EC325 Microprocessors
Bit Manipulations

Yasser F. O. Mohammad

REMINDER 1: General Info About String Instructions

- Source is always in DS:ESI
- Destination (if any) is always in ES:EDI
- To know the size of each element:
  1. Add two operands that are ignored but their size used (e.g. movs ax,bx)
  2. Add suffixes to instructor
     1. b (BYTE)
     2. w (WORD)
     3. d (DWORD)
- ESI/EDI are incremented/decremented after execution.
- Direction is controlled by DF (Direction Flag)
  - 1 means decrement (right to left)
  - 0 means increment (left to right)

REMINDER 2: String Instructions

- MOVS[B|W|D]
  - Moves a string
- SCAS[B|W|D]
  - Scans a string
- STOS[B|W|D]
  - Stores a string
- LODS[B|W|D]
  - Loads a string
- CMPS[B|W|D]
  - Compare strings

REMINDER 3: Repeating using REP

- REP INSTRUCTION
  - E.g. REP MOVS

- While CX>0

  perform INSTRUCTION

  CX=CX- 1

  END

REMINDER 4: XLAT

- Uses a table to translate
- *n* is converted to [EBX+*n*]
- The input is put into AL before XLAT

```
table       BYTE   48 DUP (' '), '0123456789', 7 DUP (' ')
            BYTE   'abcdefghijklmnopqrstuvwxyz', 6 DUP (' ')
            BYTE   'abcdefghijklmnopqrstuvwxyz', 133 DUP (' ')
```

```
            mov    ecx, strLength ; string length
            lea    ebx, table     ; address of translation table
            lea    esi, string    ; address of string
            lea    edi, string    ; destination also string
forIndex:   lodsb                 ; copy next character to AL
            xlat                  ; translate character
            stosb                 ; copy character back into string
            loop   forIndex       ; repeat for all characters
```

Logical Operation

- AND destination, source
- OR destination, source
- XOR destination, source
- NOT destination

Examples

Before	*Instruction*	*Bitwise Operation*	*After*

AX: E2 75
CX: A9 D7
	`and ax,cx`	`1110 0010 0111 0101`
		`1010 1001 1101 0111`
		`1010 0000 0101 0101`
			AX A0 55
			SF 1 ZF 0

DX: E2 75
value: A9 D7
	`or dx,value`	`1110 0010 0111 0101`
		`1010 1001 1101 0111`
		`1110 1011 1111 0111`
			DX EB F7
			SF 1 ZF 0

BX: E2 75	`xor bx,0a9d7h`	`1110 0010 0111 0101`
		`1010 1001 1101 0111`
		`0100 1011 1010 0010`
			BX 4B A2
			SF 0 ZF 0

AX: E2 75	`not ax`	`1110 0010 0111 0101` `0001 1101 1000 1010`	AX 1D 8A

Shift and Rotation Instructions

	Left	Right
Logical	SHL	SHR
Arithmetic	SAL	SAR

- S* destination, count
  - Count can be immediate or CL (mod 32)
- SHL, SAL are identical
- SAR sign extends the shifted bit
- SHR zero extends the shifted bit
- The bit that goes out is written to CF

Examples

BX: A9 D7 `sar bx,1`

1010 1001 1101 0111

1101 0100 1110 1011

BX | D4 | EB

SF 1 ZF 0
CF 1 OF 0

ace: A9 D7 `sal ace,4`

1010 1001 1101 0111

1001 1101 0111 **0000**

ace | 9D | 70

SF 1 ZF 0
CF 0 OF ?

DX: A9 D7 `shr dx,4`

1010 1001 1101 0111

0000 1010 1001 1101

DX | 0A | 9D

SF 0 ZF 0
CF 0 OF ?

AX: A9 D7 `sar ax,cl`
CL: 04

1010 1001 1101 0111

1111 1010 1001 1101

AX | FA | 9D

SF 1 ZF 0
CF 0 OF ?

Put It Together

```
.DATA                                   ; reserve storage for data
prompt      BYTE    "Enter a number:  ",0
number      BYTE    20 DUP (?)
result      BYTE    cr,Lf,"The 2's complement representation is "
hexOut      BYTE    8 DUP (?),cr,Lf,0

.CODE                                   ; start of main program code
_start:
            output  prompt              ; prompt for number
            input   number,20           ; read ASCII characters
            atod    number              ; convert to integer

            lea     ebx,hexOut+7        ; address for last character
            mov     ecx,8               ; number of characters
forCount:   mov     edx,eax             ; copy pattern
            and     edx,0000000fh       ; zero all but last hex digit
            cmp     edx,9               ; digit?
            jnle    elseLetter          ; letter if not
            or      edx,30h             ; convert to character
            jmp     endifDigit
elseLetter: add     edx,'A'-10          ; convert to letter
endifDigit:
            mov     BYTE PTR [ebx],dl ; copy character to memory
            dec     ebx                 ; point at next character
            shr     eax,4               ; shift one hex digit right
            loop    forCount            ; repeat

            output  result              ; output label and hex value

            INVOKE  ExitProcess, 0    ; exit with return code 0
PUBLIC _start                           ; make entry point public
END                                     ; end of source code
```

Double Shift Instructions

- SHLD destination, source, count
  - SHL destination,count
    - but the first count bits in destination are filled from left of source

```
ECX: 12 34 56 78     shld ecx,eax,12
EAX: 90 AB CD EF
```

ECX	45	67	89	0A
EAX	90	AB	CD	EF

CF 1 ZF 0 SF 0

- SHRD destination, source, count
  - SHL destination,count
    - but the last count bits in destination are filled from left

```
ECX: 12 34 56 78     shrd ecx,eax,CL
EAX: 90 AB CD EF
CL: 08
```

ECX	EF	12	34	56
EAX	90	AB	CD	EF

CF 0 ZF 0 SF 1

Rotation

- R[O|C][R|L] dest, count
  - Count can be immediate or CL (mod 32)
  - Like shift but the dropping bits are fed to the other side
  - RO*
    - The last dropping bit is also copied to CF
  - RC*
    - Assumes the CF is added as MSB to dest and does the rotation.

Using Rotation

Number to HEX

```
                lea     ebx,hexOut          ; address for first character
                mov     ecx,8               ; number of characters
forCount:       rol     eax,4               ; rotate first hex digit to end
                mov     edx,eax             ; copy all digits
                and     edx,0000000fh       ; zero all but last hex digit
                cmp     edx,9               ; digit?
                jnle    elseLetter          ; letter if not
                or      edx,30h             ; convert to character
                jmp     endifDigit
elseLetter:     add     edx,'A' 10          ; convert to letter
endifDigit:
                mov     BYTE PTR [ebx],dl ; copy character to memory
                inc     ebx                 ; point at next character
                loop    forCount            ; repeat
```