Unit 4

The 8051 Microcontrollers

"Microcontroller – a microprocessa along with the memory subsystem.

It has : (i) a processor to run programs

(ii) ROM to store the programs

(iii) RAM to store the data

(iv) I/O ports to handle I/O devices

(v) Timer section

8051 & Salient Features

→ 8051 is an 8 bit microcontroller, it has an 8 bit ALU. All arithmetic and logical operations are of 8 bits

→ 8051 has an 8 bit data bus

→ It has an internal ROM of 4KB used for storing programs

→ It has an internal RAM of 128 bytes for storing data.

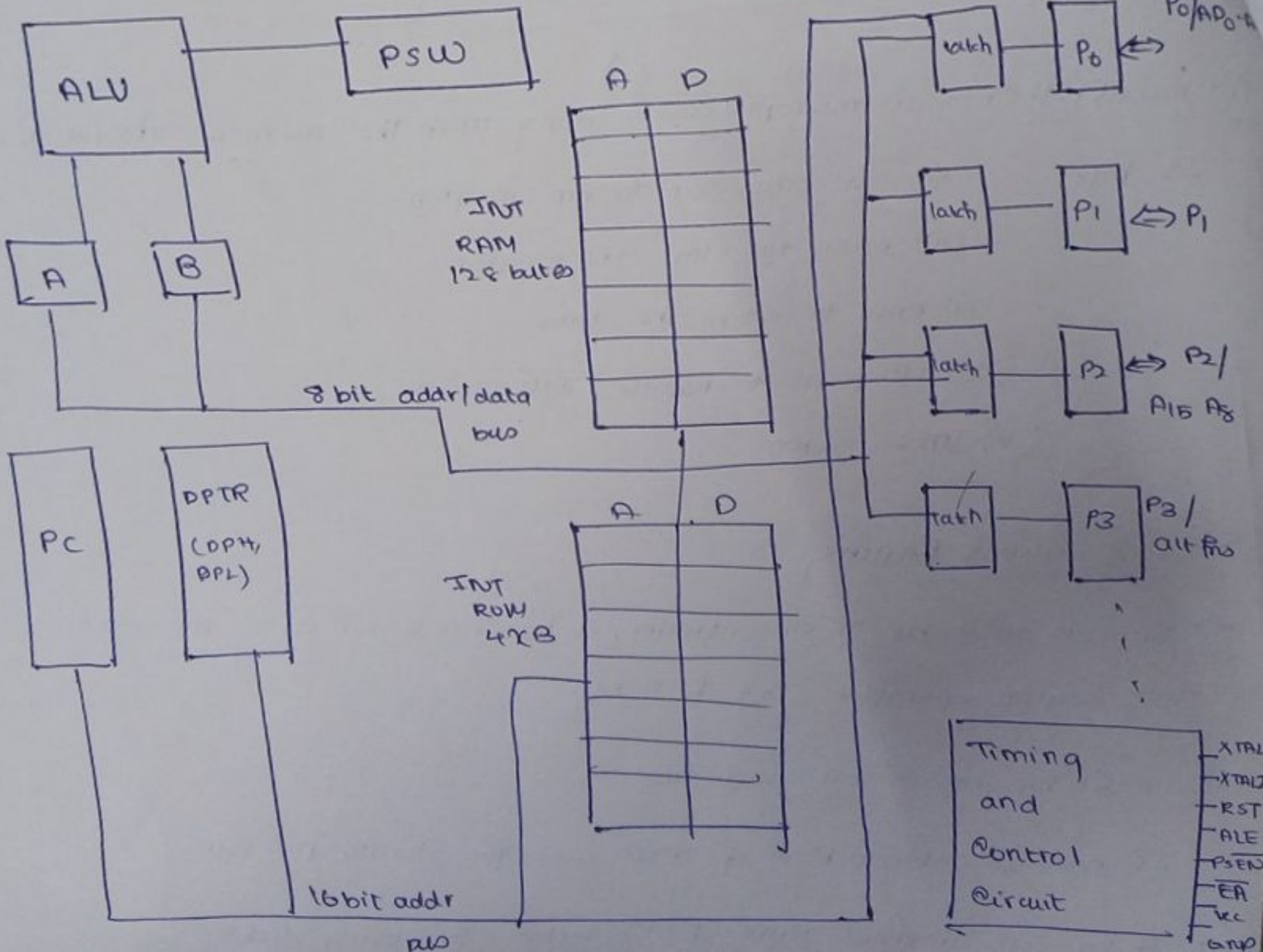
→ There are four, 8 bit, bidirectional I/O ports for interfacing external devices like keyboards, display, etc.

→ It has a serial port for long distance communication

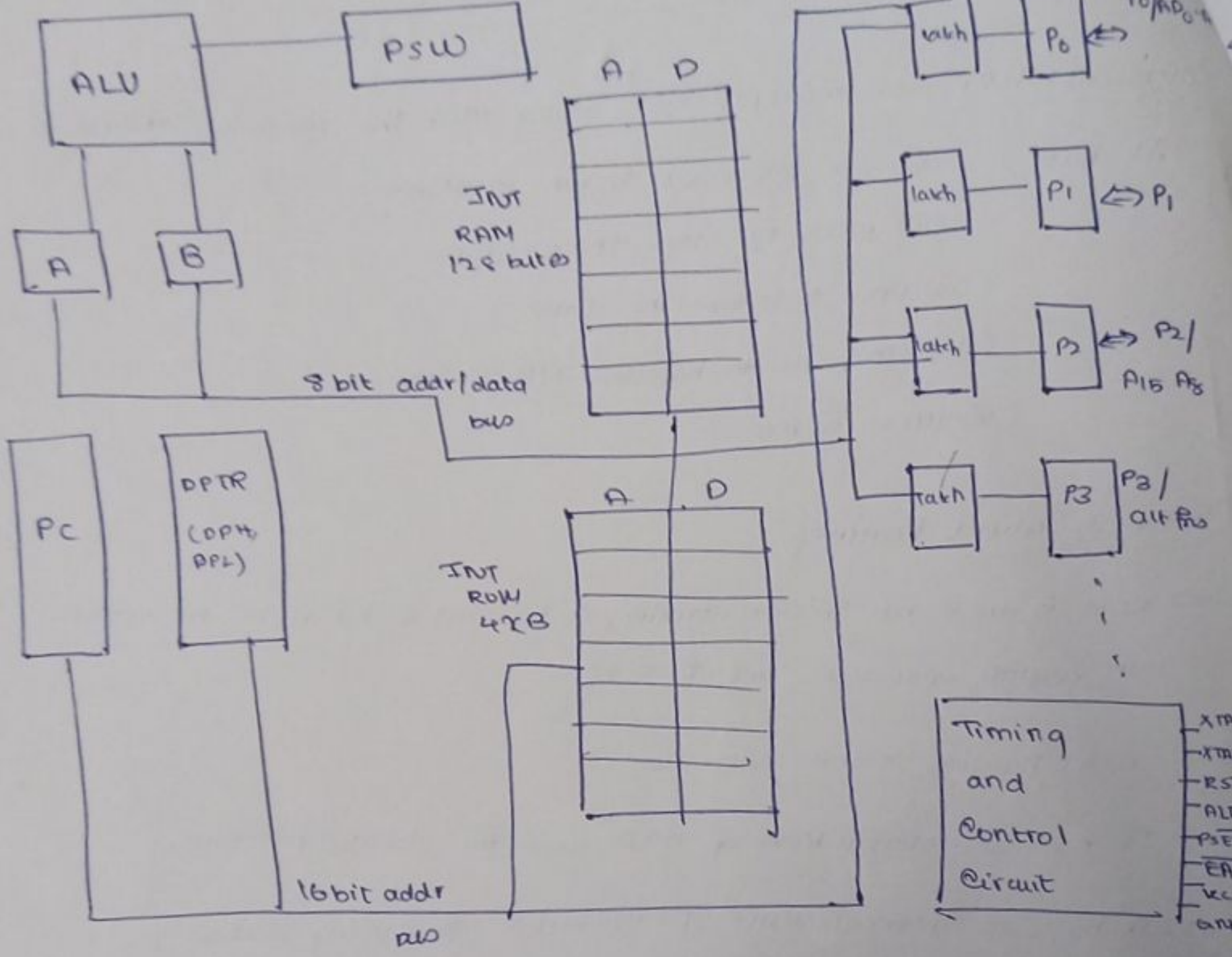
→ 8051 has two 16-bit timers, which are up counters

→ 8051 is a 40-pin IC.

* Architecture



¥ Architecture



Architecture has processor + memory + I/O

A. Processor

1. ALU - performs 8 bit arithmetic & logical operations

2. A Register (Accumulator) - an 8 bit register, it holds the first operation as well as get the result of the operation

3. B Register - an 8 bit register, dedicated for multiplication & division

4. PC - Program Counter - a 16-bit register. It holds the address of the next instruction in the program memory.

5. DPTR - Data pointer - a 16-bit register, it holds the address of data in the data memory

→ divided into

6. SP - Stack Pointer - an 8 bit register

→ contains address of the top of the stack

→ The stack is present in the internal RAM - has addresses from 00H-7FH.

→ The storing of a CPU register in the stack is called a PUSH. - increment SP by one.

→ Reading the contents of the stack back into a CPU register is called a POP

7. PSW - Program Status Word

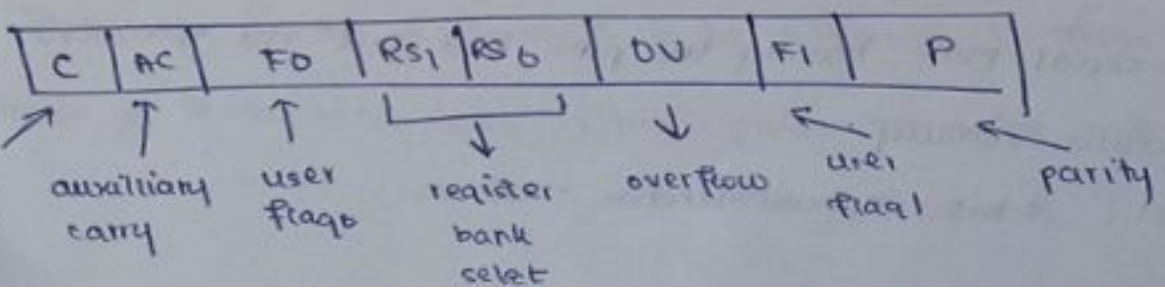
→ an 8-bit register

→ also called the Flag register, as it contains mainly the status flags

→ Those flags indicate status of the current result

→ They are changed by the ALU after every arithmetic or logic operation

→ The flags can also be changed by the programmer (PSW is a bit addressable register)



B. Memory

1. Internal RAM

→ 8051 has 128 bytes of Internal RAM

→ RAM is used to store data, hence it is called the data memory

→ There are 128 locations, each containing one byte of information

→ The address range is 00H - 7FH.

→ The 128 bytes are divided into 3 different groups:

(i) Register banks - 32 bytes - 00 - 1FH

(ii) Bit addressable - 16 bytes - 20H - 2FH

(iii) 80 bytes - 30H - 7FH - to read & write storage, called the scratch pad.

2. Internal ROM

→ 8051 has 4KB of Internal ROM

→ It stores programs, so it is also called the program memory or code memory.

→ There are 4K locations containing one byte of information

→ The address range is 0000H to 0FFFH.

→ Access programs w/ PC , data w/ DPTR

C. Input and Output

→ 8051 has 4 bit I/O ports - P0, P1, P2, P3

→ There are 2 16-bit timers

→ There is a serial port having the pins Rxd and Txd to receive and transmit data serially.

→ There are 21, 8 bit special function registers

8051 signals

① 8051 Oscillator and Clock

→ The oscillator is formed by the crystal, capacitors & an on-chip inverter.

→ Frequency - 1MHz - 16MHz

→ Time = (C x 12d) / crystal frequency

C = no. of machine cycles

XTAL1 and XTAL2

→ The on-chip oscillator requires an external clock to run it

→ A quartz crystal oscillator is connected to input XTAL1 & XTAL2

→ If a frequency source other than a crystal oscillator is used, it will be connected to XTAL1. & XTAL2 is left unconnected.

→ The speed of 8051 refers to the maximum oscillator frequency connected to XTAL.

→ The frequency can be observed on the XTAL2 pin using an oscilloscope.

② RST

→ The RESET pin is an input and is active high.

→ Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activities.

→ It is often referred to the power-on reset. Activating this will cause all values in the registers to be lost

→ In order for the RST input to be effective, it must have a minimum duration of 2 machine cycles (high pulse must be high for a min of 2 cycles)

③ EA'

→ EA' = external access is an input pin and must be connected to Vcc or GND.

→ 8051 has the capacity to have external code and data memory.

→ Normally EA pin is connected to Vcc. The EA pin must be connected to GND to indicate that the code or data is stored externally.

④ PSEN' and ALE

→ PSEN - 'program store enable' is an output pin.

→ This pin is connected to the OE pin of the external memory

External Code Memory - PSEN' = 0

External Data Memory - PSEN' = 1

→ The ALE pin is used to demultiplex the address and the data.

⑤ I/O Port Pins

A. Port 0 → designated as $AD_0 - AD_7$

→ When 8051 is connected to an external memory, Port 0 provides both address and data

→ ALE indicates if P0 has address or data

ALE = 0 ⇒ $P_0 - P_7$

ALE = 1 ⇒ $A_0 - A_7$

B. Port 1 and Port 2

→ In 8051 systems with no external memory connected, P_1 & P_2 are used as simple I/O.

→ In 8051-based systems with external memory connections:

(i) Port 2 must be used along with P0 to provide the 16 bit address for external memory

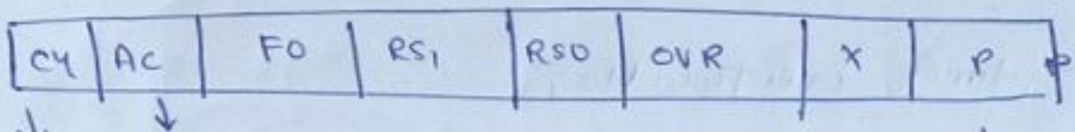
P0 - provides lower 8 bits - A_0-A_7

P2 - upper 8 bits - A_8 - A_{15} - cannot be used for I/O

Port 3

Bit	Function	
0	RxD	Serial Communication
1	TxD	
2	$\overline{INT0}$	External Interrupts
3	$\overline{INT1}$	
4	T0	Timers
5	T1	
6	$\overline{WR}$	Read / Write signals of external memories
7	$\overline{RD}$	

* Program Status Words (PSW)

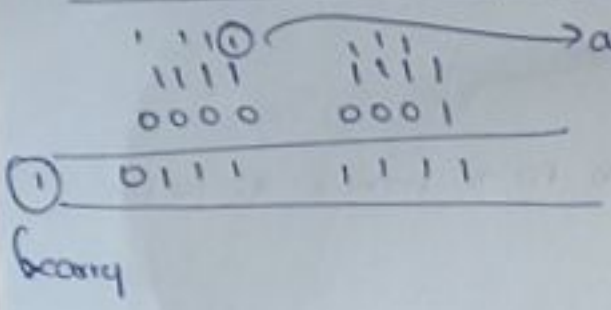


CY - carry flag
1 = carry out of MSB

AC - auxilliary carry (carry from one nibble to another)

P - parity
1 = odd
0 = even

example with carry & auxilliary carry.



* Special Function Registers (SFR)

① TL0, TH0, TL1, TH1

→ Timer 0 and Timer 1 consist of 2 SFRs each, TL0 & TL1 are the lower bytes and TH0 and TH1 are the higher bytes. to make it 16 bits in all.

② TCON Register

→ used to start or stop the timers of 8051. also indicates if the timers have overflowed.

→ It also consists of interrupt related bits

TF_1	TR_1	TF_0	TR_0	IE_1	IT_1	IE_0	IT_0

TF_1, TF_0 — timer overflow flag

TR_0, TR_1 — timer run control bit

IE_1, IE_0 - External interrupt edge flag - set by H/W when H to L transition is detected

IT_1, IT_0 - Interrupt type control bit - if 1 — on falling edge
- if 0 - occurs on low level

(3) TMOD

→ TMOD or timer mode is used to set the operating modes of the timers T0 and T1

Lower 4 bits = Timer 0

upper four bits = Timer 1

Gate	C/T	m_1	m_0	Gate	C/T	m_1	m_0

(4) SBUF - Serial Data Buffer

→ an 8 bit register solely used for serial communication

→ For a byte data to be transferred via the TxD line, it must be placed in the SBUF register.

→ It is framed with start and stop bits.

→ SBUF holds the byte of data when it is received by the 8051 RxD line

→ When RxD receives the bit, 8051 deframes it (eliminate start and stop bits), and then places it in SBUF.

(5) SCON - Serial Control

→ control the 8051's serial port

→ located at 98H

→ control operation modes, baud rates send or receive data

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0, SM1: serial mode specifier

SM2: ↓ μP communication

REN: enable reception

TB8, RB8: not widely used

TI: transfer

RI: receive interrupt flag.

⑥ IP - Interrupt Priority

-	-	PT_2	PS	PT_1	PX_1	PT_0	PX_0

PT_i = Timer i interrupt priority bit

PS = serial port interrupt bit

PX_i = external interrupt i priority bit

⑦ IE - Interrupt Enable

→ to enable or disable individual interrupts

EA	-	ET_2	ES	ET_1	EX_1	ET_0	EX_0

↗ disables all

for the rest follow the same as ⑥

⑧ PCON - power control

→ control the 8051 power modes
→ not bit addressable

SMOD	-	-	-	GF_1	GF_0	PD	IDL

GF_1, GF_0 → general purpose flag
PD → power down
IDL → Idle

→ when 8051 is powered up, SMOD is zero

→ It can be set to high by software & double the baud rate

8051 Instruction Set

Arithmetic Instructions

① ADD

ADD A, #n

ADD A, @R0

② ADDC

ADDC A, #n

ADDC A, @R0

③ SUBB (note that there is no ordinary sub instruct, to do 8-bit subtraction, where the carry is not needed, must use CLR C first)

SUBB A, #n

SUBB A, @R0

④ INC, DEC

INC A

INC Rn

INC @R0

INC DPTR

⑤ MUL AB = B.A = A×B

⑥ DIV AB = B (rem) . A (q) ← A÷B

⑦ DA = decimal adjust — To add BCD numbers use add instruction then use DA to convert it back to BCD

Logical Instructions

① ANL A, @R0
A, #n

} used to clear any bit
Take og number. Take another number - set the bit to be cleared with 0. and do and operation

② ORL A, #n
A, @R0

} used to set any bit
Take an og number. set or that bit with 1 and set the remaining with 0

③ XRL - logical XOR operation
XRL A, #25
A, @R0

} used to complement any particular bit
XOR with 1, remaining with 0

④ RL, RR - rotate left / right
rotate n number of times to check value in carry flag

⑤ RRC, RLC - rotate right / left with the carry

⑥ CPL A = 1's complement, used as a NOT operation

⑦ SWAP A = swap lower & higher nibbles

Other Instructions

1. MOVX — get data from location pointed to
   eg. MOVX A, @DPTR
2. PUSH / POP — push & pop from stack

**

3. XCH — exchange values between registers
   XCH A, R0
   XCH A, @R0

**

4. XCHD — exchange digit
   exchange only lower nibble in both numbers
   XCHD A, @R0

Branch Instructions

1. SJMP = short jump - unconditional
2. AJMP, LJMP
3. DJNZ R0, label - decrement and jump if not 0
4. CJNE A, #n, label - compare A, #n & jump to label if not equal
5. JC, JNC, JZ, JNZ

⑥ Bit Setting Instructions

(i) JB, JNB – jump if / if not bit — JB bit, label

(ii) JBC bit, label – jump if bit & complement / clear — JB P0.0, label

JB P0.0 down

⇒ if P0.0 = 1 jump to down & make P0.0=0

Microprocessors

Programming in 8051

(1) 8 bit addition / subtraction

```
MOV  R0, #00
MOV  A, #06
ADD  A, #01        (or SUBB)
INC  label
INC  R0
label: MOV DPTR, #4150
MOVX @DPTR, A
MOV  A, R0
INC  DPTR
MOVX @DPTR, A
HERE : SJMP HERE
```

(2) 8 bit multiplication / division

```
MOV  A, #03
MOV  B, #05
MUL  AB            (or DIV AB)
MOV  DPTR, #4150
MOVX @DPTR, A
INC  DPTR
MOV  A, B
MOVX @DPTR, A
HERE : SJMP HERE
```

(3) Cube of a number

```
MOV  A, #03
MOV  B, #03
MOV  R0, #03
MUL  AB
MOV  B, R0
MUL  AB
MOV  DPTR, #4150
MOVX @DPTR, A
INC  DPTR
MOV  A, B
MOVX @DPTR, A
HERE: SJMP HERE
```

(4) BCD to ASCII

```
mov A, #23
mov R0, A        → make a copy
ANL A, #F0H      → bitwise AND - get higher bits
SWAP A           → (like the ROR 4x operation)

mov R1, A        → store the higher order bit value
                   (now 02)
mov A, R0        → processing lower bits
ANL A, #0FH      → get lower bits

mov R0, A        → store the lower order bits - now (03)

mov A, R0
ADD A, #30
mov DPTR, #4150
movx @DPTR, A

mov A, R1
ADD A, #30
INC DPTR
movx @DPTR, A

here: SJMP here
```

ASCII to BCD

```
MOV   DPTR, #5300
MOVX  A, @DPTR

CJNE  A, #40, noeq      ← check if accumulator value = 40
                          if not equal go to noeq
noeq: JC  subthirty     ← check if there is a carry
                          if there is carry => must
                                               subtract 30,
      CLR  C                                   otherwise must subtract 7
      SUBB A, #07

subthirty:
      CLR  C
      SUBB A, #30

      INC  DPTR
      MOVX @DPTR, A

Here: SJMP Here.
```

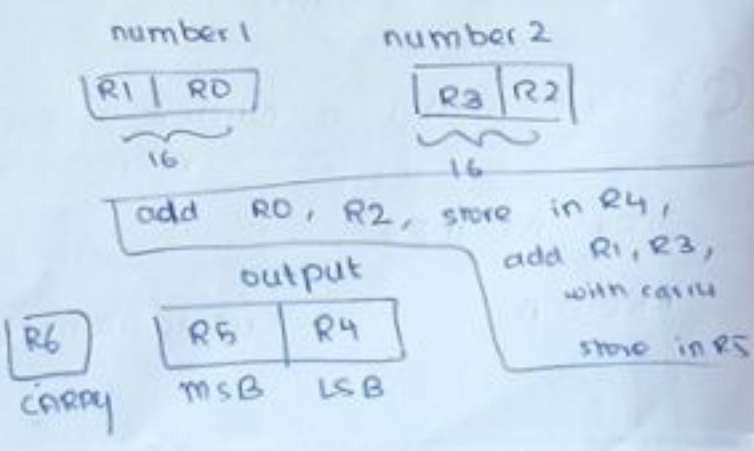
(6) 16-bit addition

```
MOV  R0, #11
MOV  R1, #22
MOV  R2, #33
MOV  R4, #44      } initialization

MOV  R6, #00      ← store carry
```

method

use R6 ~~for~~ carry



```
MOV A, R0        ← move LSB of num1 to R0
ADD A, R2        ← ADD LSB of num2 with accumulator

MOV R4, A        ← move result from lower bits in R4

MOV A, R3        ← move MSB of num1 to R0
ADDC A, R3       ← add MSB of num2 to accumulator with carry

JNC next
    INC R6       ← If there is a carry, increment R6

next: MOV DPTR, #4150
      MOVX @DPTR, A      ← store MSB at 4150

      INC DPTR
      MOV A, R4          ← move LSB to accumulator
      MOVX @DPTR, A      ← store LSB at 4151

      INC DPTR
      MOV A, R6          ← move carry to accumulator
      MOVX @DPTR, A      ← store carry at 4152

here: SJMP here
```

(7) Check whether a number is even or odd (store 00 if even FF if odd)

```
MOV A, #33
RRC A        → rotate right with carry
```

```
    JNC   L1          → if there is no carry ⇒ even

    MOV   DPTR, #4150
    MOV   A, #0FFH
    MOV   @DPTR, A
    SJMP  L2
L1: MOV   DPTR, #4150
    MOV   A, #00H
    MOV   @DPTR, A
L2: SJMP  L2
```

(8) Find the sum of n elements present in an array.

```
MOV   DPTR, #4200     → starting address of array
MOV   R0, #05         → no. of elements in the array
MOV   R1, #00         → will hold the cumulative count
SUMLOOP: MOV A, @DPTR,   → load value from array into A
         ADD  A, R1      ← add and store in accumulator
         MOV  R1, A      ← store sum in R1
         INC  DPTR       ← move to next location
         DJNZ R0, SUMLOOP  ← decrement R0 (counter)
                             & jump to SUMLOOP if not zero
         MOV  A, R1      ← store cumulative sum in A
         MOV  DPTR, #4500
         MOVX @DPTR, A   ← store final sum at 4500
HERE:    SJMP HERE
```

(9) Find the count of even and odd numbers in an array

```
mov   DPTR , #4300      → starting address of the array
mov   R3 , #05          → no. of elements in the array
mov   R2 , #00          → no. of even numbers
mov   R1 , #00          → no. of odd numbers
CLR   C

divloop:  mov  B, #02
          movx  A, @DPTR    → load value from array into A
          DIV  A B
                            →divide - remainder in B
          mov  R0, B        → move remainder to R0
          CJNE  B, #00, odd → compare B and 00 & jump
                                to odd if not equal
          INC  R2           → increment even count
          SJMP  nextitr
even :
odd:      INC  R1   →increment odd counter
nextitr:  INC DPTR
          DJNZ R3, divloop      → decrement counter, go back
          mov  DPTR, #4300        to checking inside array
          mov  A, R2            → storing even count
          movx @DPTR, A
          mov  A, R1            → storing odd count
          INC  DPTR
          movx @DPTR, A
here :    SJMP  here.
```

Find the largest number in an array

```
mov   DPTR, #4200
mov   A, @DPTR        → start of array
mov   R0, A           → no. of elements in array

mov   B, #00          → assume that 00 is the largest number

Loop
      INC  DPTR
      movX A, @DPTR   → move element from array
      CJNE A, B carrycheck   → compare and jump if not equal to each other.

carrycheck : JC  decrement   Then check for a carry in carrycheck to find if bigger or smaller
      mov  B, A

decrement : DJNZ R0, Loop   → if there is a carry A<B, no need to change anything go back to Loop where decrement happens

      mov  DPTR, #4500      → decrement counter & go back to the Loop again
      mov  A, B
      movX @DPTR

here : SJMP here.
```

(11) Convert hexadecimal to BCD

```
MOV  DPTR, #4500
MOVX A, @DPTR
MOV  B, #64
DIV  AB          (B= Rem, A= Quotient)
INC  DPTR
MOVX @DPTR, A    → store Quotient

MOV  A, B        → move remainder in to A

MOV  B, #0AH

DIV  AB          → (B= rem, A= quotient)

INC  DPTR

MOVX @DPTR, A    → store Q
MOV  A, B
INC  DPTR        → move remainder

MOVX @DPTR, A
here: SJMP here
```

(12) Convert BCD to hexadecimal

```
MOV  DPTR, #4500
MOVX A, @DPTR
MOV  R0, A       → store a copy
ANL  A, #F0      → extract higher bits
SWAP A

MOV  B, #0A
MUL  AB
```

```
MOV  R1, A   ← higher bits stored
MOV  A, R0   ← store copy in A
ANL  A,#0F   ← get lower bits
ADD  A, R1   ← add lower bits to (higher bits swapped ×10)
                in R1 now
INC  DPTR
MOVX @DPTR, A ← store hex value
here: SJMP here
```

(13) Copy a block of data from one location to another.

```
MOV  R0, #30H   → start of the array
MOV  R1, #0AH   → 10 elements
MOV  R2, #40H   → dest array
loop:
MOV  A, @R0
XCH  A, @R2     (MOV @R1, A ?)
INC  R0
INC  R2
DJNZ R2, loop
here: SJMP here
```

alternatively:

```
MOV  R0, #20H → src
MOV  R1, #30H → dest
MOV  R7, #0AH → count
back: MOV A, @R0
      MOV @R1, A
      INC R0
      INC R1
      DJNZ R7, back
here: sjmp here
```

(14) Write a program in 8051, to add 10 BCD numbers

```
mov DPTR, #2000H    → start of array
mov R1, #0AH        → count
mov R0, #00         → will load each element from
                      array into R0
CLR A               → set the accumulator to 0
mov B, #00H
repeat: movx A, @DPTR
        ADD A, #0000 R0
        JNC skip
        INC 0F0H
                B
skip:   INC DPTR
        mov R0, A
        DJNZ R1, repeat

//outside the loop
        mov DPTR, #3000H
        movx @DPTR, A
        INC DPTR
        mov A, B    → carry
        mov @DPTR, A
here: SJMP here.
```

Inverted Block Transfer Program

```
mov  R0, #20H
mov  R1, #30H
mov  R7, #0AH

mov  A, R1
ADD  A, R7
mov  R1, A

back: mov  A, @R0
      mov  @R1, A
      INC R0
      DEC R1
      DJNZ R7, back
here: SJMP here
```

(16) Sorting an Array of Numbers

```
start: mov  R7, #04   → outer counter

outer: mov R6, #04  → inner counter
       mov DPTR, #3000H → start of array

inner: movx A, @DPTR → load no.1
       INC DPTR
       movx B, @DPTR → load no.2
       CJNE A, B, check carry → check if the 2 numbers are equal
```

```
checkcarry:   JC   noswap        if there is no carry dont swap
              MOVX @DPTR, A
              DEC DPTR
              MOVX @DPTR, B
              INC DPTR

noswap:       DJNZ R6, inner
              DJNZ R7, outer

here:  SJMP here
```

@paper@

(17) Find the count of the number of 1s in an 8 bit number

```
      MOV   R0, #08H
      MOV   A, #33H
      MOV   R1, #00H
loop: RRC   A
      JC    add
      DJNZ  R0, loop
      MOV   DPTR, #4150
      MOV   A, @R1
      MOVX  @DPTR, A
      SJMP  here
add:  INC   R1
here: SJMP  here
```