ENGR 323L: Microprocessor Systems
Engineering Department, Trinity College
Instructor: Professor T. Ning

What is "assembly language" and why we use it?

- Speedy computation
- Compact program size & save implementation cost
- Special application, e.g., non-standard drivers
- To fully understand how the computer system works

ADDRESSING MODES OF 8051 FAMILY MICROCONTROLLERS:

Immediate addressing: the source being a constant embedded into code

```
e.g.,   mov   a, #0h

        mov   a, #11h
```

Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory.

For example, the instruction:

MOV A, #20h

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 20 (hexidecimal).

Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

Direct addressing: Direct addressing is so-named because the value to be stored in memory is obtained by directly retrieving it from another memory location, e.g., to specify an internal data register of an SFR by its address

```
e.g.,   mov   a, 70h  ;copy contents of internal register 70h to a

        mov   90h, a  ;copy the content of acc to SFR 90h (port 1)
```

note: the preferred way to handle references to the SFRs is by defining symbols using pseudo-operations

```
e.g.,   PORT1   equ   90h          ;define symbol PORT1
                mov   PORT1, a
```

Direct addressing is generally fast since, although the value to be loaded isn't included in the instruction, it is quickly accessable since it is stored in the 8051's Internal RAM. It is also much more flexible than Immediate Addressing since the value to be loaded is whatever is found at the given address--which may be variable.

Also, it is important to note that when using direct addressing any instruction which refers to an address between 00h and 7Fh is referring to Internal Memory. Any instruction which refers to an address between 80h and FFh is referring to the SFR control registers that control the 8051 microcontroller itself.

The obvious question that may arise is, "If direct addressing an address from 80h through FFh refers to SFRs, how can I access the upper 128 bytes of Internal RAM that are available on the 8052?" The answer is: You can't access them using direct addressing. As stated, if you directly refer to an address of 80h through FFh you will be referring to an SFR. However, you may access the 8052's upper 128 bytes of RAM by using the next addressing mode, "indirect addressing."

Indirect Addressing Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. *Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052*.

Indirect addressing appears as follows:

MOV A, @R0

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM which is found at the address indicated by R0.

For example, let's say R0 holds the value 40h and Internal RAM address 40h holds the value 67h. When the above instruction is executed the 8051 will check the value of R0. Since R0 holds 40h the 8051 will get the value out of Internal RAM address 40h (which holds 67h) and store it in the Accumulator. Thus, the Accumulator ends up holding 67h.

Indirect addressing always refers to Internal RAM; it never refers to an SFR. Thus, in a prior example we mentioned that SFR 99h can be used to write a value to the serial port. Thus one may think that the following would be a valid solution to write the value '1' to the serial port:

MOV R0, #99h ;Load the address of the serial port
MOV @R0, #01h ;Send 01 to the serial port -- WRONG!!
**This is not valid. Since indirect addressing always refers to Internal RAM these two instructions would write the value 01h to Internal RAM address 99h on an 8052. On an 8051 these two instructions would produce an undefined result since the 8051 only has 128 bytes of Internal RAM.*

External Direct External Memory is accessed using a suite of instructions which use what I call "External Direct" addressing. I call it this because it appears to be direct addressing, but it is used to access external memory rather than internal memory.

There are only two commands that use External Direct addressing mode:

MOVX A, @DPTR
MOVX @DPTR, A

As you can see, both commands utilize DPTR. In these instructions, DPTR must first be loaded with the address of external memory that you wish to read or write. Once DPTR holds the correct external memory address, the first command will move the contents of that external memory address into the Accumulator. The second command will do the opposite: it will allow you to write the value of the Accumulator to the external memory address pointed to by DPTR.

External Indirect External memory can also be accessed using a form of indirect addressing which I call External Indirect addressing. This form of addressing is usually only used in relatively small projects that have a very small amount of external RAM. An example of this addressing mode is:

MOVX @R0, A

Once again, the value of R0 is first read and the value of the Accumulator is written to that address in External RAM. Since the value of @R0 can only be 00h through FFh the project would effectively be limited to 256 bytes of External RAM. There are relatively simple hardware/software tricks that can be implemented to access more than 256 bytes of memory using External Indirect addressing; however, it is usually easier to use External Direct addressing if your project has more than 256 bytes of External RAM.

Register addressing: either the source or the destination being one of the eight registers of the active register bank

```
e.g.,   mov   psw    #10h ;select register bank 2
        mov   R0, a
        mov   r7, b

        mov   psw    #18h ;select register bank 2
        mov   R0, a
        mov   r7, b
```

Register specific addressing: some instructions are specific to the registers used

```
e.g.,   mov   a, #1   ;set 1 to acc (2 bytes: 74 01)
                      ;74: take the following byte and place in the accumulator

        mov   0E0h, #1;move the constant 1 into SFR E0h (3 bytes: 75 E0 01)
```

```
                    ;75: of the following two bytes, the first is the address of
                    a ;register into which the second byte
```

abbreviation ACC is defined as a symbol of the constant E0h

```
        mov     a, #1    ;register specific
        mov     acc, #1          ;direct addressing
```

Register indirect addressing: the address of the source or destination is not given explicitly. Instead, the content of a register is used as the target address

```
e.g.,   mov     r0, #78h         ;move 78h into register

        mov     @r0, #1          ;set the register whose address is specified in r0
                                 register, i.e., 78h
```

In all of the addressing modes, one may deduce the source or destination of the data transfer by inspecting the code. The exact source or destination is not necessarily known when the program is assembled. This allows the data flow to be manipulated in run time. (note: registers may be considered to hold data) Especially useful in constructing a data transfer loop, e.g., to copy a table.

```
e.g.,   mov     a,  #1
        mov     DPTR, #9000h
        movx    @DPTR, a

        mov     DPTR, #9001h
        movx    a, @DPTR

        mov     a, #1
        mov     r0, #0

        mov     0a0h, #90h
        movx    @r0, a

        mov     r0, #1

        movx    a, @r0
```

Register index addressing: the source of destination address is obtained by adding the value held in the accumulator to the base address

```
e.g.,   mov     DPTR, #8100h         ;set the table address

        mov     a, #0            ;
        movc    a, @a+DPTR  ;
```

Stack-oriented data transfer: a form of register indirect addressing: PUSH and POP the content of the SP is the address of the destination register in PUSH operations and is the address of the source in POP operations. The SP is incremented before the data transfer in push instructions and decremented after POP instructions. It's a good practice to initialize SP=2F or larger since 00-2F is the range of the four register banks.

```
e.g.,   [SP=2Fh]   PUSH  acc      ; put the content in acc into 30H
        [SP=30h]
```

note: PUSH and POP must be used in direct addressing mode

```
e.g.,   PUSH       a       ;register addressing (not recognized)
        PUSH       acc     ;direct addressing (acc is a symbol)
```

```
e.g.,   org    8000h
        mov    SP, #4Fh      ;initialize SP with 4F
        mov    a, #45h       ;45 is saved to accumulator
        push   acc           ;45 is pushed to 50h
        mov    b, #0
        pop    b
```

Exchange Instructions: powerful two-way data transfer

XCH (byte-wise)
1st operand: accumulator
2nd operand: internal data register (direct addressing); a register in a currently active register bank(register addressing); the internal register whose address is in R0 or R1(register indirect addressing)

```
e.g.,   XCH   a, b
        XCH   a, r0
        XCH   a, @r1
```

XCHD (nibble-wise): exchange the lower nibbles of the accumulator with an internal data register whose address is in R0 or R1

```
e.g.,   mov   r1, #7Fh      ;put 7F to r1
        mov   7Fh, #1       ;put 1 to register 7F
        xchd  a, @r1        ;exchange low nibbles of a and register 7F
```

Bit-oriented data transfer an embedded controller often manipulates single-bit data signals such as the status of a push button and the on/off switch of a motor driver. 80C51 family MCs offer extensive instructions for bit-oriented operations.

Bit addressable registers:
1) *internal registers in the range 20h-2Fh* and
2) *SFRs whose addresses are multiples of 8* (i.e., the low nibbles are either 0 or 8)
e.g., port1 (90), Acc, B, PSW, P0, P1, P2, P3, IE, IP, SCON, TCON

The bit address of their individual bits are computed as the SFR's address plus the digit of the bit

e.g., bit-5 address of ACC (E0h) is E5H
bit-0 address of PORT1 (90h) is 90h

The (jth) bit address of an internal register (i) between 20h - 2Fh is calculated as the following:

8 x (i-20) + j, for i = 20,21,...2Fh, and j=0,1,...7

conversely, a bit address *n* is in the

i = int (n/8) + 20h register of the
j = n - 8 x (i-20h) bit

e.g., bit 7 of register 20 has an address 07h (0*8+7=7)
bit 7 of register 2F has an address 7Fh (15*8+7=127)

Single bist may be moved between any addressable bits and the carry flag
The carry flag (C) is often used by the 8051 family of MCs as the 1-bit accumulator (the carry bit is bit 7 of the PSW)

e.g., connect bit 0 of Port 1 to a pushbutton and bit 1 of Port 1 to an LED

```
mov   c, p1.0      ;move the pushbutton status to the carry flag
mov   p1.1, c      ;move the carry flag to the LED

mov   c, 90h
mov   91h, c
```