Module V

8051 Architecture- Register Organization- Memory and I/O addressing- Interrupts and Stack- 8051 Addressing Modes- Instruction Set- data transfer instructions, arithmetic instructions, logical instructions, Boolean instructions, control transfer instructions- Simple programs.

Microcontroller

A microcontroller is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave's information, receiving remote signals, etc.

The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc.

Difference between Microprocessor and Microcontroller

Microcontroller	Microprocessor
Microcontrollers are used to execute a single task within an application.	Microprocessors are used for big applications.
Its designing and hardware cost is low.	Its designing and hardware cost is high.
Easy to replace.	Not so easy to replace.
It is built with CMOS technology, which requires less power to operate.	Its power consumption is high because it has to control the entire system.
It consists of CPU, RAM, ROM, I/O ports.	It doesn't consist of RAM, ROM, I/O ports. It uses its pins to interface to peripheral

8051 microcontroller

Features

- 8 bit cpu with registers A and B
- 16 bit PC and DPTR(data pointer).
- 8 bit program status word(PSW)
- 8 bit Stack Pointer
- 4K Internal ROM

- 128bytes Internal RAM

  →4 register banks each having 8 registers

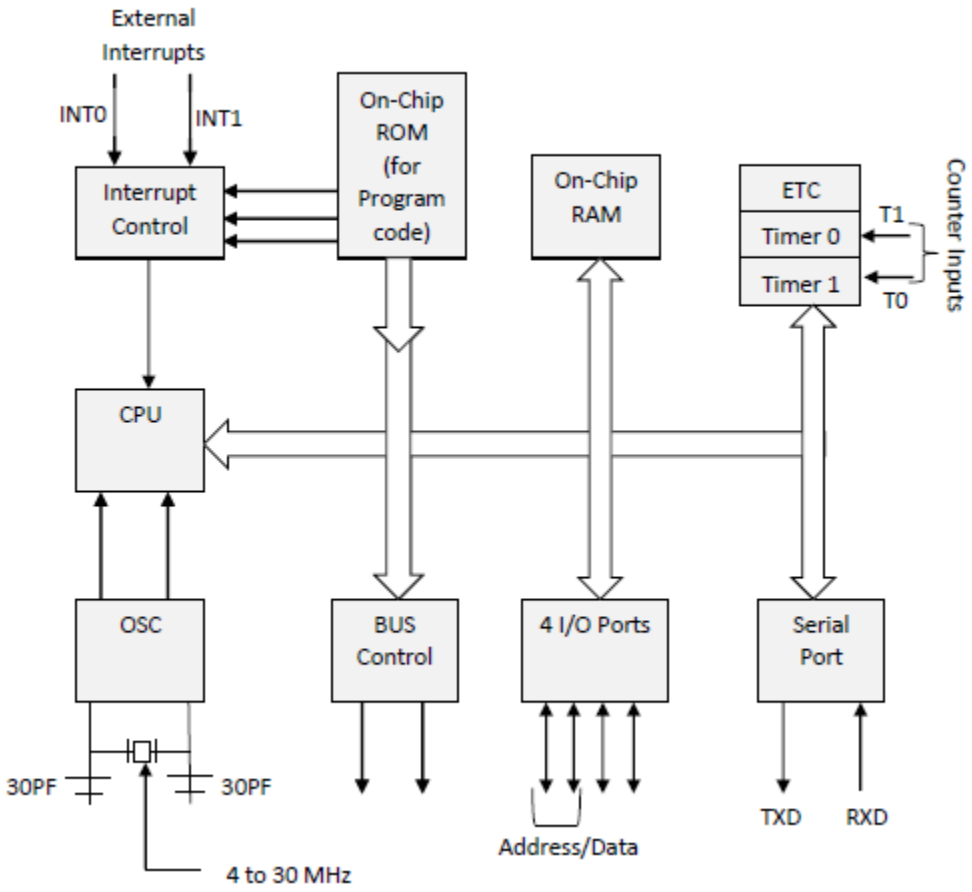
  →16 bytes,which may be addressed at the bit level.

  →80 bytes of general purpose data memory
- 32 i/o pins arranged as 4 8 bit ports:P0 to P3
- Two 16 bit timer/counters:T0 and T1
- Full duplex serial data receiver/transmitter
- Control registers:TCON,TMOD,SCON,PCON,IP and IE
- Two external and Three internal interrupt sources.

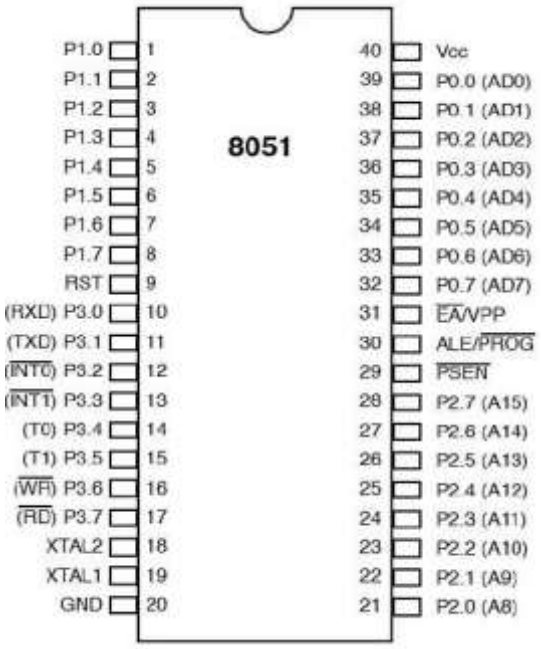
Oscillator and Clock Circuits.

Architecture of 8051 Microcontroller

In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.



The pin diagram of 8051 microcontroller looks as follows −



- **Pins 1 to 8** − These pins are known as Port 1. This port doesn't serve any other functions. It is internally pulled up, bi-directional I/O port.
- **Pin 9** − It is a RESET pin, which is used to reset the microcontroller to its initial values.
- **Pins 10 to 17** − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.
- **Pins 18 & 19** − These pins are used for interfacing an external crystal to get the system clock.
- **Pin 20** − This pin provides the power supply to the circuit.
- **Pins 21 to 28** − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.
- **Pin 29** − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.
- **Pin 30** − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.
- **Pin 31** − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.
- **Pins 32 to 39** − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.
- **Pin 40** − This pin is used to provide power supply to the circuit.

Registers in 8051

DATA POINTER (DTPR):- DTPR is a 16 bit register.

It consists of higher byte (DPH) and a lower byte (DPL).

DPTR is used to hold the external data memory address of data being fetched.

PC (Program Counter)

- 16 bit register

- Hold address of instruction being currently fetched.
- Increments continuously to point to the next instruction.

Special Function registers

ACCUMULATOR (A):-8 bit register

it is used for data transfer and arithmetic operations. After any operation result is stored in A

B REGISTER:- it is used to store the upper 8 bit result of multiplication and divisions. It is used as temporary register

PROGRAM STATUS WORD (PSW):- This special function registers and consists of different status bits that reflect the current state of microcontroller.

It contains carry (CY), the auxiliary carry(AC), the two registers bank select bits(RS1 and RS0), the overflow flag(OV), a parity bit(P), and two user defined status flags.

CY	AC	F0	RS1	RS0	OV	-	P

STACK POINTER (SP):- This is an 8 bit register. SP is incremented before the data is stored onto the stack using PUSH/CALL instructions execution. During PUSH, first SP is incremented and then copy the data. In the POP operation, initially copy the data and then decrement the SP.

PCON (Power Control): The Power Control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow going "sleep" mode which requires much less power. These modes of operation are controlled through PCON.

TCON (Timer Control, Bit-Addressable): The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed.

TMOD (Timer Mode): The Timer Mode SFR is used to configure the mode of operation of each of the two timers. each timer to be a 16-bit timer, an 8-bit auto reload timer, a 13-bit timer, or two separate timers.

TL0/TH0 (Timer 0 Low/High): These two SFRs, taken together, represent timer 0.

TL1/TH1 (Timer 1 Low/High): These two SFRs, taken together, represent timer 1.

SCON (Serial Control): The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SBUF (Serial Control): The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Likewise, any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF.

IE (Interrupt Enable): The Interrupt Enable SFR is used to enable and disable specific interrupts.

IP (Interrupt Priority): The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority.

PORT0, PORT1, PORT2, PORT3 LATCHES AND DRIVERS:- Each latch and corresponding drivers of port 0-3 is allotted to the corresponding on chip I/O port.

Port 0

Port-0 can be used as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory.

When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a bidirectional I/O port.

PORT 1

Port-1 dedicated only for I/O interfacing.

PORT 2:

Port-2 we use for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access.

PORT 3:

It works as an IO port same like Port 2 as well as it can do lots of alternate work Following are the alternate functions of port 3:

P3.0—RXD

P3.1— TXD

P3.2— INT0 BAR

P3.3— INT1 BAR

P3.4— T0

P3.5— T1

P3.6— WR BAR

P3.7— RD BAR

ALU

A unit to perform an arithmetic or logic operation at an instance

SI (Serial Interface)

- The 8051 contains UART – universal asynchronies receiver transmitter
- The serial port is full duplex
- It can transmit and receive simultaneously
- Two Port 3 pins are used to provide the serial interface
- P 3.0 is the receiver pin (RXD) P 3.1 is the transmitter pin (TXD)
- Both synchronous and asynchronous transmission supported

Register SCON controls serial data communication

Power Mode control Register

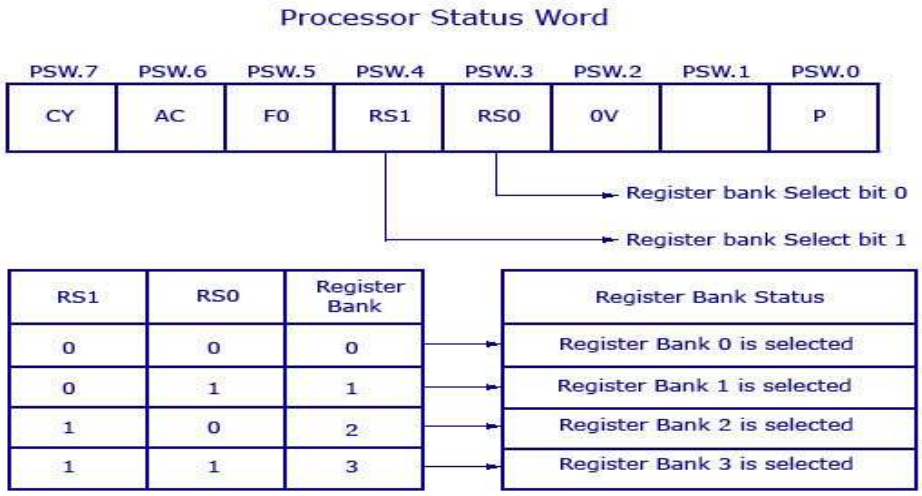
Register PCON controls processor power down, sleep modes and serial data band rate.

Timers

There are two 16-bit timers and counters in 8051 microcontroller: **timer 0 and timer 1**. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters.

Counters and Timers in 8051 microcontroller contain two special function registers: **TMOD (Timer Mode Register) and TCON (Timer Control Register)**, which are used for activating and configuring timers and counters.

PSW



Memory organization in 8051

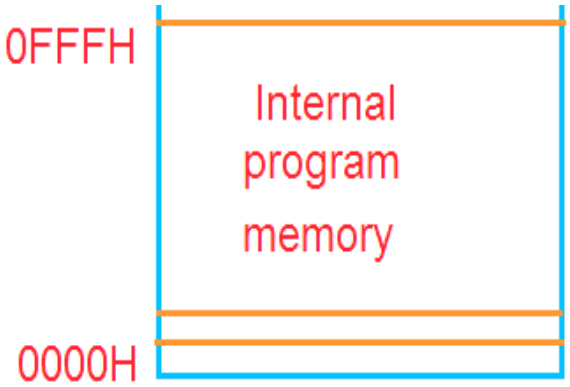
The 8051 microcontroller's memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)

Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only. Depending on the settings made in compiler, program memory may also used to store a constant variables. The 8051 executes programs stored in program memory only.

Internal Program memory

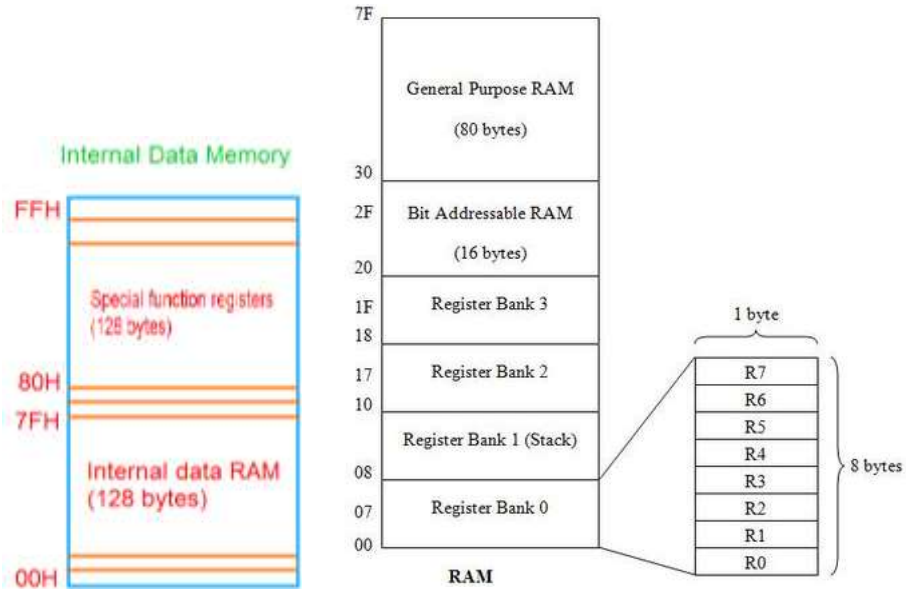
Program memory accessed through EA pin. In program memory two categories takes place:



a)If EA is high, internal program memory is accessed to 0FFFH memory location and external program memory accessed from 1000H to FFFFH memory locations.

b)If EA is low, only external program memory accessed from 0000H to FFFFH memory locations.

Internal Data Memory



The internal data memory consists of 256 bytes, these are divided into two parts:

00H-FFH for internal data RAM (128 bytes)

80H-FFH for special function registers (128 bytes)

Total 128 Byte memory is divided into three parts.

- 32 byte
- 16 byte
- 80 byte

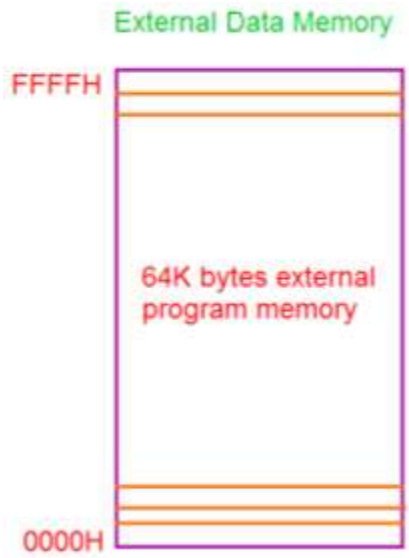
Here 16 byte classification is bit addressable. In micro-controller registers were data is stored, if one could manipulate its content bit by bit it's called bit addressable.

External Program Memory

a)If EA is high, internal program memory is accessed to 0FFFH memory location and external program memory accessed from 1000H to FFFFH memory locations.

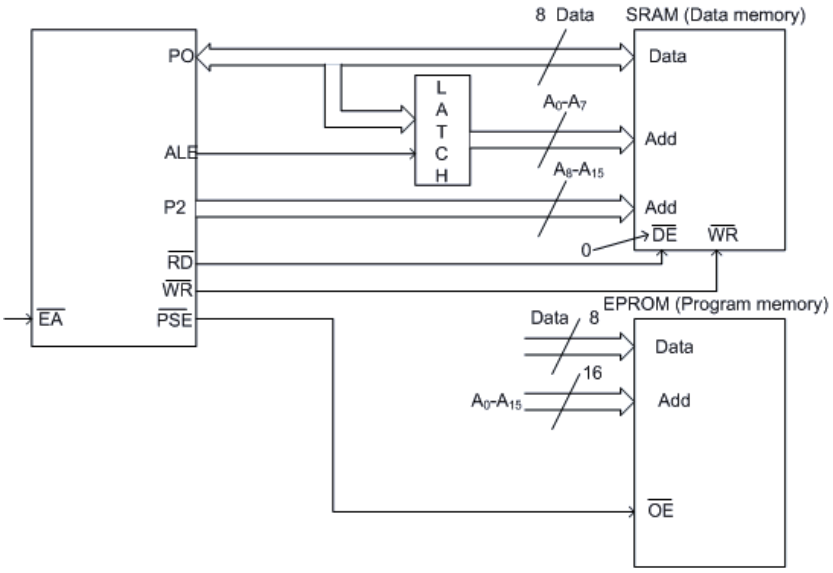
b)If EA is low, only external program memory accessed from 0000H to FFFFH memory locations.

External Data Memory



Interfacing of external memory with 8051

- External memories are interfaced using four control signals. (PSEN, ALE, RD and WR)
- ALE signal separate the A0- A7 bus for the program and X – data memories.
- PSEN signal when 0, uses the program memory code bank 2 to 31 for the code reading operation. Uses bank 0 and bank 1 also when EA is 0 during the RESET of MCU
- The RD when 0 , uses data memory for the Xdata reading operation(X Data means external data)
- The WR when 0 , uses data memory for the data write operation(X Data means external data)



I/O addressing in 8051

- The 8051 microcontroller has four parallel I/O ports , each of 8-bits .So, it provides the user 32 I/O lines for connecting the microcontroller to the peripherals.
- The four ports are P0 (Port0), P1(Port1) ,P2(Port 2) and P3 (Port3).

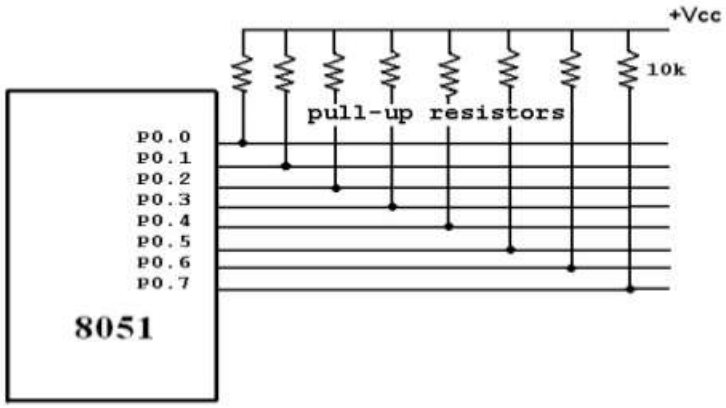
- Upon reset all the ports are output ports. In order to make them input, all the ports must be set i.e a high bit must be sent to all the port pins.
- This is normally done by the instruction

Ex: MOV A,#0FFH ; A = FF MOV P0,A ; make P0 an input port

PORT 0:

Port 0 is an 8-bit I/O port with dual purpose. If external memory is used, these port pins are used for the lower address byte address/data (AD0-AD7), otherwise all bits of the port are either input or output.

Unlike other ports, Port 0 is not provided with pull-up resistors internally, so for PORT0 pull-up resistors of nearly 10k are to be connected externally as shown in the fig



(A pull-up resistor weakly "pulls" the voltage of the wire it is connected to towards its voltage source level when the other components on the line are inactive.)

Dual role of port 0:

Port 0 can also be used as address/data bus(AD0-AD7), allowing it to be used for both address and data.

When connecting the 8051 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save the pins. ALE indicates whether P0 has address or data. When ALE = 0, it provides data D0-D7, and when ALE =1 it provides address and data with the help of a 74LS373 latch.

Port 1: Port 1 occupies a total of 8 pins (pins 1 through 8). It has no dual application and acts only as input or output port. In contrast to port 0, this port does not need any pull-up resistors. Since pull-up resistors connected internally.

Upon reset, Port 1 is configured as an output port.

To configure it as an input port , port bits must be set i.e a high bit must be sent to all the port pins.

For Ex :

MOV A, #0FFH; A=FF

MOV P1,A ; make P1 an input port by writing 1's to all of its pins

Port 2 : Port 2 is also an eight bit parallel port. (Pins 21- 28). It can be used as input or output port. As this port is provided with internal pull-up resistors it does not need any external pull-upresistors.

Upon reset, Port 2 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port.

For ex,MOV A, #0FFH ; A=FF

MOV P2, A ; make P2 an input port by writing all 1's to it

Dual role of port 2: Port2 lines are also associated with the higher order address lines A8-A15.In systems based on the 8751, 8951, and DS5000, Port2 is used as simple I/O port

PORT 3 : Port3 is also an 8-bit parallel port with dual function.(pins 10 to 17).

The port pins can be used for I/O operations as well as for control operations. The details of these additional operations are given below in the table. Port 3 also do not need any external pull-up resistors as they are provided internally similar to the case of Port2 & Port 1. Upon reset port 3is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port.

For ex,MOV A, #0FFH ; A= FF

MOV P3, A ; make P3 an input port by writing all 1's to it

Alternate Functions of Port 3 :

S.No	Port 3 bit	Pin No	Function
1	P3.0	10	RxD
2	P3.1	11	TxD
3	P3.2	12	$\overline{\text{INT0}}$
4	P3.3	13	$\overline{\text{INT1}}$
5	P3.4	14	T0
6	P3.5	15	T1
7	P3.6	16	$\overline{\text{WR}}$
8	P3.7	17	$\overline{\text{RD}}$

Two modes of operation

1. Single chip mode
2. Expanded Multiplexed mode

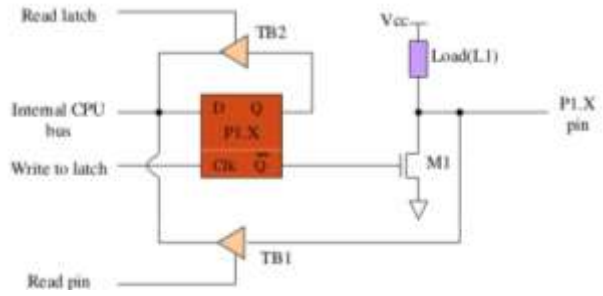
Single chip mode

- In which all the software and data internally embedded into the MCU and external memory chips are not used.

Expanded Multiplexed mode

- In this mode all software embedded either internally or externally.

Hardware Structure of I/O Pin



- Each pin of I/O ports
  - Internally connected to CPU bus
  - A D latch store the value of this pin
    - Write to latch = 1 ： write data into the D latch
  - 2 Tri-state buffer ：
    - TB1: controlled by "Read pin"
      - Read pin = 1 ： really read the data present at the pin
    - TB2: controlled by "Read latch"
      - Read latch = 1 ： read value from internal latch
  - A transistor M1 gate
    - Gate=0: open
    - Gate=1: close

Interrupts in 8051 microcontroller

- An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.
- A set of program instructions written to service an interrupt is called the Interrupt Service Routine

• 8051 has six different sources of interrupts

External: Power-up reset, INT0, INT1

Internal: Timer0, Timer1, Serial Port

Interrupt Service Routine

• When microcontroller receives an interrupt signal from any of the six interrupt sources it executes a call to interrupt service routine

• For every interrupt, there must be an interrupt service routine

• The interrupt service routine for every interrupt must be located at a fixed location in program memory, called interrupt vector.

How 8051 services an interrupt request

• 8051 finishes the instruction it is currently executing, and saves the contents of Program Counter on the stack (address of next instruction)

• It jumps to the interrupt vector location corresponding to the interrupt source

• Executes the interrupt service routine, until it encounters RETI instruction

• Returns back to the place where it was interrupted, by popping the contents of stack on PC, and starts execution at that address

Special Function Register to handling Interrupt

IE(Interrupt Enable Register)

IP(Interrupt priority Register)

Sources of Interrupt

Interrupt Enable register (IE):

EA	-	-	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

IP(Interrupt priority Register)

-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.
PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

Interrupt Vector Table

Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

Stack in 8086

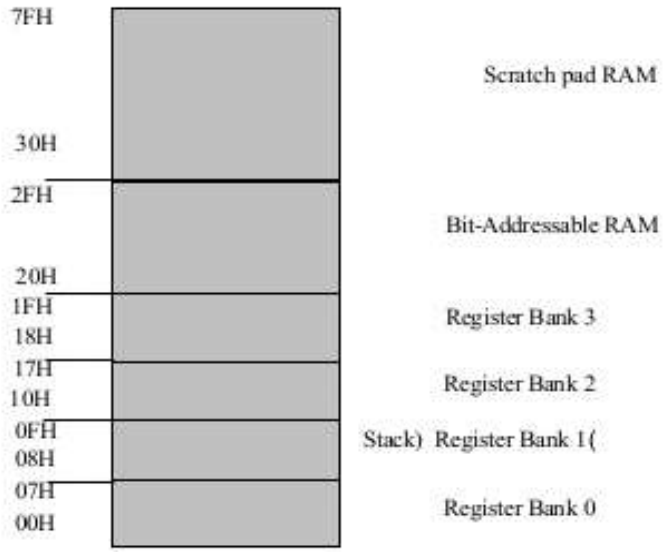
The stack is an area of random access memory (RAM) to hold the variables temporarily.

The stack is also responsible for storing address a function is called so that it can be returned correctly.

Whenever the function is called, the parameters and local variables associated with it are added to the stack (PUSH).

When the function returns, the parameters and the variables are removed ("POP") from the stack. This is why a program's stack size changes continuously while the program is running.

The register used to access the stack is called **stack pointer register**. The stack pointer is a small register used to point at the stack. When we push something into the stack memory, the stack pointer increases.



- The stack pointer in the 8051 is only 8 bits wide, which means that it can take values of 00 to FFH.
- When the 8051 is powered up, the SP register contains value 07.
- This means that RAM location 08 is the first location used for the stack by the 8051.
- The storing of a CPU register in the stack is called a PUSH, and pulling the contents off the stack back into a CPU register is called a POP.
- In other words, a register is pushed onto the stack to save it and popped off the stack to retrieve it.

Pushing onto the stack

In the 8051 the stack pointer (SP) points to the last used location of the stack. As we push data onto the stack, the stack pointer (SP) is incremented by one.

Notice that this is different from many microprocessors, notably 8086 processors in which the SP is decremented when data is pushed onto the stack.

we see that as each PUSH is executed, the contents of the register are saved on the stack and SP is incremented by 1.

For example, the instruction "PUSH 1″ pushes register Rl onto the stack.

Example 2-8

Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.

```
MOV  R6,#25H
MOV  R1,#12H
MOV  R4,#0F3H
PUSH 6
PUSH 1
PUSH 4
```

Solution:

	After PUSH 6	After PUSH 1	After PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Start SP = 07	SP = 08	SP = 09	SP = 0A

Popping from the stack

Popping the contents of the stack back into a given register is the opposite process of pushing. With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once. Example demonstrates the POP instruction.

Example 2-9

Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.

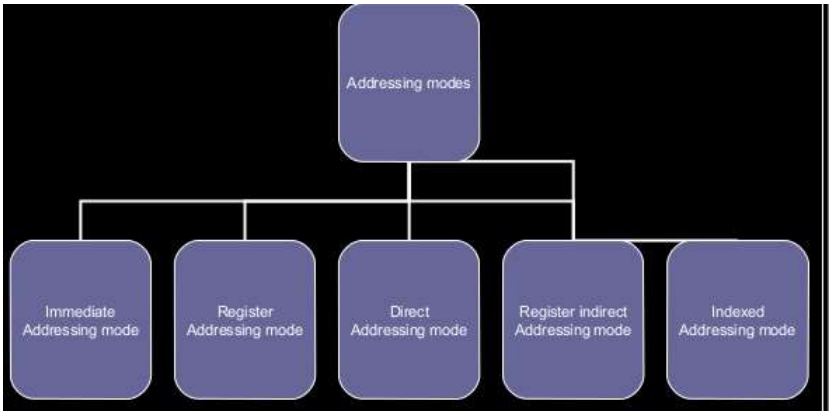
```
POP  3    ;POP stack into R3
POP  5    ;POP stack into R5
POP  2    ;POP stack into R2
```

Solution:

	After POP 3	After POP 5	After POP 2
0B 54	0B	0B	0B
0A F9	0A F9	0A	0A
09 76	09 76	09 76	09
08 6C	08 6C	08 6C	08 6C
Start SP = 0B	SP = 0A	SP = 09	SP = 08

Addressing Modes

The way of accessing data is called addressing mode. The CPU can access the data in different ways by using addressing modes. The 8051 microcontroller consists of five addressing modes such as:



Immediate Addressing Mode:

In this addressing mode, the source must be a value that can be followed by the '#' and destination must be SFR registers, general purpose registers and address. It is used for immediately storing the value in the memory registers.

Syntax:

```
MOV A, #20h    //A is an accumulator register, 20 is stored in the A//
MOV R0,#15     // R0 is a general purpose register; 15 is stored in the R0 register//
MOV P0, #07h   //P0 is a SFR register;07 is stored in the P0//
MOV 20h,#05h   //20h is the address of the register; 05 stored in the 20h//
MOV DPTR,#4532H // DPTR=4532H.
```

Register Addressing Mode:

In this addressing mode, the source and destination must be a register, but not general purpose registers. So the data is not moved within the general purpose bank registers.

Syntax:

```
MOV A, B;      // A is a SFR register, B is a general purpose register//
MOV R0, R1     //Invalid instruction, GPR to GPR not possible//
```

Direct Addressing Mode

In this addressing mode, the source or destination (or both source and destination) must be an address, but not value.

Direct addressing mode In direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction. Contrast this with the immediate addressing mode in which the operand itself is provided with the instruction.

Syntax:

```
MOV A,20h      // 20h is an address; A is a register//
MOV 00h, 07h   // both are addressed of the GPS registers//
```

Indirect Addressing Mode (Register Indirect mode):

In this addressing mode, the source or destination (or destination or source) must be a indirect address, but not a value

In the register indirect addressing mode, a register is used as a pointer to the data. If the data is inside the CPU, only register R0 and R1 are used for this purpose. In other words, R2-R7cannot be used to hold the address of an operand located in RAM when using this addressing mode.

When R0 and R1 are used as pointers, that is, when they hold the address of RAM locations, they must be preceded by the "@" sign.

Ex: - MOV A,@R0 // move contents of RAM location whose address is held by R0 into A.

MOV @R1, B // move contents of B RAM location whose address is held by R1

Base Index Addressing Mode:

This addressing mode is used to read the data from the external memory or ROM memory. All addressing modes cannot read the data from the code memory. The code must read through the DPTR register. The DPTR is used to point the data in the code or external memory.

Syntax:

MOVC A, @A+DPTR //C indicates code memory//

MOCX A, @A+DPTR // X indicate external memory//

Because the data elements are stored in the program space ROM of the 8051,it uses the instruction MOVC instead of MOV.

The 16-bit register DPTR and register "A" are used to form the data element stored in on-chip ROM.

Instruction Set

The 8051 microcontroller can follow CISC instructions with Harvard architecture. In case of the 8051 programming different types of CISC instructions include:

- Data Transfer Instruction set
- Arithmetic Instruction set
- Branching Instruction set
  - Conditional Branching
  - Unconditional Branching
- Logical Instruction set
- Bit Oriented Instruction set

Data Transfer Instruction set

Mnemonics	Operational description	Addressing mode	No. of bytes occupied
Mov a,#num	Copy the immediate data num in to acc	immediate	2

Mov Rx,A	Copy the data from acc to Rx	register	1
Mov A,Rx	Copy the data from Rx to acc	register	1
Mov Rx,#num	Copy the immediate data num in to Rx	immediate	2
Mov A,addr	Copy the data from direct address to acc	direct	2
Mov addr,A	Copy the data from acc to direct address	direct	2
Mov addr,#num	Copy the immediate data num in to direct address	direct	3
Mov addr1,addr2	Copy the data from address2 to address1	direct	3
Mov Rx,addr	Copy the data from direct address to Rx	direct	2
Mov addr,Rx	Copy the data from Rx to direct address	direct	2
Mov @Rp,A	Copy the data in acc to address in Rp	Indirect	1
Mov A,@Rp	Copy the data that is at address in Rp to acc	Indirect	1
Mov addr,@Rp	Copy the data that is at address in Rp to address	Indirect	2
Mov @Rp,addr	Copy the data in address to address in Rp	Indirect	2
Mov @Rp,#num	Copy the immediate byte num to the address in Rp	Indirect	2
Movx A,@Rp	Copy the content of external add in Rp to acc	Indirect	1
Movx A,@DPTR	Copy the content of external add in DPTR to acc	Indirect	1
Movx @Rp,A	Copy the content of acc to the external add in Rp	Indirect	1

Movx @DPTR,A	Copy the content of acc to the external add in DPTR	Indirect	1
Movc A,@a+DPTR	The address is formed by adding acc and DPTR and its content is copied to acc	indirect	1
Movc A, @a+PC	The address is formed by adding acc and PC and its content is copied to acc	indirect	1
Push addr	Increment SP and copy the data from source add to internal RAM address contained in SP	Direct	2
Pop addr	copy the data from internal RAM address contained in SP to destination add and decrement SP	direct	2
Xch A, Rx	Exchange the data between acc and Rx	Register	1
Xch A, addr	Exchange the data between acc and given add	Direct	2
Xch A,@Rp	Exchange the data between acc and address in Rp	Indirect	1
Xchd A, @Rp	Exchange only lower nibble of acc and address in Rp	indirect	1

Arithmetic Instruction Set:

The arithmetic instructions perform the basic operations such as:

- Addition
- Multiplication
- Subtraction
- Division

MNEMONICS	DESCRIPTION	BYTES
ADD A,Rr	A+Rr-->A	1
ADD A,@Rp	A+(Rp)-->A	2
ADD A, #N	A+N-->A	1
SUB A,Rr	A-Rr-->A	2
SUB A,@Rp	A-(Rp)-->A	1
SUB A, #N	A-N-->A	2
DEC A	A-1-->A	1
DEC Rr	Rr-1-->	2
DEC ADD	(ADD)-1-->ADD	1
INC A	A+1-->A	2
INC Rr	Rr+1-->	1
INC ADD	(ADD)+1-->ADD	2
DIV AB	A/B-->AB	1
MUL AB	A*B-->AB	2

DAA- Decimal adjust after addition

Addition:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to the register R0//
MOV A, #05H      // move the value 5 to accumulator A//
ADD A, R0        // addA value with R0  value and stores the result inA//
END
```

Multiplication:

```
ORG 0000h
MOV A, #03H      // move the value 3 to the register A//
MOV B, #05H      // move the value 5 to accumulator B//
MUL AB        // Multiplied result is  stored in the Accumulator A //
END
```

Subtraction:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to register R0//
MOV A, #05H      // move the value 5 to accumulator A//
SUBB A, R0     // Result value is stored in the Accumulator A //
END
```

Division:

```
ORG 0000h
MOV A, #10H      // move the value 3 to register R0//
MOV B, #04H      // move the value 5 to accumulator A//
DIV AB      // final value is stored in the Accumulator A // reminder in B, Quotient in A
END
```

Logical Instruction Set:

Mnemonics	Description	Bytes
ANL A, Rr	A AND Rr-->A	1
ANL A, ADD	A AND(ADD)-->A	2
ORL A, Rr	A OR Rr-->A	1
ORL A, ADD	A OR(ADD)-->A	2
XRL A, Rr	A XOR Rr-->A	1
XRL A, ADD	A XOR(ADD)-->A	2
CLR A	00-->A	1
CPL A	A bar-->A	2

Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

ORL A, R0 //00100000/00000101=00000000//

Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

ANL A, R0

3. Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

XRL A, R0

The 8051 microcontroller consist **four shift operators**:

- RR —> Rotate Right
- RRC —>Rotate Right through carry
- RL —> Rotate Left
- RLC —>Rotate Left through carry

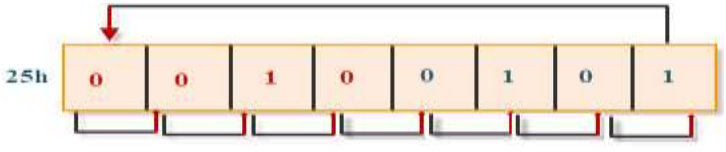
Rotate Right (RR):

In this shifting operation, the MSB becomes LSB and all bits shift towards right side bit-by-bit, serially.

Syntax:

MOV A, #25h

RRA



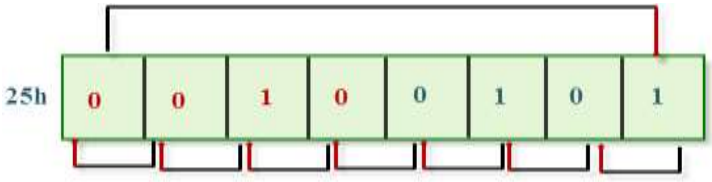
Rotate Left (RL):

In this shifting operation, the MSB becomes LSB and all bits shift towards Left side bit-by-bit, serially.

Syntax:

MOV A, #25h

RLA



RRC Rotate Right through Carry:

In this shifting operation, the LSB moves to carry and the carry becomes MSB, and all the bits are shift towards right side bit by bit position.

Syntax:

MOV A, #27h

RRC A

RLC Rotate Left through Carry:

In this shifting operation, the MSB moves to carry and the carry becomes LSB and all the bits shift towards left side in a bit-by-bit position.

Syntax:

MOV A, #27h

RLC A

Branch Instruction set

Conditional Instructions

The CPU executes the instructions based on the condition by checking the single bit status or byte status. The 8051microcontroller consists of various conditional instructions such as:

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and jump if register ≠ 0
CJNE A, data	Jump if A ≠ data
CJNE reg, #data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

Call and Jump Instructions:

The call and jump instructions are used to avoid the code replication of the program. When some specific code used more than once in different places in the program, if we mention specific name to code then we could use that name anywhere in the program without entering a code for every time. This reduces the complexity of the program.

CALL → ACALL & LCALL

The 8051 provides 2 forms for the CALL instruction:

- Absolute Call – ACALL
  - Uses an 11-bit address
  - The subroutine must be within the same 2K page.
- Long Call – LCALL
  - Uses a 16-bit address
  - The subroutine can be anywhere.

Both forms push the 16-bit address of PC on the stack and update the stack pointer.

LJMP (long jump)

LJMP is an unconditional long jump. It is a 3-byte instruction in which the first byte is the opcode, and the second and third bytes represent the 16-bit address of the target location.

SJMP (short jump)

In this 2-byte instruction, the first byte is the opcode and the second byte is the relative address of the target location.

AJMP

Function: Absolute Jump Within 2K Block

Syntax: AJMP *code address*

Loop Instructions:

The loop instructions are used to repeat the block each time while performing the increment and decrement operations. The 8051 microcontroller consist two types of loop instructions:

- CJNE —> compare and jump if not equal
- DJNZ —> decrement and jump if not zero

Compare and Jump if Not Equal – CJNE

- Compare the magnitude of the two operands and jump if they are not equal.
  - The values are considered to be unsigned.
  - The Carry flag is set / cleared appropriately.
  - CJNE A, direct, rel
  - CJNE A, #data, rel
  - CJNE Rn, #data, rel

Decrement and Jump if Not Zero – DJNZ

- Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.

```
DJNZ 20H,LOC1
DJNZ 30H,LOC2
DJNZ 40H,LOC3
```

- If internal RAM locations 20H, 30H and 40H contain the values 01H, 5FH and 16H respectively,
- the above instruction sequence will cause a jump to the instruction at LOC2, with the values 00H, 5EH, and 15H in the 3 RAM locations

Return instructions: RET and RETI(Return from subroutine and Return from interrupt service routine)

BIT-Oriented Instructions

Mnemonic	Description	Byte	Cycle
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

Simple Programs using 8051

Write A Program To Add Two 8 Bit Number And Store The Result At External Memory Location 2050H.

```
ORG 0000H
 MOV A, #05H
 MOV R, #09H
ADD A, R1
MOV DPTR, #2050H      // Initialize DPTR with 2050 address
MOVX @ DPTR, A        // Move content of A into 2050. We can see output in 2050 location
```

```
AGAIN: SJMP AGAIN                // this loop work infinitely, indicates end of the program
```

INPUT:

A=05H

R1=09H

OUTPUT:

2050 = 0EH..

Subtraction of two 8 bit numbers

```
        ORG 4100
        MOV A,#data1
        SUBB A,#data2
        MOV DPTR,#4500
        MOVX   @DPTR,A    // X   indicate external memory
        HERE: SJMP HERE
```

Input: 66

23

Output: 43 (4500)

Assembly program for Simple 2 16bit number addition using registers

```
    ORG 0000H
    MOV R7, #34H          // SAY R7 HAS LOWER BYTE 34
    MOV R6, #24H          // R6 HAS HIGHER BYTE 24
                          // MAKING NUMBER 2434

    MOV R5,#45H           // ANOTHER NUMBER SAY 3345
    MOV R4, #33H          //WITH R5 LOWER BYTE 45
                          // R4 HIGHER BYTE 33
    MOV A, R5
    ADD A , R7
    MOV R3, A
    MOV A, R6
    ADDC A, R4
    MOV R2,A
    END
```

Multiplication of two 8 bit numbers

```
MOV DPTR,#8600H      // initialize DPTR with 8600
```

```
MOVX A,@DPTR            // Move content of DPTR into A
MOV B,A
MOV DPTR,#8601H         // Read the second operand from 8601 location
MOVX A,@DPTR
MUL AB
MOV DPTR,#8701H         // Initialize DPTR with 8701 for storing output data
MOVX @DPTR,A
MOV DPTR,#8700H         //  The least significant byte of the result is placed in the Accumulator and the
// most significant-byte is placed in the "B" register. So we need to display from B
MOV A,B
MOVX @DPTR,A
E: SJMP E
```

Factorial of a number using subroutine

```
ORG 0000
MOV R1,#05
MOV A, R1
LCALL FACT
E: SJMP E
        FACT:
            CJNE R1,#00,UP      // compare R1 and 00 if it is not equal go to UP
        RET
        UP: DEC R1              // Decrement R1 by one ie 4
        MOV B, R1
        MUL AB
        LJMP FACT
```

Sorting of n numbers

```
ORG 0000H
MOV R7,#4
loop1:MOV R0,#40H
  MOV R6,#04
loop:MOV A,@R0
 INC R0
 MOV 50H,@R0
 CJNE A, 50H, next
 SJMP down
next:JC down
```

```
  MOV @R0,A
  DEC R0
  MOV @R0,50H
down:DJNZ R6,loop
  DJNZ R7,loop1
END
```

To add n bytes stored in external RAM(Sum of n numbers)

```
        MOV R0, #0A
        MOV R1, #00
        MOV DPTR, #9000
        Loop: MOVX A, @DPTR
        ADD A, R1
        MOV R1, A
        INC DPTR
        DJNZ R0, LOOP
```

Fibonacci series (Refer Page 87 in solved Text book)

```
 MOV R1, 30 H
MOV R7,#40H
MOV @R7, #00H
INC R7
MOV @R7, #01H
MOV R5, #42H
DEC R1
DEC R1
DEC R7
LOOP: MOV A, @R7
        INC R7
        ADD A, @R7
        MOV @R5,A
        INC R5
        DJNZ R1, LOOP
STOP: SJMP STOP
```