8051 Addressing Mode and Instruction Set

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

SILICON LABS

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     R0, 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
       … …                    ;R3 = 50H
NEQU:  JC     LOC1            ;If R3 < 50H
       … …              ;R7 > 50H
LOC1:  … …                    ;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

8051 Instruction Set

ACALL: Absolute Call

ADD, ADDC: Add Acc. (With Carry)

AJMP: Absolute Jump

ANL: Bitwise AND

CJNE: Compare & Jump if Not Equal

CLR: Clear Register

CPL: Complement Register

DA: Decimal Adjust

DEC: Decrement Register

DIV: Divide Accumulator by B

DJNZ: Dec. Reg. & Jump if Not Zero

INC: Increment Register

JB: Jump if Bit Set

JBC: Jump if Bit Set and Clear Bit

JC: Jump if Carry Set

JMP: Jump to Address

JNB: Jump if Bit Not Set

JNC: Jump if Carry Not Set

JNZ: Jump if Acc. Not Zero

JZ: Jump if Accumulator Zero

LCALL: Long Call

LJMP: Long Jump

MOV: Move Memory

MOVC: Move Code Memory

MOVX: Move Extended Memory

MUL: Multiply Accumulator by B

NOP: No Operation

ORL: Bitwise OR

POP: Pop Value From Stack

PUSH: Push Value Onto Stack

RET: Return From Subroutine

RETI: Return From Interrupt

RL: Rotate Accumulator Left

RLC: Rotate Acc. Left Through Carry

RR: Rotate Accumulator Right

RRC: Rotate Acc. Right Through Carry

SETB: Set Bit

SJMP: Short Jump

SUBB: Sub. From Acc. With Borrow

SWAP: Swap Accumulator Nibbles

XCH: Exchange Bytes

XCHD: Exchange Digits

XRL: Bitwise Exclusive OR

Undefined: Undefined Instruction