8051 Addressing Mode and Instruction Set

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)