## BRANCH CONTROL INSTRUCTIONS

Here you will see operations like:

- SJMP, AJMP & LJMP
- ACALL & LCALL
- RET & RETI
- CJNE
- DJNZ
- JC & JNC
- JZ & JNZ
- JB, JNB & JBC

# BRANCH OPERATIONS OF 8051 (SJMP, AJMP AND L JMP)

## SHORT JUMP

*Syntax*: **SJMP radd**; // Short Jump using the relative address

*Range*: **(-128 … +127) locations** because "radd" is an 8-bit signed number

*Size of instruction*: **2 Bytes** (Opcode of SJMP= 1Byte, radd = 1Byte)

*New address calculation*: **PC ← PC (address of next instruction) + radd**

*Usage*: **SJMP** (unconditional) and ALL Conditional jumps like JC, JNC, CJNE, DJNZ etc. ← Important for VIVA

*Description:* *Here the branch address (radd) is calculated as a relative distance from the next instruction to the branch location. In simple terms, instead of telling where we want to jump, we are telling how far we want to jump. This "radd" is then added to PC, which normally contains address of the next instruction.* ***For examples of SJMP, please refer #BharatSir Lecture Notes***

## ABSOLUTE JUMP

*Syntax*: **AJMP sadd**;// Absolute Jump using the short address

*Range*: **max 2KB** as long as the Jump is within the **Same Page**

*Size of instruction*: **2 Bytes** (Opcode of AJMP= 1Byte, sadd = 1Byte)

*New address calculation*:

| PC ← (16) | PC | Opcode of AJMP | Sadd |
|---|---|---|---|
| | 5 bits<br>Remains the same as branch is in the same page | 3 bits<br>Hence AJMP has 8 opcodes | 8 bits<br>Lower 8 bits of the jump location |

*Usage*: **AJMP and ACALL**.

*Description*: *Here the entire program memory (64 KB), is divided into 32 pages, each page being of 2KB. We can jump to any location of the same page, giving a max range of 2 KB. As the jump is in the same page, only the lower 11 bits of the address will change. Out of them, lower 8 bits are given by "sadd" and the higher 3 bits are given by the opcode of AJMP. 3 bits have 8 combinations, hence AJMP has 8 opcodes.* ***For examples of AJMP, please refer #BharatSir Lecture Notes***

## LONG JUMP

*Syntax*: **LJMP ladd**; // Long Jump using the long (full) address

*Range*: **64 KB** because "ladd" is a 16-bit address so can be any value from 0000H… FFFFH.

*Size of instruction*: **3 Bytes** (Opcode of LJMP= 1Byte, ladd = 2Bytes)

*New address calculation*: **PC ← ladd**

*Usage*: **LJMP, LCALL.**

*Description*: *This is the simplest type of Jump. Here we simply give the address where we wish to jump using "ladd". This "ladd" is then simply put into PC.* ***For examples of LJMP, please refer #BharatSir Lecture Notes***

| | Jump operation | Call operation |
|---|---|---|
| 1 | In a Jump, we simply branch to the new location and continue from there on. | In a Call, we branch to the new location, which is basically a subroutine, we execute the subroutine, and at the end, we return to the main program right at the NEXT instruction after the Call instruction. |
| 2 | There is no intention to return to the previous location.. | To invoke the subroutine we use Call instruction.<br>To return to the main program we use RET instruction. |
| 3 | There is no need for storing any return address into the stack | During Call, the return address (PC), is pushed into the stack.<br>During RET, the return address is popped from the stack and put back into PC. |
| 4 | Jumps can be of three types: Short, Absolute or Long Jump. | Call can be of two types: Absolute or Long Call. |
| 5 | Furthermore, Jumps can be unconditional or conditional. | Calls are always unconditional in 8051. |

| | RET | RETI |
|---|---|---|
| 1 | RET is used at the end of an ordinary Subroutine | RETI is used at the end of an ISR |
| 2 | RET will simply return back to the next instruction of the main program. | RETI will not only return back to the next instruction of the main program, but will also re-enable the interrupts, by making EA bit ← 1. |
| 3 | Operation:<br>**POP PC;** PCH← [SP]<br>PCL← [SP-1]<br>SP← SP-2 | Operation:<br>**POP PC;** PCH← [SP]<br>PCL← [SP-1]<br>SP← SP-2<br>**EA ←1;** Enables interrupts by making EA bit "1" in the IE-SFR. |

## 79) SJMP radd | "Short jump to relative address"

Example:

**SJMP label;**

*Operation:*

***Program takes a short jump to the location "label".***
*Please refer to the earlier discussion for a detailed explanation on SJMP, AJMP and LJMP.*
*They are VERY IMPORTANT for Theory, Practicals and VIVA exams.*

No of cycles required: **2**

---

## 80) AJMP sadd | "Absolute jump to short address"

Example:

**AJMP label;**

*Operation:*

***Program takes an absolute jump to the location "label".***
*Please refer to the earlier discussion for a detailed explanation on SJMP, AJMP and LJMP.*
*They are VERY IMPORTANT for Theory, Practicals and VIVA exams.*

No of cycles required: **2**

---

## 81) LJMP ladd | "Long jump to long address"

Example:

**LJMP label;**

*Operation:*

***Program takes a long jump to the location "label".***
*Please refer to the earlier discussion for a detailed explanation on SJMP, AJMP and LJMP.*
*They are VERY IMPORTANT for Theory, Practicals and VIVA exams.*

No of cycles required: **2**

---

## 82) JMP @A+DPTR | "Jump"

Example:

**JMP @A+DPTR;**

*Operation:*

***Program takes a jump to the location whose address is calculated by A + DPTR.***
*If A = 25H and DPTR is 1000H then the program will jump to location 1025H by putting 1025H into PC.*

No of cycles required: **2**

## CONDITIONAL JUMPS

A conditional jump instruction can have two possible executions, depending upon the condition.
If condition is true, program will jump to the branch location, specified by the label.
If condition is false, program will simply proceed to the next instruction.
Please do remember, **All conditional jumps are SHORT type of jumps.**

---

## 87) DJNZ Rr, radd | "Decrement and Jump if Not Zero"

Example:

**DJNZ R7, Back; Decrement R7 and if not equal to "0", go to "Back"**

*Operation:*

***This instruction is extremely useful.***
*It is used by the programmer to create loops.*
*We initialize the loop count in some register like R7.*
*At the end of the loop we write this instruction: "DJNZ R7, Back"*
*Back is the Label where we wish to begin the loop from.*
*This instruction will first decrement R7.*
*It will then check if R7 has become 0 or not.*
*If R7 is NOT ZERO, program will jump to the location Back.*
*This will go on happening till R7 finally becomes Zero.*
*That's when the loop will break and program will simply proceed to the next instruction.*
*The count could have been in any register from R0… R7.*
*Get this clear, whatever is the count, we get the exact same number of iterations.*
*Neither one more, nor one less.*
*A count of 5 will produce exactly 5 iterations, neither 4 and nor 6.*
*Hope you remember various cases we discussed at BHARAT ACADEMY, for minimum and maximum number of iterations.*

No of cycles required: **2**

---

## 88) DJNZ addr, radd | "Decrement and Jump if Not Zero"

Example:

**DJNZ 25H, Back; Decrement the contents of location 25H, and if not equal to "0", go to "Back"**

*Operation:*

***This instruction is again used to make loops just like the previous one.***
***Except that, now the count is in a memory address, instead of a register.***
*Say, we initialize the loop count in some RAM location like 25H.*
*At the end of the loop we write this instruction: "DJNZ 25H, Back"*
*This instruction will first decrement contents of location 25H.*
*It will then check if the value has become 0 or not.*
*Based on that, it will do the iterations the same way as explained in the above instruction.*

No of cycles required: **2**

---

## 89) CJNE A, #n, radd | "Compare and Jump if Not Equal"

Example:

**CJNE A, #25H, Down; Compare A with number 25H. If not equal jump to location "Down"**

*Operation:*

***This instruction will compare A register with the number 25H.***
***If they are NOT EQUAL, program will jump to location "Down", else will simply proceed to the next instruction.***

No of cycles required: **2**

---

## 90) CJNE A, addr, radd | "Compare and Jump if Not Equal"

Example:

**CJNE A, 25H, Down; Compare A with [25H]. If not equal jump to location "Down"**

*Operation:*

***This instruction will compare A register with the contents of location 25H.***
***If they are NOT EQUAL, program will jump to location "Down", else will simply proceed to the next instruction.***

No of cycles required: **2**

---

## 91) CJNE Rr, #n, radd | "Compare and Jump if Not Equal"

Example:

**CJNE R0, #25H, Down; Compare R0 with number 25H. If not equal jump to location "Down"**

*Operation:*

***This instruction will compare R0 register with the number 25H.***
***If they are NOT EQUAL, program will jump to location "Down", else will simply proceed to the next instruction.***

No of cycles required: **2**

---

## 92) CJNE @Rp, #n, radd | "Compare and Jump if Not Equal"

Example:

**CJNE @R0, #25H, Down; Compare [R0] with number 25H. If not equal jump to location "Down"**

*Operation:*

***This instruction will compare the contents of the location pointed by R0, with the number 25H.***
***If they are NOT EQUAL, program will jump to location "Down", else will simply proceed to the next instruction.***

No of cycles required: **2**

## 93) JC radd | "Jump if Carry"

Example:

**JC Down; If Carry flag is "1", jump to location "Down"**

*Operation:*

***This instruction is used to check the Carry flag.***
***If Carry flag = "1", program will jump to the location "Down"***
***Else, program will simply proceed to the next location.***
*JC and JNC are used very frequently in programs where carry is involved.*

No of cycles required: **2**

---

## 94) JNC radd | "Jump if No Carry"

Example:

**JNC Down; If Carry flag is "0", jump to location "Down"**

*Operation:*

***This instruction is used to check the Carry flag.***
***If Carry flag = "0", program will jump to the location "Down"***
***Else, program will simply proceed to the next location.***

No of cycles required: **2**

---

## 95) JZ radd | "Jump if Zero"

Example:

**JZ Down; If A register is "Zero", jump to location "Down"**

*Operation:*

***This instruction is used to check the value of A Register.***
***If A Register = "0", program will jump to the location "Down"***
***Else, program will simply proceed to the next location.***
*In other processors, JZ and JNZ operate on Zero flag.*
*Since 8051 does not have a Zero Flag, these instructions simply check the value of A Register.*

No of cycles required: **2**

---

## 96) JNZ radd | "Jump if Not Zero"

Example:

**JNZ Down; If A register is "Not Zero", jump to location "Down"**

*Operation:*

***This instruction is used to check the value of A Register.***
***If A Register ≠ "0", program will jump to the location "Down"***
***Else, program will simply proceed to the next location.***

No of cycles required: **2**

## BOOLEAN CONDITIONAL INSTRUCTIONS

These instructions check the value if individual bits to decide whether to Jump or Not.
They are very useful in Embedded system Application programming of Timers, Serial Port etc.

**Often this sub-group is asked as an independent 5 mark question in the exam.**

---

### 97) JB bit, radd | "Jump if Bit"

Example:

**JB P0.0, Down; If P0.0 = "1", jump to location "Down"**

*Operation:*

***This instruction is used to check the value of a bit.***
*Suppose the bit in question is P0.0*
*If P0.0 = "1", then jump to location "Down"*
*Else, simply proceed top next location.*

No of cycles required: **2**

---

### 98) JNB bit, radd | "Jump if Not Bit"

Example:

**JNB P0.0, Down; If P0.0 = "0", jump to location "Down"**

*Operation:*

***This instruction is used to check the value of a bit.***
*Suppose the bit in question is P0.0*
*If P0.0 = "0", then jump to location "Down"*
*Else, simply proceed top next location.*

No of cycles required: **2**

---

### 99) JBC bit, radd | "Jump if Bit and complement/ clear"

Example:

**JBC P0.0, Down; If P0.0 = "1", jump to location "Down" and make P0.0 ← "0"**

*Operation:*

***This instruction is used to check the value of a bit.***
*Suppose the bit in question is P0.0*
*If P0.0 = "1", then jump to location "Down", and while jumping, also make P0.0 ← 0*
*Else, simply proceed top next location.*

No of cycles required: **2**