



Dodatak

Skup instrukcija

Arithmetic Operations

- ◆ [$@Ri$] implies contents of memory location pointed to by R0 or R1
- ◆ Rn refers to registers R0-R7 of the currently selected register bank

Mnemonic	Description
ADD A, Rn	$A = A + [Rn]$
ADD A, direct	$A = A + [\text{direct memory}]$
ADD A, @Ri	$A = A + [\text{memory pointed to by Ri}]$
ADD A, #data	$A = A + \text{immediate data}$
ADDC A, Rn	$A = A + [Rn] + CY$
ADDC A, direct	$A = A + [\text{direct memory}] + CY$
ADDC A, @Ri	$A = A + [\text{memory pointed to by Ri}] + CY$
ADDC A, #data	$A = A + \text{immediate data} + CY$
SUBB A, Rn	$A = A - [Rn] - CY$
SUBB A, direct	$A = A - [\text{direct memory}] - CY$
SUBB A, @Ri	$A = A - [@Ri] - CY$
SUBB A, #data	$A = A - \text{immediate data} - CY$
INC A	$A = A + 1$
INC Rn	$[Rn] = [Rn] + 1$
INC direct	$[\text{direct}] = [\text{direct}] + 1$
INC @Ri	$[@Ri] = [@Ri] + 1$
DEC A	$A = A - 1$
DEC Rn	$[Rn] = [Rn] - 1$
DEC direct	$[\text{direct}] = [\text{direct}] - 1$
DEC @Ri	$[@Ri] = [@Ri] - 1$
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal adjust A

ADD A,<source-byte> ADDC A,<source-byte>

- ◆ ADD adds the data byte specified by the source operand to the accumulator, leaving the result in the accumulator
- ◆ ADDC adds the data byte specified by the source operand, the carry flag and the accumulator contents, leaving the result in the accumulator
- ◆ Operation of both the instructions, ADD and ADDC, can affect the carry flag (CY), auxiliary carry flag (AC) and the overflow flag (OV)
 - CY=1 If there is a carryout from bit 7; cleared otherwise
 - AC =1 If there is a carryout from the lower 4-bit of A i.e. from bit 3; cleared otherwise
 - OV=1 If the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

SUBB A,<source-byte>

- ◆ SUBB subtracts the specified data byte and the carry flag together from the accumulator, leaving the result in the accumulator

CY=1 If a borrow is needed for bit 7; cleared otherwise

AC =1 If a borrow is needed for bit 3, cleared otherwise

OV=1 If a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not into bit 6.

- ◆ *Example:*

The accumulator holds 0C1H (11000001B), Register1 holds 40H (01000000B) and the CY=1. The instruction,

SUBB A, R1

gives the value 70H (01110000B) in the accumulator, with the CY=0 and AC=0 but OV=1

INC <byte>

- ◆ Increments the data variable by 1. The instruction is used in register, direct or register direct addressing modes

- ◆ *Example:*

```
INC    6FH
```

If the internal RAM location 6FH contains 30H, then the instruction increments this value, leaving 31H in location 6FH

- ◆ *Example:*

```
MOV    R1, #5E
INC    R1
INC    @R1
```

- ◆ If R1=5E (01011110) and internal RAM location 5FH contains 20H, the instructions will result in R1=5FH and internal RAM location 5FH to increment by one to 21H

DEC <byte>

- ◆ The data variable is decremented by 1
- ◆ The instruction is used in accumulator, register, direct or register direct addressing modes
- ◆ A data of value 00H underflows to FFH after the operation
- ◆ No flags are affected

INC DPTR

- ◆ Increments the 16-bit data pointer by 1
- ◆ DPTR is the only 16-bit register that can be incremented
- ◆ The instruction adds one to the contents of DPTR directly

MUL AB

- ◆ Multiplies A & B and the 16-bit result stored in [B15-8], [A7-0]
- ◆ Multiplies the unsigned 8-bit integers in the accumulator and the B register
- ◆ The **Low** order byte of the 16-bit product will go to the accumulator and the **High** order byte will go to the B register
- ◆ If the product is greater than 255 (FFH), the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.
- ◆ If ACC=85 (55H) and B=23 (17H), the instruction gives the product 1955 (07A3H), so B is now 07H and the accumulator is A3H. The overflow flag is set and the carry flag is cleared.



DIV AB

- ◆ Divides A by B
- ◆ The integer part of the quotient is stored in A and the remainder goes to the B register
- ◆ If ACC=90 (5AH) and B=05(05H), the instruction leaves 18 (12H) in ACC and the value 00 (00H) in B, since $90/5 = 18$ (quotient) and 00 (remainder)
- ◆ Carry and OV are both cleared
- ◆ *If B contains 00H before the division operation, then the values stored in ACC and B are undefined and an overflow flag is set. The carry flag is cleared.*

DA A

- ◆ This is a decimal adjust instruction
- ◆ It adjusts the 8-bit value in ACC resulting from operations like ADD or ADDC and produces two 4-bit digits (in packed Binary Coded Decimal (BCD) format)
- ◆ Effectively, this instruction performs the decimal conversion by adding 00H, 06H, 60H or 66H to the accumulator, depending on the initial value of ACC and PSW
- ◆ If ACC bits A3-0 are greater than 9 (xxxx1010-xxxx1111), or if AC=1, then a value 6 is added to the accumulator to produce a correct BCD digit in the lower order nibble
- ◆ If CY=1, because the high order bits A7-4 is now exceeding 9 (1010xxxx-1111xxxx), then these high order bits will be increased by 6 to produce a correct proper BCD in the high order nibble but not clear the carry

Logical Operations

Mnemonic	Description
ANL A, Rn	$A = A \& [Rn]$
ANL A, direct	$A = A \& [\text{direct memory}]$
ANL A, @Ri	$A = A \& [\text{memory pointed to by Ri}]$
ANL A, #data	$A = A \& \text{immediate data}$
ANL direct, A	$[\text{direct}] = [\text{direct}] \& A$
ANL direct, #data	$[\text{direct}] = [\text{direct}] \& \text{immediate data}$
ORL A, Rn	$A = A \text{ OR } [Rn]$
ORL A, direct	$A = A \text{ OR } [\text{direct}]$
ORL A, @Ri	$A = A \text{ OR } [@Ri]$
ORL A, #data	$A = A \text{ OR immediate data}$
ORL direct, A	$[\text{direct}] = [\text{direct}] \text{ OR } A$
ORL direct, #data	$[\text{direct}] = [\text{direct}] \text{ OR immediate data}$
XRL A, Rn	$A = A \text{ XOR } [Rn]$
XRL A, direct	$A = A \text{ XOR } [\text{direct memory}]$
XRL A, @Ri	$A = A \text{ XOR } [@Ri]$
XRL A, #data	$A = A \text{ XOR immediate data}$
XRL direct, A	$[\text{direct}] = [\text{direct}] \text{ XOR } A$
XRL direct, #data	$[\text{direct}] = [\text{direct}] \text{ XOR immediate data}$
CLR A	Clear A
CPL A	Complement A
RL A	Rotate A left
RLC A	Rotate A left (through C)
RR A	Rotate A right
RRC A	Rotate A right (through C)
SWAP A	Swap nibbles

- ◆ Logical instructions perform Boolean operations (AND, OR, XOR, and NOT) on data bytes on a **bit-by-bit** basis

ANL <dest-byte>,<source-byte>

- ◆ This instruction performs the logical AND operation on the source and destination operands and stores the result in the destination variable

- ◆ No flags are affected

- ◆ *Example:*

ANL A,R2

If ACC=D3H (11010011) and R2=75H (01110101), the result of the instruction is ACC=51H (01010001)

- ◆ The following instruction is also useful when there is a need to mask a byte

- ◆ *Example:*

ANL P1,#10111001B



ORL <dest-byte>,<source-byte>

- ◆ This instruction performs the logical OR operation on the source and destination operands and stores the result in the destination variable
- ◆ No flags are affected

- ◆ *Example:*

```
ORL    A,R2
```

If ACC=D3H (11010011) and R2=75H (01110101), the result of the instruction is ACC=F7H (11110111)

- ◆ *Example:*

```
ORL    P1,#11000010B
```

This instruction sets bits 7, 6, and 1 of output Port 1



XRL <dest-byte>,<source-byte>

- ◆ This instruction performs the logical XOR (Exclusive OR) operation on the source and destination operands and stores the result in the destination variable

- ◆ No flags are affected

- ◆ *Example:*

```
XRL    A,R0
```

If ACC=C3H (11000011) and R0=AAH (10101010), then the instruction results in ACC=69H (01101001)

- ◆ *Example:*

```
XRL    P1,#00110001
```

This instruction complements bits 5, 4, and 0 of output Port 1



CLR A and CPL A

CLR A

- ◆ This instruction clears the accumulator (all bits set to 0)
- ◆ No flags are affected
- ◆ If ACC=C3H, then the instruction results in ACC=00H

CPL A

- ◆ This instruction logically complements each bit of the accumulator (one's complement)
- ◆ No flags are affected
- ◆ If ACC=C3H (11000011), then the instruction results in ACC=3CH (00111100)

RL A

- ◆ The 8 bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position.
- ◆ No flags are affected
- ◆ If ACC=C3H (11000011), then the instruction results in ACC=87H (10000111) with the carry unaffected

RLC A

- ◆ The instruction rotates the accumulator contents one bit to the left through the carry flag
- ◆ Bit 7 of the accumulator will move into carry flag and the original value of the carry flag will move into the Bit 0 position
- ◆ No other flags are affected
- ◆ If ACC=C3H (11000011), and the carry flag is 1, the instruction results in ACC=87H (10000111) with the carry flag set



RR A

- ◆ The 8 bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position.
- ◆ No flags are affected
- ◆ If ACC=C3H (11000011), then the instruction results in ACC=E1H (11100001) with the carry unaffected

RRC A

- ◆ The instruction rotates the accumulator contents one bit to the right through the carry flag
- ◆ The original value of carry flag will move into Bit 7 of the accumulator and Bit 0 rotated into carry flag
- ◆ No other flags are affected
- ◆ If ACC=C3H (11000011), and the carry flag is 0, the instruction results in ACC=61H (01100001) with the carry flag set

SWAP A

- ◆ This instruction interchanges the low order 4-bit nibbles (A3-0) with the high order 4-bit nibbles (A7-4) of the ACC
- ◆ The operation can also be thought of as a 4-bit rotate instruction
- ◆ No flags are affected
- ◆ If ACC=C3H (11000011), then the instruction leaves ACC=3CH (00111100)

Data Transfer Instructions

- ◆ Data transfer instructions can be used to transfer data between an internal RAM location and SFR location without going through the accumulator
- ◆ It is possible to transfer data between the internal and external RAM by using indirect addressing
- ◆ The upper 128 bytes of data RAM are accessed only by indirect addressing and the SFRs are accessed only by direct addressing

Data Transfer Instructions

Mnemonic	Description
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data 16	[DPTR] = immediate data
MOVC A,@A+DPTR	A = Code byte from [@A+DPTR]
MOVC A,@A+PC	A = Code byte from [@A+PC]
MOVX A,@Ri	A = Data byte from external ram [@Ri]
MOVX A,@DPTR	A = Data byte from external ram [@DPTR]
MOVX @Ri, A	External[@Ri] = A
MOVX @DPTR,A	External[@DPTR] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A,Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCH A, @Ri	A = [@Rn], [@Rn] = A
XCHD A,@Ri	Exchange low order digits

MOV <dest-byte>, <source-byte>

- ◆ This instruction moves the source byte into the destination location
- ◆ The source byte is not affected, neither are any other registers or flags
- ◆ *Example:*

```
MOV    R1, #60        ;R1=60H
MOV    A, @R1         ;A=[ 60H]
MOV    R2, #61        ;R2=61H
ADD    A, @R2         ;A=A+[ 61H]
MOV    R7, A          ;R7=A
```

- ◆ If internal RAM locations 60H=10H, and 61H=20H, then after the operations of the above instructions R7=A=30H. The data contents of memory locations 60H and 61H remain intact.

MOV DPTR, #data 16

- ◆ This instruction loads the data pointer with the 16-bit constant and no flags are affected
- ◆ *Example:*

```
MOV    DPTR, #1032
```
- ◆ This instruction loads the value 1032H into the data pointer, i.e. DPH=10H and DPL=32H.

MOVC A,@A + <base-reg>

- ◆ This instruction moves a code byte from program memory into ACC
- ◆ The effective address of the byte fetched is formed by adding the original 8-bit accumulator contents and the contents of the base register, which is either the data pointer (DPTR) or program counter (PC)
- ◆ 16-bit addition is performed and no flags are affected
- ◆ The instruction is useful in reading the look-up tables in the program memory
- ◆ If the PC is used, it is incremented to the address of the following instruction before being added to the ACC
- ◆ *Example:*

```
                CLR    A
LOC1:           INC    A
                MOVC   A,@A + PC
                RET
Look_up        DB     10H
                DB     20H
                DB     30H
                DB     40H
```

- ◆ The subroutine takes the value in the accumulator to 1 of 4 values defined by the DB (define byte) directive
- ◆ After the operation of the subroutine it returns ACC=20H

MOVX <dest-byte>, <source-byte>

- ◆ This instruction transfers data between ACC and a byte of external data memory
- ◆ There are two forms of this instruction, the only difference between them is whether to use an 8-bit or 16-bit indirect addressing mode to access the external data RAM
- ◆ The 8-bit form of the MOVX instruction uses the EMI0CN SFR to determine the upper 8 bits of the effective address to be accessed and the contents of R0 or R1 to determine the lower 8 bits of the effective address to be accessed

- ◆ *Example:*

```
MOV    EMI0CN, #10H    ;Load high byte of
                        ;address into EMI0CN.
MOV    R0, #34H        ;Load low byte of
                        ;address into R0(or R1).
MOVX   A, @R0          ;Load contents of 1034H
                        ;into ACC.
```

MOVX <dest-byte>, <source-byte>

- ◆ The 16-bit form of the MOVX instruction accesses the memory location pointed to by the contents of the DPTR register

- ◆ *Example:*

```
MOV    DPTR, #1034H ;Load DPTR with 16 bit
                        ;address to read (1034H).
MOVX   A, @DPTR    ;Load contents of 1034H
                        ;into ACC.
```

- ◆ The above example uses the 16-bit immediate MOV DPTR instruction to set the contents of DPTR
- ◆ Alternately, the DPTR can be accessed through the SFR registers DPH, which contains the upper 8 bits of DPTR, and DPL, which contains the lower 8 bits of DPTR

PUSH Direct

- ◆ This instruction increments the stack pointer (SP) by 1
- ◆ The contents of *Direct*, which is an internal memory location or a SFR, are copied into the internal RAM location addressed by the stack pointer
- ◆ No flags are affected

- ◆ *Example:*

PUSH 22H

PUSH 23H

- ◆ Initially the SP points to memory location 4FH and the contents of memory locations 22H and 23H are 11H and 12H respectively. After the above instructions, SP=51H, and the internal RAM locations 50H and 51H will store 11H and 12H respectively.

POP Direct

- ◆ This instruction reads the contents of the internal RAM location addressed by the stack pointer (SP) and decrements the stack pointer by 1. The data read is then transferred to the *Direct* address which is an internal memory or a SFR. No flags are affected.

- ◆ *Example:*

```
POP    DPH
POP    DPL
```

- ◆ If SP=51H originally and internal RAM locations 4FH, 50H and 51H contain the values 30H, 11H and 12H respectively, the instructions above leave SP=4FH and DPTR=1211H

```
POP    SP
```

- ◆ If the above line of instruction follows, then SP=30H. In this case, SP is decremented to 4EH before being loaded with the value popped (30H)

XCH A,<byte>

- ◆ This instruction swaps the contents of ACC with the contents of the indicated data byte
- ◆ *Example:*
`XCH A,@R0`
- ◆ Suppose R0=2EH, ACC=F3H (11110011) and internal RAM location 2EH=76H (01110110). The result of the above instruction leaves RAM location 2EH=F3H and ACC=76H.

XCHD A,@Ri

- ◆ This instruction exchanges the low order nibble of ACC (bits 0-3), with that of the internal RAM location pointed to by Ri register
- ◆ The high order nibbles (bits 7-4) of both the registers remain the same
- ◆ No flags are affected
- ◆ Example:

XCHD A,@R0

If R0=2EH, ACC=76H (01110110) and internal RAM location 2EH=F3H (11110011), the result of the instruction leaves RAM location 2EH=F6H (11110110) and ACC=73H (01110011)



Boolean Variable Instructions

- ◆ The C8051F020 processor can perform single bit operations
- ◆ The operations include *set*, *clear*, as well as *and*, *or* and *complement* instructions
- ◆ Also included are bit-level moves or conditional jump instructions
- ◆ All bit accesses use direct addressing

Mnemonic	Description
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
SETB bit	Set direct bit
CPL C	Complement c
CPL bit	Complement direct bit
ANL C,bit	AND bit with C
ANL C,/bit	AND NOT bit with C
ORL C,bit	OR bit with C
ORL C,/bit	OR NOT bit with C
MOV C,bit	MOV bit to C
MOV bit,C	MOV C to bit
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit,rel	Jump if specified bit set
JNB bit,rel	Jump if specified bit not set
JBC bit,rel	if specified bit set then clear it and jump

CLR <bit>

- ◆ This operation clears (reset to 0) the specified bit indicated in the instruction
- ◆ No other flags are affected
- ◆ CLR instruction can operate on the carry flag or any directly-addressable bit
- ◆ *Example:*

CLR P2.7

If Port 2 has been previously written with DCH (11011100), then the operation leaves the port set to 5CH (01011100)



SETB <bit>

- ◆ This operation sets the specified bit to 1
- ◆ SETB instruction can operate on the carry flag or any directly-addressable bit
- ◆ No other flags are affected
- ◆ *Example:*
 - `SETB C`
 - `SETB P2.0`
- ◆ If the carry flag is cleared and the output Port 2 has the value of 24H (00100100), then the result of the instructions sets the carry flag to 1 and changes the Port 2 value to 25H (00100101)

CPL <bit>

- ◆ This operation complements the bit indicated by the operand
- ◆ No other flags are affected
- ◆ CPL instruction can operate on the carry flag or any directly-addressable bit
- ◆ *Example:*
 - CPL P2.1
 - CPL P2.2
- ◆ If Port 2 has the value of 53H (01010011) before the start of the instructions, then after the execution of the instructions it leaves the port set to 55H (01010101)



ANL C, <source-bit>

- ◆ This instruction ANDs the bit addressed with the Carry bit and stores the result in the Carry bit itself
- ◆ If the source bit is a logical 0, then the instruction clears the carry flag; else the carry flag is left in its original value
- ◆ If a slash (/) is used in the source operand bit, it means that the logical complement of the addressed source bit is used, **but the source bit itself is not affected**
- ◆ No other flags are affected

- ◆ *Example:*

```
MOV    C,P2.0      ;Load C with input pin
                    ;state of P2.0.
ANL    C,P2.7      ;AND carry flag with
                    ;bit 7 of P2.
MOV    P2.1,C      ;Move C to bit 1 of Port 2.
ANL    C,/OV       ;AND with inverse of OV flag.
```

- ◆ If P2.0=1, P2.7=0 and OV=0 initially, then after the above instructions, P2.1=0, CY=0 and the OV remains unchanged, i.e. OV=0



ORL C, <source-bit>

- ◆ This instruction ORs the bit addressed with the Carry bit and stores the result in the Carry bit itself
- ◆ It sets the carry flag if the source bit is a logical 1; else the carry is left in its original value
- ◆ If a slash (/) is used in the source operand bit, it means that the logical complement of the addressed source bit is used, **but the source bit itself is not affected**
- ◆ No other flags are affected

- ◆ *Example:*

```
MOV    C,P2.0      ;Load C with input pin
                    ;state of P2.0.
ORL    C,P2.7      ;OR carry flag with
                    ;bit 7 of P2.
MOV    P2.1,C      ;Move C to bit 1 of
                    ;port 2.
ORL    C,/OV       ;OR with inverse of OV
                    ;flag.
```

MOV <dest-bit>,<source-bit>

- ◆ The instruction loads the value of source operand bit into the destination operand bit
- ◆ One of the operands **must** be the carry flag; the other may be any directly-addressable bit
- ◆ No other register or flag is affected

- ◆ *Example:*

```
MOV    P2.3,C
```

```
MOV    C,P3.3
```

```
MOV    P2.0,C
```

- ◆ If P2=C5H (11000101), P3.3=0 and CY=1 initially, then after the above instructions, P2=CCH (11001100) and CY=0.

JC rel

- ◆ This instruction branches to the address, indicated by the label, if the carry flag is set, otherwise the program continues to the next instruction
- ◆ No flags are affected

- ◆ *Example:*

```
CLR    C
SUBB   A,R0
JC     ARRAY1
MOV    A,#20H
```

- ◆ The carry flag is cleared initially. After the SUBB instruction, if the value of A is smaller than R0, then the instruction sets the carry flag and causes program execution to branch to ARRAY1 address, otherwise it continues to the MOV instruction.

JNC rel

- ◆ This instruction branches to the address, indicated by the label, if the carry flag is **not** set, otherwise the program continues to the next instruction
- ◆ No flags are affected. **The carry flag is not modified.**

- ◆ *Example:*

```
CLR    C
SUBB   A,R0
JNC    ARRAY2
MOV    A,#20H
```

- ◆ The above sequence of instructions will cause the jump to be taken if the value of A is greater than or equal to R0. Otherwise the program will continue to the MOV instruction.

JB <bit>,rel

- ◆ This instruction jumps to the address indicated if the destination bit is 1, otherwise the program continues to the next instruction
- ◆ No flags are affected. **The bit tested is not modified.**

- ◆ *Example:*

```
JB ACC.7,ARRAY1
```

```
JB P1.2,ARRAY2
```

- ◆ If the accumulator value is 01001010 and Port 1=57H (01010111), then the above instruction sequence will cause the program to branch to the instruction at ARRAY2



JNB <bit>,rel

- ◆ This instruction jumps to the address indicated if the destination bit is 0, otherwise the program continues to the next instruction
- ◆ No flags are affected. **The bit tested is not modified.**

- ◆ *Example:*

```
JNB ACC.6,ARRAY1
```

```
JNB P1.3,ARRAY2
```

- ◆ If the accumulator value is 01001010 and Port 1=57H (01010111), then the above instruction sequence will cause the program to branch to the instruction at ARRAY2



JBC <bit>,rel

- ◆ If the source bit is 1, this instruction clears it and branches to the address indicated; else it proceeds with the next instruction
- ◆ **The bit is not cleared if it is already a 0.** No flags are affected.
- ◆ *Example:*

```
JBC    P1.3,ARRAY1  
JBC    P1.2,ARRAY2
```
- ◆ If P1=56H (01010110), the above instruction sequence will cause the program to branch to the instruction at ARRAY2, modifying P1 to 52H (01010010)

Program Branching Instructions

- ◆ Program branching instructions are used to control the flow of actions in a program
- ◆ Some instructions provide decision making capabilities and transfer control to other parts of the program, e.g. conditional and unconditional branches

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn,#data,rel	
CJNE @Ri,#data,rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

ACALL addr11

- ◆ This instruction **unconditionally** calls a subroutine indicated by the address
- ◆ The operation will cause the PC to increase by 2, then it pushes the 16-bit PC value onto the stack (low order byte first) and increments the stack pointer twice
- ◆ The PC is now loaded with the value *addr11* and the program execution continues from this new location
- ◆ The subroutine called must therefore start within the same 2 kB block of the program memory

- ◆ No flags are affected

- ◆ *Example:*
ACALL LOC_SUB

- ◆ If SP=07H initially and the label “LOC_SUB” is at program memory location 0567H, then executing the instruction at location 0230H, SP=09H, internal RAM locations 08H and 09H will contain 32H and 02H respectively and PC=0567H

LCALL addr16

- ◆ This instruction calls a subroutine located at the indicated address
- ◆ The operation will cause the PC to increase by 3, then it pushes the 16-bit PC value onto the stack (low order byte first) and increments the stack pointer twice
- ◆ The PC is then loaded with the value *addr16* and the program execution continues from this new location
- ◆ Since it is a Long call, the subroutine may therefore begin anywhere in the full 64 kB program memory address space
- ◆ No flags are affected

- ◆ *Example:*

LCALL LOC_SUB

- ◆ Initially, SP=07H and the label “LOC_SUB” is at program memory location 2034H. Executing the instruction at location 0230H, SP=09H, internal RAM locations 08H and 09H contain 33H and 02H respectively and PC=2034H



RET

- ◆ This instruction returns the program from a subroutine
- ◆ RET pops the high byte and low byte address of PC from the stack and decrements the SP by 2
- ◆ The execution of the instruction will result in the program to resume from the location just after the “call” instruction
- ◆ No flags are affected
- ◆ Suppose SP=0BH originally and internal RAM locations 0AH and 0BH contain the values 30H and 02H respectively. The instruction leaves SP=09H and program execution will continue at location 0230H

RETI

- ◆ This instruction returns the program from an interrupt subroutine
- ◆ RETI pops the high byte and low byte address of PC from the stack and restores the interrupt logic to accept additional interrupts
- ◆ SP decrements by 2 and no other registers are affected. However the PSW is not automatically restored to its pre-interrupt status
- ◆ After the RETI, program execution will resume immediately after the point at which the interrupt is detected
- ◆ Suppose SP=0BH originally and an interrupt is detected during the instruction ending at location 0213H
 - Internal RAM locations 0AH and 0BH contain the values 14H and 02H respectively
 - The RETI instruction leaves SP=09H and returns program execution to location 0234H

AJMP addr11

- ◆ The AJMP instruction transfers program execution to the destination address which is located at the absolute short range distance (short range means 11-bit address)
- ◆ The destination must therefore be within the same 2kB block of program memory
- ◆ *Example:*
`AJMP NEAR`
- ◆ If the label NEAR is at program memory location 0120H, the AJMP instruction at location 0234H loads the PC with 0120H

LJMP addr16

- ◆ The LJMP instruction transfers program execution to the destination address which is located at the absolute long range distance (long range means 16-bit address)
- ◆ The destination may therefore be anywhere in the full 64 kB program memory address space
- ◆ No flags are affected
- ◆ *Example:*

```
LJMP FAR_ADR
```
- ◆ If the label FAR_ADR is at program memory location 3456H, the LJMP instruction at location 0120H loads the PC with 3456H



SJMP rel

- ◆ This is a short jump instruction, which increments the PC by 2 and then adds the relative value '*rel*' (signed 8-bit) to the PC
- ◆ This will be the new address where the program would branch to unconditionally
- ◆ Therefore, the range of destination allowed is from -128 to +127 bytes from the instruction
- ◆ *Example:*
SJMP RELSRT
- ◆ If the label RELSRT is at program memory location 0120H and the SJMP instruction is located at address 0100H, after executing the instruction, PC=0120H.



JMP @A + DPTR

- ◆ This instruction adds the 8-bit unsigned value of the ACC to the 16-bit data pointer and the resulting sum is returned to the PC
- ◆ Neither ACC nor DPTR is altered
- ◆ No flags are affected
- ◆ *Example:*

```
MOV    DPTR, #LOOK_TBL
JMP    @A + DPTR
LOOK_TBL: AJMP  LOC0
        AJMP  LOC1
        AJMP  LOC2
```

If the ACC=02H, execution jumps to LOC1

- ◆ AJMP is a two byte instruction

JZ rel

- ◆ This instruction branches to the destination address if ACC=0; else the program continues to the next instruction
- ◆ The ACC is not modified and no flags are affected

- ◆ *Example:*

```
SUBB A,#20H  
JZ    LABEL1  
DEC  A
```

- ◆ If ACC originally holds 20H and CY=0, then the SUBB instruction changes ACC to 00H and causes the program execution to continue at the instruction identified by LABEL1; otherwise the program continues to the DEC instruction

JNZ rel

- ◆ This instruction branches to the destination address if any bit of ACC is a 1; else the program continues to the next instruction
- ◆ The ACC is not modified and no flags are affected

- ◆ *Example:*

```
DEC    A
JNZ    LABEL2
MOV    RO, A
```

- ◆ If ACC originally holds 00H, then the instructions change ACC to FFH and cause the program execution to continue at the instruction identified by LABEL2; otherwise the program continues to MOV instruction



CJNE <dest-byte>, <source-byte>, rel

- ◆ This instruction compares the magnitude of the *dest-byte* and the *source-byte* and branches if their values are not equal
- ◆ The carry flag is set if the unsigned *dest-byte* is less than the unsigned integer *source-byte*; otherwise, the carry flag is cleared
- ◆ Neither operand is affected
- ◆ *Example:*

```
CJNE    R3, #50H, NEQU
        ... ..
NEQU:   JC      LOC1
        ... ..
LOC1:   ... ..
```

```
;R3 = 50H
;If R3 < 50H
;R7 > 50H
;R3 < 50H
```

DJNZ <byte>,<rel-addr>

- ◆ This instruction is "decrement jump not zero"
- ◆ It decrements the contents of the destination location and if the resulting value is not 0, branches to the address indicated by the source operand
- ◆ An original value of 00H underflows to FFH
- ◆ No flags are affected

- ◆ *Example:*

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.
 - Note, the first instruction will not branch to LOC1 because the [20H] = 00H, hence the program continues to the second instruction
 - Only after the execution of the second instruction (where the location [30H] = 5FH), then the branching takes place

NOP

- ◆ This is the no operation instruction
- ◆ The instruction takes one machine cycle operation time
- ◆ Hence it is useful to time the ON/OFF bit of an output port
- ◆ *Example:*

```
CLR    P1.2
```

```
NOP
```

```
NOP
```

```
NOP
```

```
NOP
```

```
SETB   P1.2
```

- ◆ The above sequence of instructions outputs a low-going output pulse on bit 2 of Port 1 lasting exactly 5 cycles.
 - Note a simple SETB/CLR generates a 1 cycle pulse, so four additional cycles must be inserted in order to have a 5-clock pulse width