CS253 Architectures II

Lecture 3

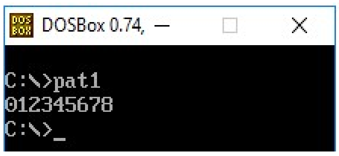
Assembly Language

Charles Markham

Printing patterns to the console

The code below loops dl through the values from 48 to 57. This is the range of ASCII character codes used on the console screen to display the numerals 0 to 8.

```
        .MODEL    medium
        .STACK
        .DATA
        .CODE
        .STARTUP

        mov dl,'0'  ; '0' is ASCII 48

again:  mov ah,02h  ; print ASCII
        int 021h    ; char in dl

        inc dl;

True    cmp dl,'9'  ; '9' is
        jnz again   ; ASCII 57
False
        .EXIT
END
```



The cmp command subtracts 57 (the ASCII value for '9') from dl and sets the Carry and Zero flags of the CPU. However the value in dl is not changed (use the `sub dl,` command if you want to do this). If dl contains 57 then the zero flag will be set. The jnz (jump if not zero) command then acts on the state of CPU zero flag. The zero flag is used to check equivalence.

Printing patterns to the console

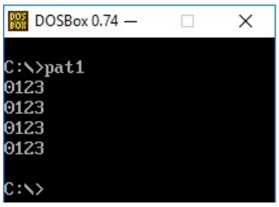
The code shown uses a nested loop so as to create a grid of numbers.

```
          .STARTUP
          mov bl,4

nextline: mov dl,'0'  ; '0' is ASCII 48

nextchar: mov ah,02h  ; print ASCII char in dl
          int 021h

          inc dl;

          cmp dl,'4'  ; '4' is ASCII 53
          jnz nextchar

          push dx     ; save value of dl and dh on stack
          mov ah,02h  ; print ASCII char in dl
          mov dl,10   ; line feed (next line)
          int 021h
          mov dl,13   ; carriage return (move to start of line)
          int 021h
          pop dx      ; restore value in dl (and dh)

          dec bl
          jnz nextline
          .EXIT
```

dl['0','3']

bl[0,3]

CR,LF



Carriage return (return to start of line) is ASCII 13
Line feed (move to next line) is ASCII 10

Addressing Modes

The different ways the processor can access memory are known as addressing modes.

mov al,…… or mov …..,al

* Immediate addressing
* Register addressing
* Direct addressing
* Register Indirect Addressing

  Register Indirect with Displacement

  Register Indirect with base and register

  Register Indirect with base and index and constant

Immediate addressing

In immediate addressing the operand (data) appears in the instruction. An example of an immediate instruction is one that moves a constant into an internal register.

mov ax,568

The constant 568 is loaded into ax .

Register Addressing

The addressing mode indicates that the operand is contained in one of the general registers of the CPU. The registers can be 8 or 16 bits. All operations in register addressing are internal to the CPU, no 20bit address is required.

mov ax,bx

The contents of register bx is copied into ax .

mov dl,bh

The contents of register bh is copied into dl .

Register Addressing

The addressing mode indicates that the operand is contained in one of the general registers of the CPU. The registers can be 8 or 16 bits. All operations in register addressing are internal to the CPU, no 20bit address is required.

mov ax,bx

The contents of register bx is copied into ax .

mov dl,bh

The contents of register bh is copied into dl .

Direct Addressing

A direct address completely specifies in the instruction the location in memory that contains the operand. A 20-bit address is formed by combining an OFFSET value specified in the instruction with the relevant segment register.

mov ax,*Count*

The contents of the memory location DS:*Count* is copied into ax .

Note you can force the uP to use a different segment register, using the following syntax.

mov ax,ES:*Count*

Where ES is the extra segment register.

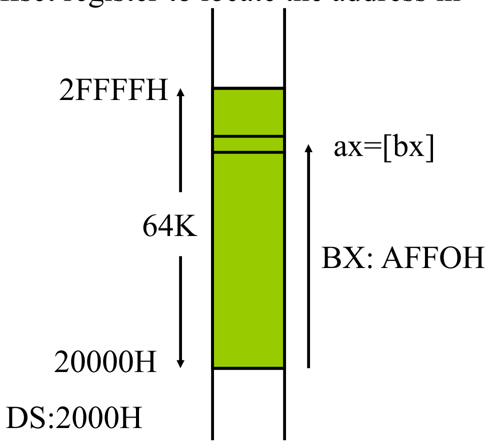
Register Indirect Addressing

In this mode of addressing the 16 bit offset address is contained in a 16 bit register (index register, BX,BP,SI or DI). The segment register is combined with the offset register to locate the address in memory to be accessed.

mov ax, [bx]

In this example ax is loaded with the contents of memory locations 2AFF0H and 2AFF1H.

Note: The high byte (bh) comes from the higher memory address.



Register Indirect Addressing

Register Indirect Addressing is the method of choice for accessing memory in a similar manner to arrays in higher level languages.

Move the contents of the memory location pointed to by register bx in the data segment into register ax

mov ax, [bx]

Put the value stored in register ax into the location pointed to by bx in the data segment.

mov [bx],ax

Short program to show the operation of the four addressing modes.

```
0000                            .DATA

0000    04D2            A       WORD     1234
0002    10E1            B       WORD     4321

0000                            .CODE
                                .STARTUP

0017    B8 1A85                 mov ax,6789

001A    BB 0000 R               mov bx,OFFSET  A

001D    8B 17                   mov dx,[bx]     ; Register indirect dx=1234

001F    8B 16 0002 R            mov dx,B        ; Direct dx=4321

0023    BA 2694                 mov dx,9876     ; Immediate dx=9876

0026    8B D0                   mov dx,ax       ; Register dx=6789

                                .EXIT

Symbol table:   N a m e         Type    Value(Address)
                  A             Word     0000
                  B             Word     0002
```

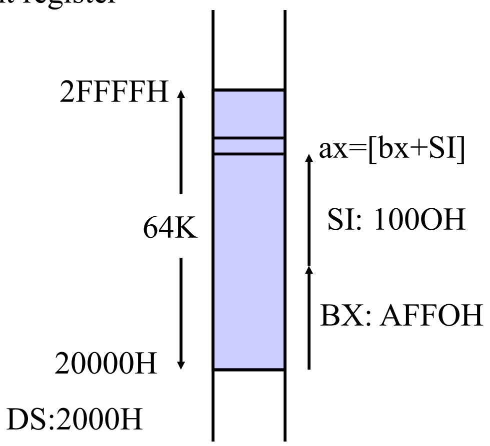
Register Indirect with Base and Index Register

This mode is very similar to the previous mode, the offset is obtained by adding two 16 bit registers and then combining the result with the relevant segment register

mov ax, [bx][SI]

Offset = BX+SI

ax is loaded with the contents of
2000:(1000+AFF0)=2BFF0

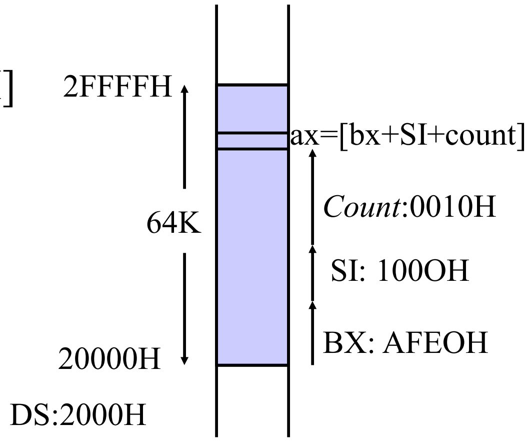


Register Indirect with Base,Constant and Index Register

This register addressing mode combines three numbers to create the offset address.

mov ax,*Count* [bx][SI]

Offset = BX+SI+*Count*



8086 Instruction Format

Byte 1								Byte 2			Low Byte	High Byte
1	0	0	0	1	0							
opcode						D	W	Mod	Reg	R/M		

Opcode specifies the function in this case MOV

D gives direction, From Register D=0, To Register D=1

Reg: Used to identify the register.

W=0 To move a word (e.g. AX) W=1 To move a byte (e.g. BL)

Mod: Addressing mode, offset information

R/M: Addressing mode, the register

8086 Instruction code for registers

Register W=0	Register W=1	Reg
AL	AX	000
BL	BX	011
CL	CX	001
DL	DX	010
AH	SP	100
BH	DI	111
CH	BP	101
DH	SI	110

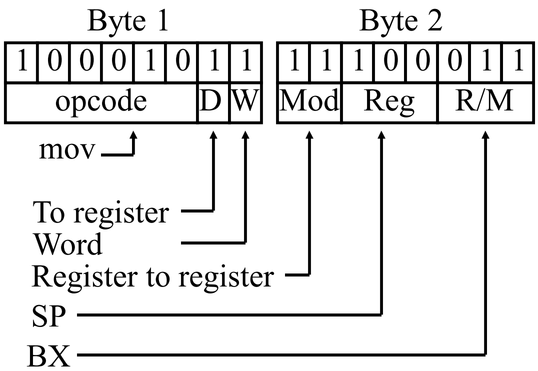
8086 Instruction Code

R/M \ Mod	00	01	10	11 W=0	11 W=1
000	[BX]+[SI]	[BX]+[SI]+d8	[BX]+[SI]+d16	AL	AX
001	[BX]+[DI]	[BX]+[DI]+d8	[BX]+[DI]+d16	CL	CX
010	[BP]+[SI]	[BP]+[SI]+d8	[BP]+[SI]+d16	DL	DX
011	[BP]+[DI]	[BP]+[DI]+d8	[BP]+[DI]+d16	BL	BX
100	[SI]	[SI]+d8	[SI]+d16	AH	SP
101	[DI]	[DI]+d8	[DI]+d16	CH	BP
110	d16	[BP]+d8	[BP]+d16	DH	SI
111	[BX]	[BX]+d8	[BX]+d16	BH	DI

Mod and R/M bits select the addressing mode to be used.

Coding Example

mov SP,BX



Hand coding vs Assembler

Using the ideas shown earlier it is possible to work out the binary code for each instruction that you wish to perform.

This is a very slow and tiring way to create code.

Assemblers greatly simplify the task of converting instructions (Operators and operands) to binary code.

However it was worth seeing how an instruction set is configured as it helps us to see how the CPU decodes an instruction.

Reading from the Keyboard

```
        .STARTUP

nextc:  mov ah,8
        int 021h   ;al character pressed
        mov dl,al

        mov ah,02h ;print dl
        int 021h

        cmp dl,'q'
        jnz nextc

        .EXIT
```

Read a character from the keyboard, ASCII in al.

Very simple typing tutor (14 bytes!)

ASCII

Characters are represented by a 7 bit binary code known as ASCII.

ASCII American Standard Code For Information Interchange.

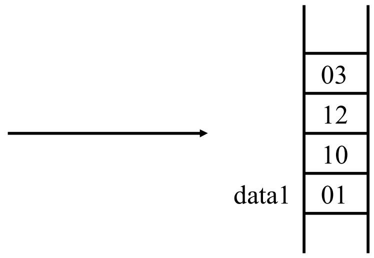
Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch
32	sp	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

Note: To convert a number in the range (0,9) from BCD (unpacked) to ASCII add 30H or 48Decimal.

Assembler Directives

SEGMENT	Start of a code block, like { in c.
ENDS	End of a code block, like } in c.
EQU	Equate, assigns a name to a constant.
DB	Define byte, 8 bit data
DW	Define word, 16 bit data
DD	Define double word, 32 bit data

```
        .MODEL   medium
        .STACK
        .DATA
data1   DB   01H, 10H,
        DW   312H

        .CODE
        .STARTUP
```



Compiling a program

Enter the DOS Prompt from Win95 or WinNT.

CD c:\temp\ or CD c:\masm\go (Win95)

EDIT name.asm or use Notepad

Type in your program, save it and exit.

MASM /L name.asm (to compile the code)

link name.obj (to link the object file)

name (to run the .exe file created)

name.lst (a full listing created by MASM)

Hand in your best program from the lab session.