Unit 2:
Instruction Set

8051 Microcontroller Programs in Assembly Language

The assembly language is made up of elements which all are used to write the program in sequential manner. Follow the given rules to write programming in assembly language.

Rules of Assembly Language

The assembly code must be written in upper case letters

The labels must be followed by a colon (label:)

All symbols and labels must begin with a letter

All comments are typed in lower case

The last line of the program must be the END directive

The assembly language mnemonics are in the form of op-code, such as MOV, ADD, JMP, and so on, which are used to perform the operations.

Op-code: The op-code is a single instruction that can be executed by the CPU. Here the op-code is a MOV instruction.

Operands: The operands are a single piece of data that can be operated by the op-code. Example, multiplication operation is performed by the operands that are multiplied by the operand.

Syntax: MUL a,b;

The Elements of an Assembly Language Programming:

- Assembler Directives
- Instruction Set
- Addressing Modes

Assembler Directives:

The assembling directives give the directions to the CPU. The 8051 microcontroller consists of various kinds of assembly directives to give the direction to the control unit. The most useful directives are 8051 programming, such as:

- ORG (Origin)
- DB (define byte)
- EQU (equivalent)
- END

ORG(origin): This directive indicates the start of the program. This is used to set the register address during assembly. For example; ORG 0000h tells the compiler all subsequent code starting at address 0000h.

Syntax: ORG 0000h ;Tells the Assembler to assemble the next statement at 0000H

DB(define byte): The define byte is used to allow a string of bytes. For example, print the "EDGEFX" wherein each character is taken by the address and finally prints the "string" by the DB directly with double quotes.

Syntax:

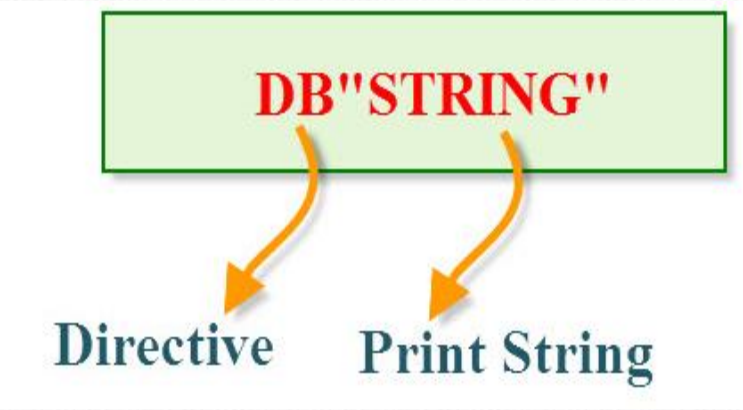
ORG 0000h

MOV a, #00h

————————

————————

DB"EDGEFX"



EQU (equivalent): The equivalent directive is used to equate address of the variable.

Syntax:

reg equ,09h

MOV reg,#2h

END: The END directive is used to indicate the end of the program.

Syntax:

reg equ,09h

MOV reg,#2h

END

Example :

The following example demonstrates the addition of two numbers and then the storage of final value in the Bank0 register using an assembly level program.

```
Org 0000h
MOV PSW,#00h     //PSW=program status word (open
the bank0 memory)
MOV A, 15
ADD A, 20
MOV 00h, A
END
```

Assembly program to move 6 natural numbers in bank0 register R0-R5

```
ORG 0000h (starting addresses declaration)
MOV PSW, #00h (open the bank0 memory)
MOV r0, #00h (starting address of bank0 memory)
MOV r1, #01h
MOV r2, #02h
MOV r3, #03h
MOV r4, #04h
MOV r5, #05h
END
```

Assembly program to move 6 natural numbers in bank1 register R0-R7

```
Org 0000h (starting addresses declaration)
MOV PSW, #08h (open the bank1 memory)
MOV r0, 00h (value send to the bank1 memory)
MOV r1, 01h
MOV r2, 02h
MOV r3, 03h
MOV r4, 05h
MOV r5, 06h
END
```

The Assembly program for subtraction used with an Accumulator

```
Org 0000h
MOV A, #03h (1byte data)
MOV B, #05h
SUBB A, B
END
```

The Assembly program for multiplication used with a B-Register

```
Org 0000h
MOV A, #09h
MOV B, #03h
MUL A, B (Final value stored in A)
END
```

The Assembly program for Division used with a B-Register

```
Org 0000h
MOV A, #09h
MOV B, #03h
DIC A, B (Final value stored in A)
END
```

Addressing Modes:

- Addressing mode is a way to address an operand.
- Operand means the data we are operating upon (in most cases source data).
- It can be a direct address of memory, it can be register names, it can be any numerical data etc.

MOV A,#6AH

- Here the data 6A is the operand, often known as source data.
- When this instruction is executed, the data 6AH is moved to accumulator A.

In 8051 There are six types of addressing modes.

- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Indexed Addressing Mode
- Implied Addressing Mode

Immediate Addressing Mode

Let's begin with an example.

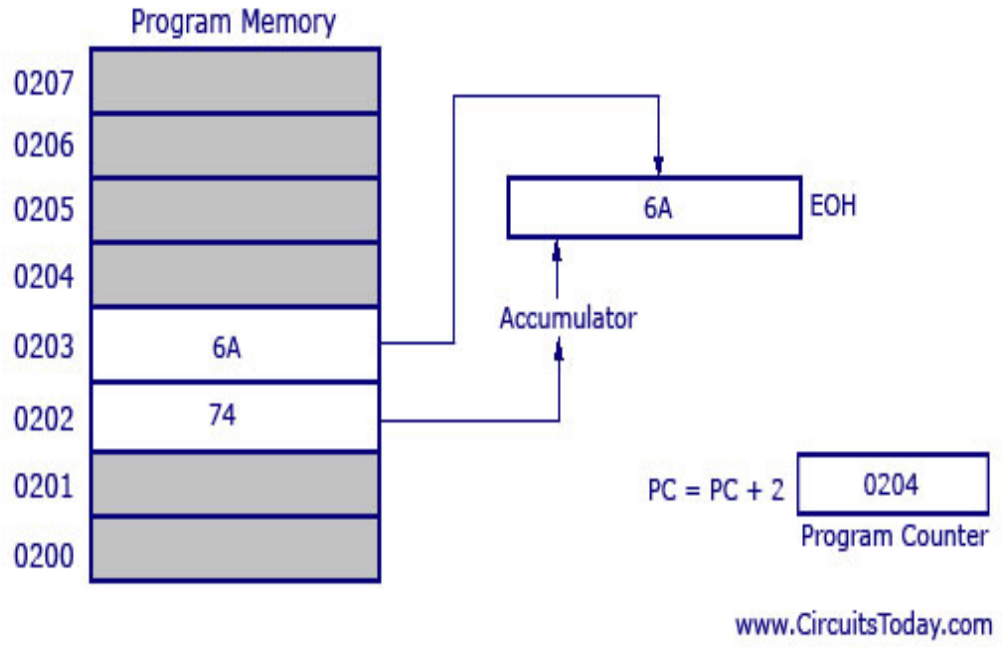
MOV A, #6AH

In general we can write MOV A, #data

This addressing mode is named as "immediate" because it transfers an 8-bit data immediately to the accumulator (destination operand).

Immediate Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1



- The picture above describes the instruction and its execution.
- The opcode for MOV A, # data is 74H.
- The opcode is saved in program memory at 0202 address.
- The data 6AH is saved in program memory 0203. (Any part of the program memory can be used, this is just an example)
- When the opcode 74H is read, the next step taken would be to transfer whatever data at the next program memory address (here at 0203) to accumulator A (E0H is the address of accumulator).
- This instruction is of two bytes and is executed in one cycle.
- So after the execution of this instruction, program counter will add 2 and move to o204 of program memory.

Direct Addressing Mode

This is another way of addressing an operand. Here the address of the data (source data) is given as operand. Lets take an example.

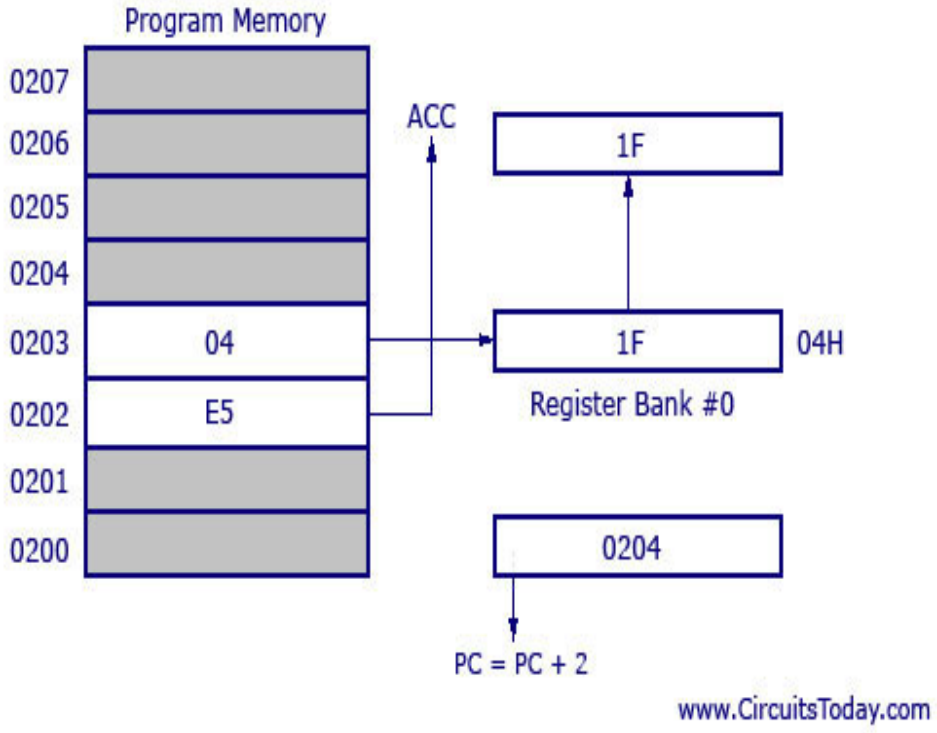
MOV A, 04H

Here 04H is the address of register 4 of register bank#0. When this instruction is executed, what ever data is stored in register 04H is moved to accumulator. In the picture below we can see, register 04H holds the data 1FH. So the data 1FH is moved to accumulator.

Note: We have not used **'#'** in direct addressing mode, unlike immediate mode. If we had used **'#'**, the data value 04H would have been transferred to accumulator instead of 1FH.

Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #04H	E5	2	1



- As shown in picture above this is a 2 byte instruction which requires 1 cycle to complete.
- Program counter will increment by 2 and stand in 0204.
- The opcode for instruction ***MOV A, address*** is E5H.
- When the instruction at 0202 is executed (E5H), accumulator is made active and ready to receive data.
- Then program control goes to next address that is 0203 and look up the address of the location (04H) where the source data (to be transferred to accumulator) is located.
- At 04H the control finds the data 1F and transfers it to accumulator and hence the execution is completed.

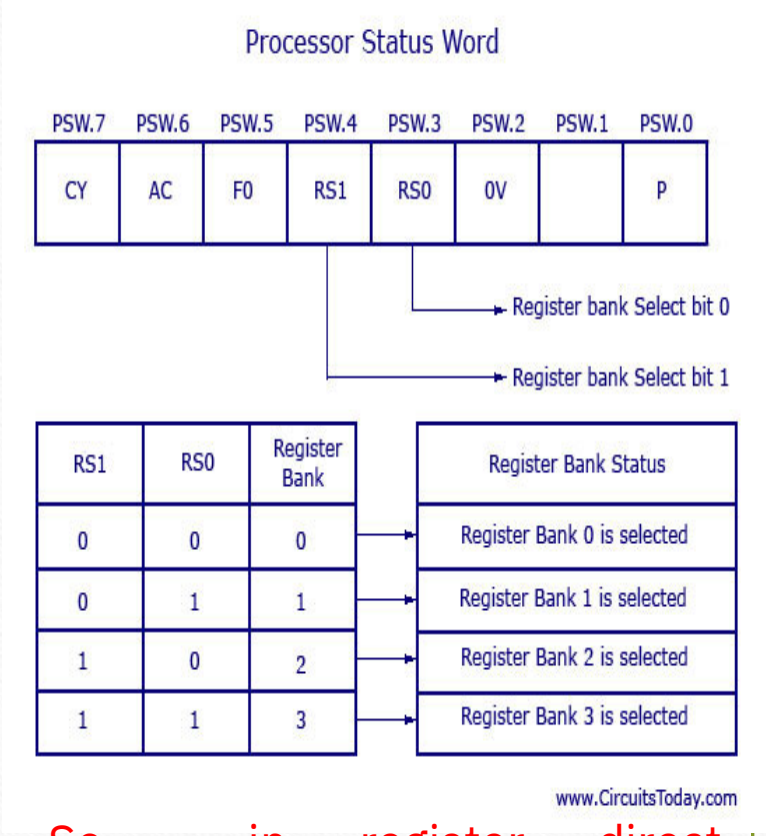
Register Direct Addressing Mode

In this addressing mode we use the register name directly (as source operand). An example is shown below.

MOV A,B

MOV A, R4

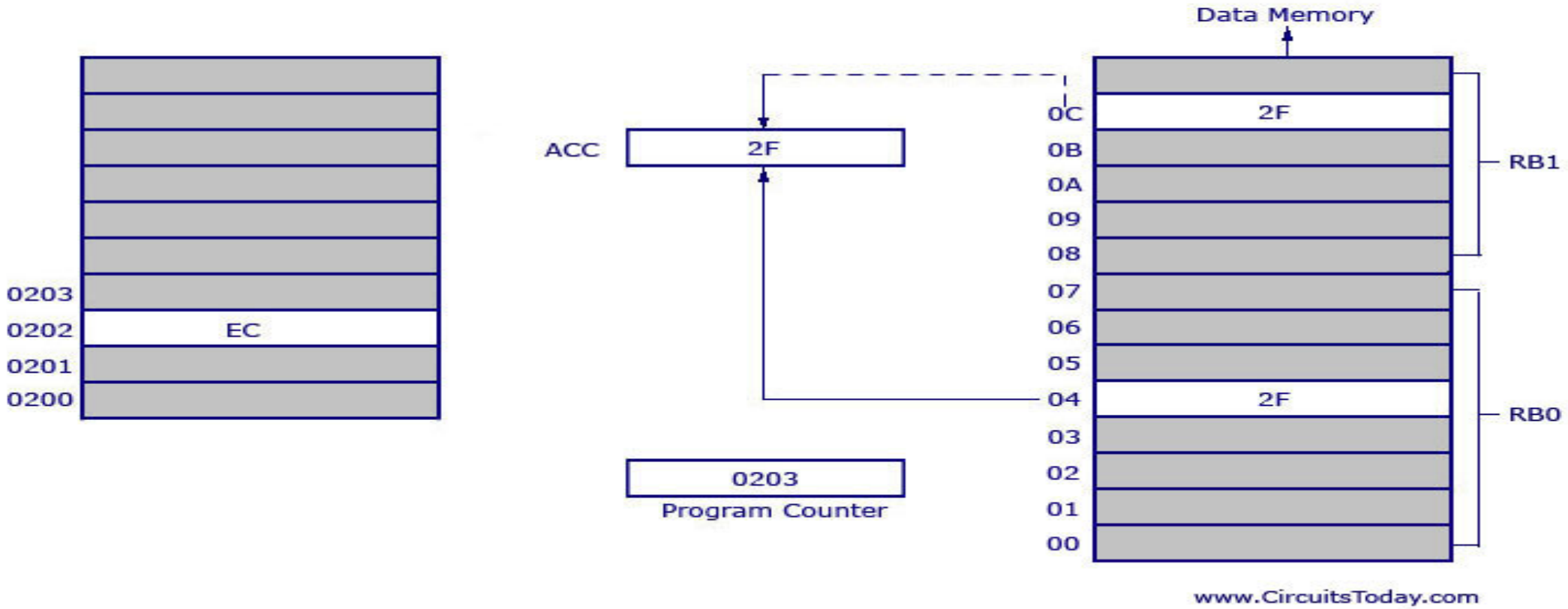
- At a time registers can take value from R0,R1…to R7. You may already know there are 32 such registers. So how you access 32 registers with just 8 variables to address registers?
- Here comes the use of register banks.
- There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7.
- At a time only one register bank can be selected. Selection of register bank is made possible through a Special Function Register (SFR) named Processor Status Word (PSW).
- PSW is an 8 bit SFR where each bit can be programmed. Bits are designated from PSW.0 to PSW.7
- Register banks are selected using PSW.3 and PSW.4 These two bits are known as register bank select bits as they are used to select register banks. A picture below shows the PSW register and the Register Bank Select bits with status.



So in register direct addressing mode, data is transferred to accumulator from the register (based on which register bank is selected).

Register Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



- So we see that opcode for MOV A, R4 is EC.
- The opcode is stored in program memory address 0202 and when it is executed the control goes directly to R4 of the respected register bank (that is selected in PSW).
- If register bank #0 is selected then the data from R4 of register bank #0 will be moved to accumulator. (Here it is 2F stored at 04 H).
- 04 H is the address of R4 of register bank #0. Movement of data (2F) in this case is shown as bold line.
- At the dotted line, 2F is getting transferred to accumulator from data memory location 0C H. Now understand that 0C H is the address location of Register 4 (R4) of register bank #1.
- Programmers usually get confused with register bank selection. Also keep in mind that data at R4 of register bank #0 and register bank #1 (or even other banks) will not be same. So wrong selection of register banks will result in undesired output.

Also note that the instruction above is 1 byte and requires 1 cycle for complete execution. This means using register direct addressing mode can save program memory.

Register Indirect Addressing Mode

So in this addressing mode, address of the data (source data to transfer) is given in the register operand.

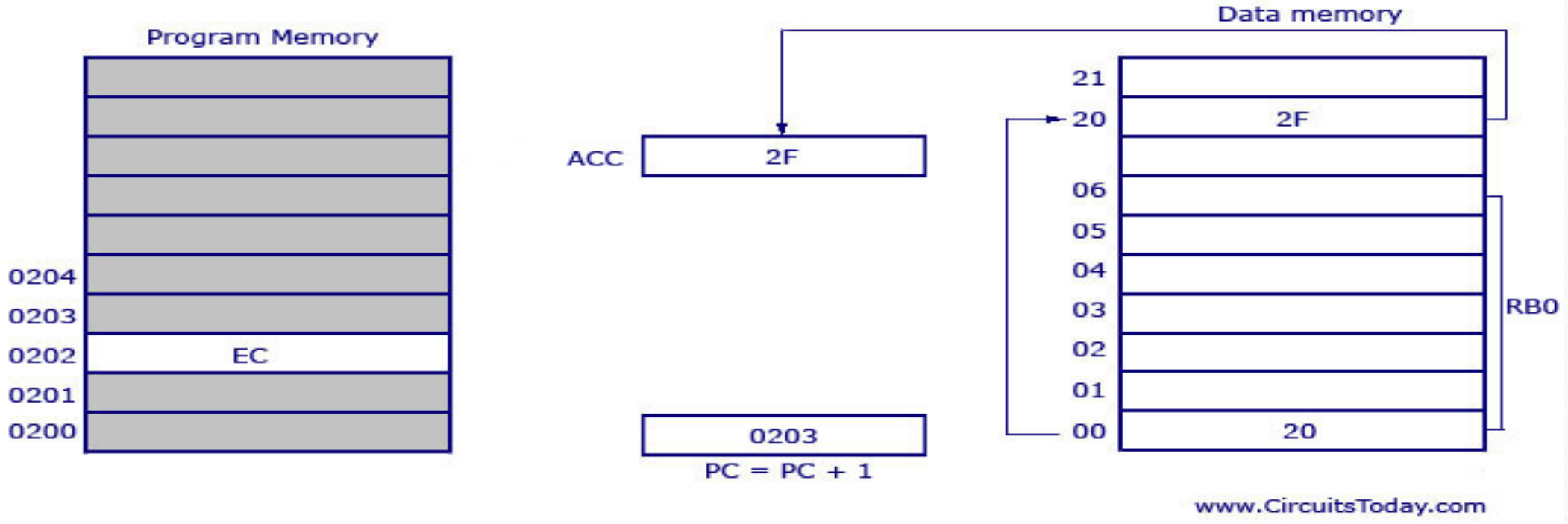
MOV A, @R0

Here the value inside R0 is considered as an address, which holds the data to be transferred to accumulator. The @ symbol indicates that the addressing mode is indirect

Example: If R0 holds the value 20H, and we have a data 2F H stored at the address 20H, then the value 2FH will get transferred to accumulator after executing this instruction.

Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



- So the opcode for **MOV A, @R0** is E6H.
- Assuming that register bank #0 is selected.
- So the R0 of register bank #0 holds the data 20H.
- Program control moves to 20H where it locates the data 2FH and it transfers 2FH to accumulator.
- This is a single byte instruction and the program counter increments 1 and moves to 0203 of program memory.

Note: Only R0 and R1 are allowed to form a register indirect addressing instruction. In other words programmer can must make any instruction either using @R0 or @R1.

Indexed Addressing Mode

Well lets see two examples first.

MOVC A, @A+DPTR and MOVC A, @A+PC

Where, MOVC is move code, DPTR is data pointer and PC is program counter (both are 16 bit registers).

Lets take the first example.

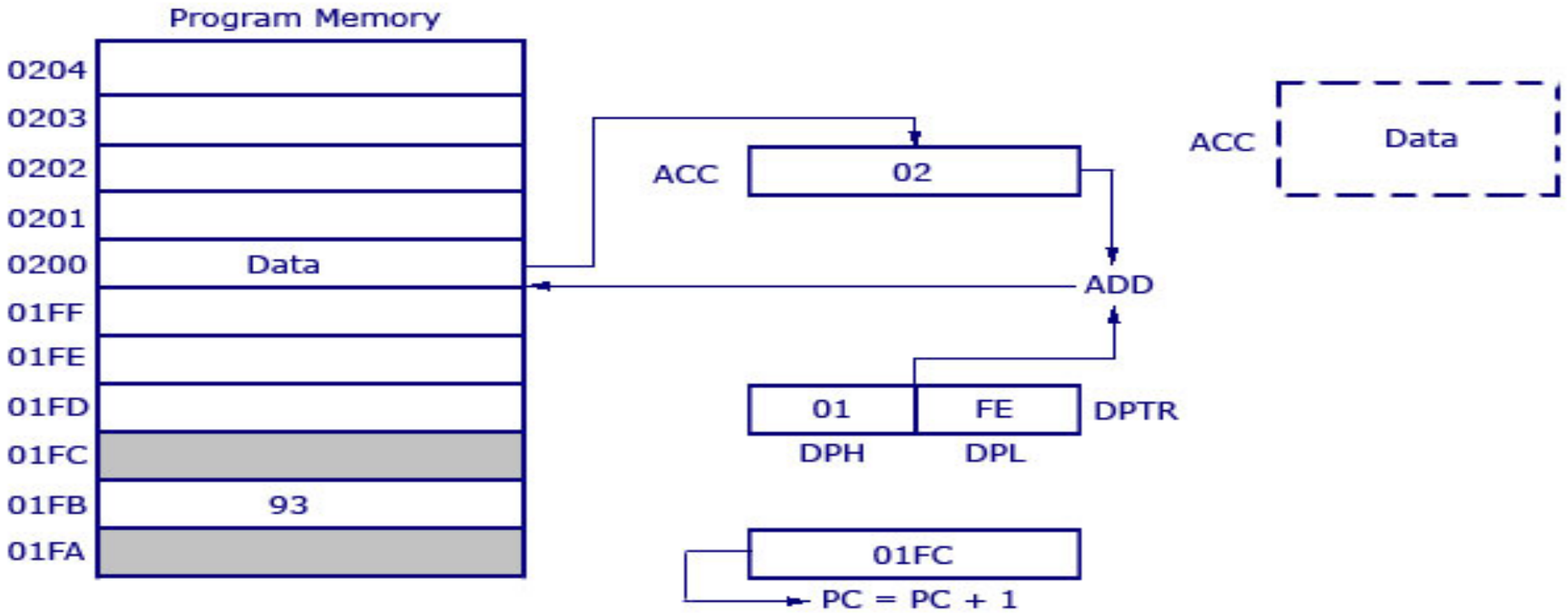
MOVC A, @A+DPTR

What's the first impression you have now?

The source operand is @A+DPTR and we know we will get the source data (to transfer) from this location. It is nothing but adding contents of DPTR with present content of accumulator. This addition will result a new data which is taken as the address of source data (to transfer). The data at this address is then transferred to accumulator. Take a look at the picture below.

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2



- The opcode for the instruction is 93H.
- DPTR holds the value 01FE, where 01 is located in DPH (higher 8 bits) and FE is located in DPL (lower 8 bits).
- Accumulator now has the value 02H.
- A 16 bit addition is performed and now 01FE H+02 H results in 0200 H.
- What ever data is in 0200 H will get transferred to accumulator.
- The previous value inside accumulator (02H) will get replaced with new data from 0200H. ***New data in the accumulator is shown in dotted line box***.

- This is a 1 byte instruction **with 2 cycles needed for execution**. What you infer from that? The execution time required for this instruction is high compared to previous instructions (which all were 1 cycle).
- The other example **MOVC A, @A+PC** works the same way as above example. The only difference is, instead of adding DPTR with accumulator, here data inside program counter (PC) is added with accumulator to obtain the target address.

Data Transfer instructions

List of data transfer instructions

- Data transfer instructions are responsible for transferring data between various memory storing elements like registers, RAM, and ROM.
- The execution time of these instructions varies based on how complex an operation they have to perform.
- In the table given below, all the data transfer instructions are listed.
- In the table
- [A]= Accumulator; [Rn]=Register in RAM; DPTR=Data Pointer; PC=Program Counter

The Data Transfer Instructions are associated with transfer of data between registers or external program memory or external data memory. The Mnemonics associated with Data Transfer are given below.

Mnemonic	**Description**
MOV	Move Data
MOVC	Move Code
MOCX	Move External Data
PUSH	Move Data to Stack
POP	Copy Data from Stack
XCH	Exchange Data between two Registers
XCHD	Exchange Lower Order Data between two Registers

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
MOV	A, #Data	A ← Data	Immediate	2	1
	A, Rn	A ← Rn	Register	1	1
	A, Direct	A ← (Direct)	Direct	2	1
	A, @Ri	A ← @Ri	Indirect	1	1
	Rn, #Data	Rn ← data	Immediate	2	1
	Rn, A	Rn ← A	Register	1	1
	Rn, Direct	Rn ← (Direct)	Direct	2	2
	Direct, A	(Direct) ← A	Direct	2	1
	Direct, Rn	(Direct) ← Rn	Direct	2	2
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct	3	2
	Direct, @Ri	(Direct) ← @Ri	Indirect	2	2
	Direct, #Data	(Direct) ← #Data	Direct	3	2
	@Ri, A	@Ri ← A	Indirect	1	1
	@Ri, Direct	@Ri ← Direct	Indirect	2	2
	@Ri, #Data	@Ri ← #Data	Indirect	2	1
	DPTR, #Data16	DPTR ← #Data16	Immediate	3	2
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed	1	2
	A, @A+PC	A ← Code Pointed by A+PC	Indexed	1	2
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect	1	2
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect	1	2
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect	1	2
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect	1	2
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct	2	2
POP	Direct	(Direct) ← Stack Pointer SP	Direct	2	2
XCH	Rn	Exchange ACC with Rn	Register	1	1
	Direct	Exchange ACC with Direct Byte	Direct	2	1
	@Ri	Exchange ACC with Indirect RAM	Indirect	1	1
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect	1	1

ELECTRONICS HUB

Arithmetic Instructions

Using Arithmetic Instructions, you can perform addition, subtraction, multiplication and division. The arithmetic instructions also include increment by one, decrement by one and a special instruction called Decimal Adjust Accumulator.

Mnemonic	Description
ADD	Addition without Carry
ADDC	Addition with Carry
SUBB	Subtract with Carry
INC	Increment by 1
DEC	Decrement by 1
MUL	Multiply
DIV	Divide

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	A ← A + Data	Immediate	2	1
	A, Rn	A ← A + Rn	Register	1	1
	A, Direct	A ← A + (Direct)	Direct	2	1
	A, @Ri	A ← A + @Ri	Indirect	1	1
ADDC	A, #Data	A ← A + Data + C	Immediate	2	1
	A, Rn	A ← A + Rn + C	Register	1	1
	A, Direct	A ← A + (Direct) + C	Direct	2	1
	A, @Ri	A ← A + @Ri + C	Indirect	1	1
SUBB	A, #Data	A ← A – Data – C	Immediate	2	1
	A, Rn	A ← A – Rn – C	Register	1	1
	A, Direct	A ← A – (Direct) – C	Direct	2	1
	A, @Ri	A ← A – @Ri – C	Indirect	1	1
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--	1	4
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--	1	4
DEC	A	A ← A – 1	Register	1	1
	Rn	Rn ← Rn – 1	Register	1	1
	Direct	(Direct) ← (Direct) – 1	Direct	2	1
	@Ri	@Ri ← @Ri – 1	Indirect	1	1
INC	A	A ← A + 1	Register	1	1
	Rn	Rn ← Rn + 1	Register	1	1
	Direct	(Direct) ← (Direct) + 1	Direct	2	1
	@Ri	@Ri ← @Ri + 1	Indirect	1	1
	DPTR	DPTR ← DPTR + 1	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

ELECTRONICS HUB

Logical Instructions

The next group of instructions are the Logical Instructions, which perform logical operations like AND, OR, XOR, NOT, Rotate, Clear and Swap. Logical Instruction are performed on Bytes of data on a bit-by-bit basis.

Mnemonic	Description
ANL	Logical AND
ORL	Logical OR
XRL	Ex-OR
CLR	Clear Register
CPL	Complement the Register
RL	Rotate a Byte to Left
RLC	Rotate a Byte and Carry Bit to Left
RR	Rotate a Byte to Right
RRC	Rotate a Byte and Carry Bit to Right
SWAP	Exchange lower and higher nibbles in a Byte

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ANL	A, #Data	A ← A AND Data	Immediate	2	1
	A, Rn	A ← A AND Rn	Register	1	1
	A, Direct	A ← A AND (Direct)	Direct	2	1
	A, @Ri	A ← A AND @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) AND A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct	3	2
ORL	A, #Data	A ← A OR Data	Immediate	2	1
	A, Rn	A ← A OR Rn	Register	1	1
	A, Direct	A ← A OR (Direct)	Direct	2	1
	A, @Ri	A ← A OR @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) OR A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct	3	2
XRL	A, #Data	A ← A XRL Data	Immediate	2	1
	A, Rn	A ← A XRL Rn	Register	1	1
	A, Direct	A ← A XRL (Direct)	Direct	2	1
	A, @Ri	A ← A XRL @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) XRL A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct	3	2
CLR	A	A← 00H	--	1	1
CPL	A	A ← A	--	1	1
RL	A	Rotate ACC Left	--	1	1
RLC	A	Rotate ACC Left through Carry	--	1	1
RR	A	Rotate ACC Right	--	1	1
RRC	A	Rotate ACC Right through Carry	--	1	1
SWAP	A	Swap Nibbles within ACC	--	1	1

ELECTRONICS HUB

Boolean or Bit Manipulation Instructions

As the name suggests, Boolean or Bit Manipulation Instructions deal with bit variables. We know that there is a special bit-addressable area in the RAM and some of the Special Function Registers (SFRs) are also bit addressable.

Mnemonic	Description
CLR	Clear a Bit (Reset to 0)
SETB	Set a Bit (Set to 1)
MOV	Move a Bit
JC	Jump if Carry Flag is Set
JNC	Jump if Carry Flag is Not Set
JB	Jump if specified Bit is Set
JNB	Jump if specified Bit is Not Set
JBC	Jump if specified Bit is Set and also clear the Bit
ANL	Bitwise AND
ORL	Bitwise OR
CPL	Complement the Bit

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
CLR	C	C ← 0 (C = Carry Bit)	1	1
	Bit	Bit ← 0 (Bit = Direct Bit)	2	1
SET	C	C ← 1	1	1
	Bit	Bit ← 1	2	1
CPL	C	C ← $\overline{C}$	1	1
	Bit	Bit ← $\overline{Bit}$	2	1
ANL	C, /Bit	C ← C. $\overline{Bit}$ (AND)	2	1
	C, Bit	C ← C . Bit (AND)	2	1
ORL	C, /Bit	C ← C + $\overline{Bit}$ (OR)	2	1
	C, Bit	C ← C + Bit (OR)	2	1
MOV	C, Bit	C ← Bit	2	1
	Bit, C	Bit ← C	2	2
JC	rel	Jump is Carry (C) is Set	2	2
JNC	rel	Jump is Carry (C) is Not Set	2	2
JB	Bit, rel	Jump is Direct Bit is Set	3	2
JNB	Bit, rel	Jump is Direct Bit is Not Set	3	2
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit	3	2

ELECTRONICS HUB

Program Branching Instructions

The last group of instructions in the 8051 Microcontroller Instruction Set are the Program Branching Instructions. These instructions control the flow of program logic. The mnemonics of the Program Branching Instructions are as follows.

Mnemonic	Description
LJMP	Long Jump (Unconditional)
AJMP	Absolute Jump (Unconditional)
SJMP	Short Jump (Unconditional)
JZ	Jump if A is equal to 0
JNZ	Jump if A is not equal to 0
CJNE	Compare and Jump if Not Equal
DJNZ	Decrement and Jump if Not Zero
NOP	No Operation
LCALL	Long Call to Subroutine
ACALL	Absolute Call to Subroutine (Unconditional)
RET	Return from Subroutine
RETI	Return from Interrupt
JMP	Jump to an Address (Unconditional)

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC	2	2
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC	3	2
RET	--	Return from Subroutine (SP) → PC	1	2
RETI	--	Return from Interrupt	1	2
AJMP	ADDR11	Absolute Jump ADDR11 → PC	2	2
LJMP	ADDR16	Long Jump ADDR16 → PC	3	2
SJMP	rel	Short Jump PC + 2 + rel → PC	2	2
JMP	@A + DPTR	A + DPTR → PC	1	2
JZ	rel	If A=0, Jump to PC + rel	2	2
JNZ	rel	If A ≠ 0, Jump to PC + rel		
CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal	3	2
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal	3	2
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal	3	2
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal	3	2
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero	2	2
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero	3	2
NOP		No Operation	1	1

ELECTRONICS HUB

Example 1: Data transfer

```
MOV PSW, #00h
MOV A,#55H ;load value 55H into reg. A
MOV RO,A ;copy contents of A into RO
         ; (now A=RO=55H)
MOV Rl,A ;copy contents of A into Rl
         ;(now A=RO=Rl=55H)
MOV R2,A ;copy contents of A into R2
         ;now A=RO=Rl=R2=55H)
MOV R3,#95H ;load value 95H into R3
            ; (now R3=95H)
MOV A,R3 ;copy contents of R3 into A
         ;now A=R3=95H)
END
```

Example 2: Data transfer

```
MOV PSW, #00h
MOV A,#23H ;load 23H into A (A=23H)
MOV RO,#12H ;load 12H into RO (RO=12H)
MOV Rl,#lFH ;load 1FH into Rl (Rl=lFH)
MOV R2,#2BH ;load 2BH into R2 (R2=2BH)
MOV B,#3CH ;load 3CH into B (B=3CH)
MOV R7,#9DH ;load 9DH into R7 (R7=9DH)
MOV R5, #OF9H ;load F9H into R5 (R5=F9H)
MOV R6,#12 ;load 12 decimal (OCH)
           ;into reg. R6 (R6=OCH)
END
```

Example 3: Addition

```
MOV PSW, #00h
MOV A,#25H          ;load 25H into A

MOV R2,#34H        ;load 34H into R2

ADD A,R2            ;add R2 to accumulator
                    ;(A = A + R2)
END
```

Example 3: Addition

```
MOV PSW, #00h
MOV R5,#25H          ;load 25H into R5 (R5 = 25 h )
MOV R7 ,#34H         ;1oad 34H into R7 (R7 =34 h)
MOV A ,#0            ;load 0 Into A (A=0, clear A)
ADD A, R5            ;add to A content of R5
                     ;where A = A + R5
ADD A, R7            ;add to A Content of R7
                     ;where A = A + R7
END
```