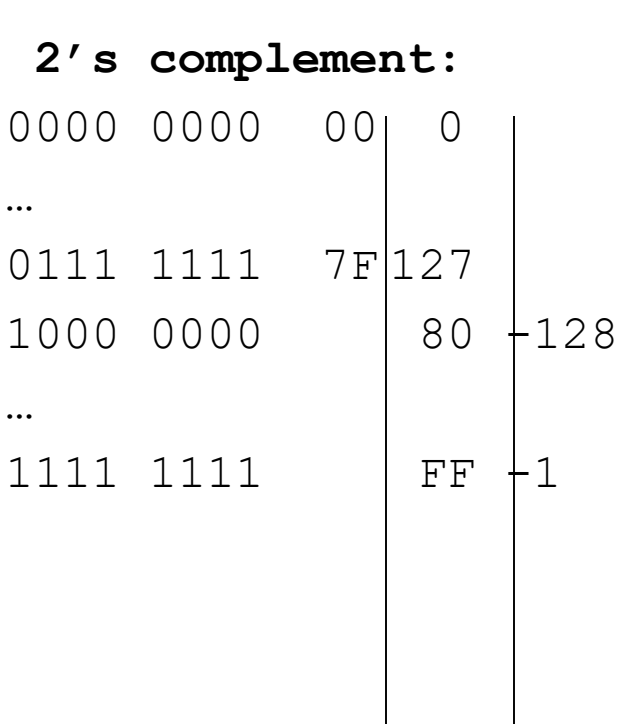
- What is the value of the C, AC, OV flags after the second instruction is executed?

C = 1

AC = 1

OV = 0

Signed Addition and Overflow




```
 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)
```

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
```

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
```

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.

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
```

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

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

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
```

Logic Instructions

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

Logic instructions do NOT affect the flags in PSW

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
```

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

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
```

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

CLR (Set all bits to 0)

CLR A

CLR byte (direct mode)

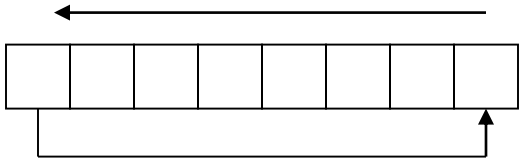
CLR Ri (register mode)

CLR @Ri (register indirect mode)

Rotate

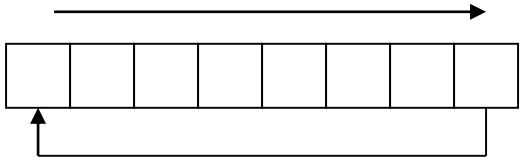
- Rotate instructions operate only on **a**

RL a



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

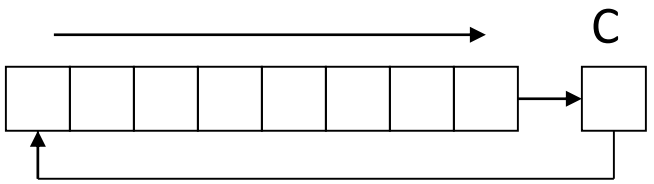
RR a



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

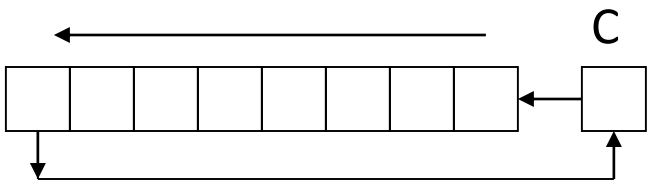
Rotate through Carry

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
```

Swap

SWAP a

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

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.

Shift/Mutliply Example

- Program segment to multiply by 2 and add 1.

Program Flow Control

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

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

Program Flow Control

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

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

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:
```

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

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
```

Iterative Loops(examples)

```
       mov a,#50h
       mov b,#00h
       cjne a,#50h,next
       mov  b,#01h
next:  nop
       end
```

```
       mov a,#25h
       mov r0,#10h
       mov r2,#5
Again: mov @ro,a
       inc r0
       djnz r2,again
       end
```

```
       mov a,#0aah
       mov b,#10h
Back1:mov r6,#50
Back2:cpl a
       djnz r6,back2
       djnz b,back1
       end
```

```
       mov a,#0h
       mov r4,#12h
Back:  add a,#05
       djnz r4,back
       mov  r5,a
       end
```