DB
Directive used by assembler to store the permanent data into the program memory (during the burning process)

ORG 100h
DB 50h, 90h

Program M

0100h
0101h

50h
80h
The Stack

The **stack** is part of the RAM

In 8051, all RAM locations can be as stack

The Stack Pointer (SP)

The **SP** is one of the SFR

It has initial value "07h"

It points toward the top of the stack area
The Stack is affected by

1- Push & Pop instructions
2- Call & Ret instructions
3- the interrupts (automatic call)
4- Retl instructions (ret from interrupt)
The data transfers from/to Stack

Last input is first output
Pushing data-byte into the stack

PUSH 60h

Source address

Initial values

After PUSH
Pushing other data-byte into the stack

PUSH 80h

After PUSH 80h
After Pop data-byte from the stack

Before POP

After POP 00H

POP 00H
The execution of instruction “LCALL” on the stack (2 slides)
Program memory

Before CALL

R0

SP

After CALL

R0

SP

RAM

FFH

Before CALL

FFH

After CALL

Pushing return address

PC → 0000H
0001H
0002H
0003H

PC → 7711H

LCALL 77h
11h

3 Bytes

0000H

7711H

7740H

RET
After RET
The instruction types of MCS-51

The ISA of MCS-51 family includes **5 types** of instructions as following:

a) **Data transfer** Instructions.
b) **Logical operation** instructions.
c) **Bit-Oriented** Instructions.
d) **Arithmetic operation** Instructions.
e) **Branching** Instructions.
Each instruction is known by the shortened name of its operation and called “Mnemonic” alike:

- **Move data**: MOV
- **Increment data**: INC
- **Adding 2 data**: ADD
- **Division 2 data**: DIV
- **No operation**: NOP
- **Jump if carry is set**: JC
- **Complement**: CPL
## some terminologies of ISA for MCS-51

<table>
<thead>
<tr>
<th>Instruction’s terminology</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The Accumulator</td>
<td>CPL A</td>
</tr>
<tr>
<td>C</td>
<td>The Carry flag</td>
<td>CLR C</td>
</tr>
<tr>
<td>Bit</td>
<td>The addressable bit</td>
<td>SETB 00H</td>
</tr>
<tr>
<td>Rn</td>
<td>The Register R0 to R7 of the selected bank</td>
<td>DEC R5</td>
</tr>
<tr>
<td>Rx</td>
<td>Address of any internal RAM location (from 00h to FFh) or (from 0 to 255) or the SFR’s names</td>
<td>MOV PSW, 20H</td>
</tr>
<tr>
<td>@Ri</td>
<td>Indirect 8-bit address of internal or external RAM location (i = 1 or 0 only)</td>
<td>INC @R0, DEC @R1</td>
</tr>
<tr>
<td>Instruction's terminology</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Rel</td>
<td>The relative address in the program memory within range (from –128 to +127) bytes</td>
<td>Again: SJMP again</td>
</tr>
<tr>
<td>#data</td>
<td>The 8-bit immediate data (from 00h to FFh)</td>
<td>MOV A, #50h</td>
</tr>
<tr>
<td>#data</td>
<td>The immediate 16-bit data (from 0000h to FFFFh)</td>
<td>MOV DPTR, #1020h</td>
</tr>
<tr>
<td>Addr</td>
<td>The 16-bit branching address within 64 KB</td>
<td>LJMP Label 3</td>
</tr>
<tr>
<td>Addr</td>
<td>The 11-bit branching address within 2 KB</td>
<td>Start: AJMP Start</td>
</tr>
</tbody>
</table>

- SJMP: Jump to another location in the program.
- MOV: Move data from one location to another.
- LJMP: Jump to a label.
- AJMP: Jump to another location in the program, where the address is 128 bytes away from the current location.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV @Ri, #data</td>
<td>Move 8-bit data → internal RAM (indirectly[Ri])</td>
</tr>
<tr>
<td>MOV @Ri, A</td>
<td>Move data from A → internal RAM (indirectly[Ri])</td>
</tr>
<tr>
<td>MOV @Ri, Rx</td>
<td>Move data from internal RAM → internal RAM (indirectly[Ri])</td>
</tr>
<tr>
<td>MOV A, #data</td>
<td>Move 8-bit data → A</td>
</tr>
<tr>
<td>MOV A, @Ri</td>
<td>Move data from internal RAM (indirectly[Ri]) → A</td>
</tr>
<tr>
<td>MOV A, Rx</td>
<td>Move data from internal RAM → A</td>
</tr>
<tr>
<td>MOV A, Rn</td>
<td>Move data from (R0...R7) → A</td>
</tr>
<tr>
<td>MOV Rx, #data</td>
<td>Move 8-bit data → internal RAM</td>
</tr>
<tr>
<td>MOV Rx, @Ri</td>
<td>Move data from internal RAM (indirectly[Ri]) → internal RAM</td>
</tr>
<tr>
<td>MOV Rx, A</td>
<td>Move data from A → internal RAM</td>
</tr>
<tr>
<td>MOV Rx, Rx</td>
<td>Move data from internal RAM → internal RAM</td>
</tr>
<tr>
<td>MOV Rx, Rn</td>
<td>Move data from (R0...R7) → internal RAM</td>
</tr>
<tr>
<td>MOV DPTR, #data16</td>
<td>Move 16-bit data → DPTR</td>
</tr>
<tr>
<td>MOV Rn, #data</td>
<td>Move 8-bit data → (R0...R7)</td>
</tr>
<tr>
<td>MOV Rn, A</td>
<td>Move data from A → (R0...R7)</td>
</tr>
<tr>
<td>MOV Rn, Rx</td>
<td>Move data from internal RAM → (R0...R7)</td>
</tr>
<tr>
<td>MOVCA, @A+DPTR</td>
<td>Move data from Flash-M (indirectly [A+DPTR]) → A</td>
</tr>
<tr>
<td>MOVCA, @A+PC</td>
<td>Move data from Flash-M (indirectly [A+PC]) → A</td>
</tr>
<tr>
<td>MOVX A, @DPTR</td>
<td>Move data from external RAM (indirectly[DPTR]) → A</td>
</tr>
<tr>
<td>MOVX A, @Ri</td>
<td>Move data from external RAM (indirectly[Ri]) → A</td>
</tr>
<tr>
<td>MOVX @DPTR, A</td>
<td>Move data from A → external RAM (indirectly[DPTR])</td>
</tr>
<tr>
<td>MOVX @Ri, A</td>
<td>Move data from A → external RAM (indirectly[Ri])</td>
</tr>
</tbody>
</table>
Available MOV-instructions

The available operands of "MOV"

Invalid data-movement instructions
MOV R1, 8h

Before execution “MOV R1, 8H”

After execution “MOV R1, 8H”

MOV P3, A

Before execution “MOV P3, A”

After execution “MOV P3, A”
MOV SCON, #50h

Before and after the execution of the instruction “MOV SCON, #50H”
MOV A, @R1

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>70H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AEh</td>
</tr>
<tr>
<td>70H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = E0H</td>
<td>13h</td>
<td></td>
</tr>
<tr>
<td>FFH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before and after the execution "MOV A, @R1"
Draw Memory block-diagram and write assembly sub-program to read 2 samples (digital data) from a digital sensor via port 2 and store them into 2 successive internal RAM locations starting from address 77h using indirect addressing @Ri of MCS-51.
The subprogram

Using simulator “prog-studio”

INCLUDE 8051.MC

MOV P2,#FFH ; Setting port 2 as input
MOV R0,#77H ; Loading R0 by 77h
MOV @R0, P2 ; Moving 1st data to R77

INC R0 ; incrementing R0
MOV @R0, P2 ; Moving 2nd data to R78
The 16-bit Data transfer

Example → MOV DPTR, #EE44h