Lecture 05: Assembly Language Programming (2)

Reference Book:

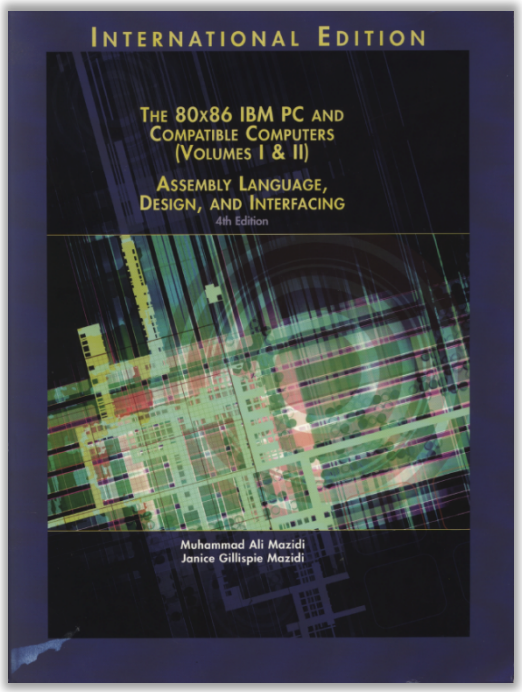
The 80x86 IBM PC and Compatible Computers

Chapter 3

Arithmetic & Logic Instructions and Programs

Chapter 6

Signed Numbers, Strings, and Tables



Arithmetic Instructions

- Addition
- Subtraction
- Multiplication
- Division

Unsigned Addition

- **ADD** dest, src ;*dest = dest + src*
  - Dest can be a register or in memory
  - Src can be a register, in memory or an immediate
  - **No mem-to-mem operations in 80X86**
  - Change ZF, SF, AF, CF, OF, PF
- ADC dest, src ;*dest = dest + src +CF*
  - For multi-byte numbers
  - If there is a carry from last addition, adds 1 to the result

Addition Example of Individual Bytes

```
TITLE     PROG3-1A  (EXE)   ADDING 5 BYTES
PAGE      60,132
MODEL SMALL
.STACK 64
;
            .DATA
COUNT       EQU     05
DATA        DB      125,235,197,91,48
            ORG     0008H
SUM         DW      ?
;
            .CODE
MAIN        PROC    FAR
            MOV     AX,@DATA
            MOV     DS,AX
            MOV     CX,COUNT        ;CX is the loop counter
            MOV     SI,OFFSET DATA  ;SI is the data pointer
            MOV     AX,00           ;AX will hold the sum
BACK:       ADD     AL,[SI]         ;add the next byte to AL
            JNC     OVER            ;if no carry, continue
            INC     AH              ;else accumulate carry in AH
OVER:       INC     SI              ;increment data pointer
            DEC     CX              ;decrement loop counter
            JNZ     BACK            ;if not finished, go add next byte
            MOV     SUM,AX          ;store sum
;
            MOV     AH,4CH
            INT     21H             ;go back to DOS
MAIN        ENDP
            END     MAIN
```

Draw the layout of the data segment in memory

What's this for?

```
BACK:  ADD     AL,[SI]
       ADC     AH,00    ;add 1 to AH if CF=1
       INC     SI
```

Addition Example of Multi-byte Nums

```
TITLE     PROG3-2 (EXE)  MULTIWORD ADDITION
PAGE      60,132
.MODEL SMALL
.STACK 64
;-----------------------------------------
          .DATA
DATA1     DQ      548FB9963CE7H
          ORG     0010H
DATA2     DQ      3FCD4FA23B8DH
          ORG     0020H
DATA3     DQ      ?
;-----------------------------------------
          .CODE
MAIN      PROC    FAR
          MOV     AX,@DATA
          MOV     DS,AX
          CLC                          ;clear carry before first addition
          MOV     SI,OFFSET DATA1      ;SI is pointer for operand1
          MOV     DI,OFFSET DATA2      ;DI is pointer for operand2
          MOV     BX,OFFSET DATA3      ;BX is pointer for the sum
          MOV     CX,04                ;CX is the loop counter
BACK:     MOV     AX,[SI]              ;move the first operand to AX
          ADC     AX,[DI]              ;add the second operand to AX
          MOV     [BX],AX              ;store the sum
          INC     SI                   ;point to next word of operand1
          INC     SI
          INC     DI                   ;point to next word of operand2
          INC     DI
          INC     BX                   ;po
          INC     BX
          LOOP    BACK                 ;if
          MOV     AH,4CH
          INT     21H                  ;go
MAIN      ENDP
          END     MAIN
```

```
LOOP   xxxx   ;is equivalent to the following two instructions

DEC    CX
JNZ    xxxx
```

Unsigned Subtraction

- **SUB** dest, src ;*dest = dest - src*
  - Dest can be a register or in memory
  - Src can be a register, in memory or an immediate
  - No mem-to-mem operations in 80X86
  - Change ZF, SF, AF, CF, OF, PF
- SBB dest, src ;*dest = dest - src – CF*
  - For multi-byte numbers
  - If there is a borrow from last subtraction, subtracts 1 from the result

Subtraction Example of Individual Bytes

- CPU carries out
  1. take the 2's complement of the *src*
  2. add it to the *dest*
  3. invert the carry

After these three steps, if

- CF = 0: positive result;

```
MOV   AL,3FH      AL     3F   0011 1111      0011 1111
MOV   BH,23H    - BH   - 23  - 0010 0011    +1101 1101   (2's complement)
SUB   AL,BH              1C                 1 0001 1100  CF=0 (step 3)
```

- CF = 1: negative result, left in 2's complement
  - Magnitude: NOT + INC (if a programmer wants the magnitude)

```
  4C   0100 1100                    0100 1100
- 6E   0110 1110   2's comp        +1001 0010
- 22                               0 1101 1110   CF=1 (step 3) the result is negative
```

Subtraction Example of Multi-byte Nums

```
DATA_A      DD  62562FAH
DATA_B      DD  412963BH
RESULT      DD  ?
...
            ...  ...
            MOV     AX,WORD PTR DATA_A
            SUB     AX,WORD PTR DATA_B
            MOV     WORD PTR RESULT,AX
            MOV     AX,WORD PTR DATA_A +2
            SBB     AX,WORD PTR DATA_B +2
            MOV     WORD PTR RESULT+2,AX
```

AX = 62FA – 963B = CCBF CF = 1

AX = 625 – 412 – 1 = 212. CF = 0

RESULT is 0212CCBF.

Unsigned Multiplication

- **MUL** *operand*
- byte X byte:
  - One implicit operand is **AL**, the other is the *operand*, result is stored in **AX**
- word X word:
  - One implicit operand is **AX**, the other is the *operand*, result is stored in **DX** & **AX**
- word X byte:
  - **AL** hold the byte, **AH = 0**, the word is the *operand*, result is stored in **DX** & **AX**;

Unsigned Multiplication Example

```
MOV     AL,DATA1
MOV     BL,DATA2
MUL     BL
MOV     RESULT,AX
```

```
MOV     AL,DATA1
MOV     SI,OFFSET DATA2
MUL     BYTE PTR [SI]
MOV     RESULT,AX
```

```
DATA3           DW      2378H
DATA4           DW      2F79H
RESULT1         DW      2 DUP(?)

MOV     AX,DATA3
MUL     DATA4
MOV     RESULT1,AX
MOV     RESULT1+2,DX
```

```
DATA5           DB      6BH
DATA6           DW      12C3H
RESULT3         DW      2 DUP(?)

MOV     AL,DATA5
SUB     AH,AH
MUL     DATA6
MOV     BX,OFFSET RESULT3
MOV     [BX],AX
MOV     [BX]+2,DX
```

Unsigned Division

- **DIV** *denominator*
  - Denominator cannot be zero
  - Quotient cannot be too large for the assigned register
- byte / byte:
  - Numerator in **AL**, clear **AH**; quotient is in **AL,** remainder in **AH**
- word / word:
  - Numerator in **AX**, clear **DX**; ; quotient is in **AX,** remainder in **DX**
- word / byte:
  - Numerator in **AX**; quotient is in **AL** (max 0FFH)**,** remainder in **AH**
- double-word / word:
  - Numerator in **DX, AX**; quotient is in **AX** (max 0FFFFH)**,** remainder in **DX**
  - Denominator can be in a register or in memory

Unsigned Division Example

```
MOV     AL,DATA7
SUB     AH,AH
DIV     10
```

```
MOV     AX,10050
SUB     DX,DX
MOV     BX,100
DIV     BX
MOV     QOUT2,AX
MOV     REMAIND2,DX
```

```
MOV     AX,2055
MOV     CL,100
DIV     CL
MOV     QUO,AL
MOV     REMI,AH
```

```
DATA1   DD      105432
DATA2   DW      10000
QUOT    DW      ?
REMAIN  DW      ?

MOV     AX,WORD PTR DATA1
MOV     DX,WORD PTR DATA1+2
DIV     DATA2
MOV     QUOT,AX
MOV     REMAIN,DX
```

Logic Instructions

- AND
- OR
- XOR
- NOT
- Logical SHIFT
- ROTATE
- COMPARE

AND

- AND dest, src
  - Bit-wise logic
  - dest can be a register or in memory; src can be a register, in memory, or immediate

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

- OR dest, src
  - Bit-wise logic
  - dest can be a register or in memory; src can be a register, in memory, or immediate

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

- XOR dest, src
  - Bit-wise logic
  - dest can be a register or in memory; src can be a register, in memory, or immediate

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

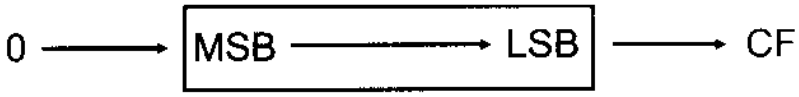
NOT

- NOT *operand*
  - Bit-wise logic
  - Operand can be a register or in memory

X	NOT X
1	0
0	1

Logical SHIFT

- SHR dest, times
  - dest can be a register or in memory
  - 0->MSB->…->LSB->CF
  - Times = 1:

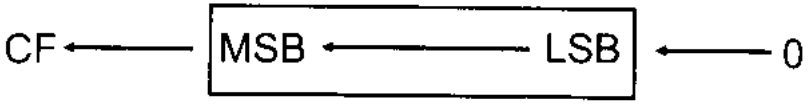



```
SHR xx, 1
```

  - Times >1:

```
MOV CL, times
SHR xx, CL
```

- SHL dest, times
  - All the same except in **reverse** direction



Example: BCD & ASCII Numbers Conversion

- BCD: Binary Coded Decimal
  - Digits 0~9 in binary representation
  - Unpacked, packed
-

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

ASCII -> Unpacked BCD Conversion

- Simply remove the higher 4 bits "0011"
- E.g.,

```
asc         DB    '3'
unpack      DB    ?
-------------------------
MOV AH, asc
AND AH, 0Fh
MOV unpack, AH
```

ASCII -> Packed BCD Conversion

- First convert ASCII to unpacked BCD
- Then, combine two unpacked into one packed
- E.g.,

```
asc         DB    '23'
unpack      DB    ?
-------------------------
MOV AH, asc
MOV AL, asc+1
AND AX, 0F0Fh
MOV CL, 4
SHL AH, CL
OR AH, AL
MOV unpack, AH
```

ROTATE

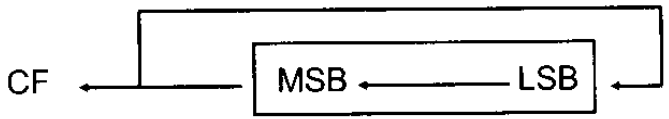
- ROR *dest, times*
  - *dest* can be a register, in memory
  - *Times* = 1:


```
ROR xx, 1
```

  - *Times* >1:

```
MOV CL, times
ROR xx, CL
```

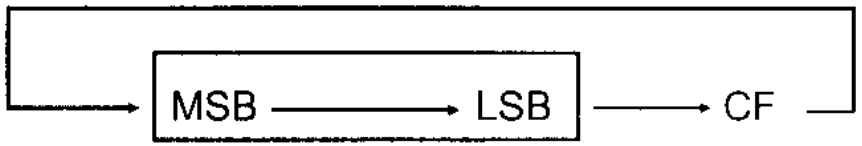
- ROL *dest, times*
  - All the same except in **reverse** direction



ROTATE Cont.

- RCR *dest, times*
  - *dest* can be a register, in memory
  - *Times* = 1:

```
RCR xx, 1
```



  - *Times* >1:

```
MCV CL, times
RCR xx, CL
```

- RCL *dest, times*
  - All the same except in **reverse** direction

COMPARE of Unsigned Numbers

- CMP dest, src
  - Flags affected as (dest – src) but operands remain unchanged

Table 3-3: Flag Settings for Compare Instruction

Compare operands	CF	ZF
destination > source	0	0
destination = source	0	1
destination < source	1	0

- E.g., `CMP AL, 23`

  `JA lable1` ; jump if above, CF = ZF = 0

Jump Based on Unsigned Comparison

These flags are based on unsigned comparison

Mnemonic	Description	Flags/Registers
JA	Jump if above op1>op2	CF = 0 and ZF = 0
JNBE	Jump if not below or equal op1 not <= op2	CF = 0 and ZF = 0
JAE	Jump if above or equal op1>=op2	CF = 0
JNB	Jump if not below op1 not <opp2	CF = 0
JB	Jump if below op1<op2	CF = 1
JNAE	Jump if not above nor equal op1< op2	CF = 1
JBE	Jump if below or equal op1 <= op2	CF = 1 or ZF = 1
JNA	Jump if not above op1 <= op2	CF = 1 or ZF = 1

COMPARE of Signed Numbers

- CMP dest, src
  - Same instruction as the unsigned case
  - but different understanding about the numbers and therefore different flags checked

destination > source	OF=SF or ZF=0
destination = source	ZF=1
destination < source	OF=negation of SF

Jump Based on Signed Comparison

These flags are based on signed comparison

Mnemonic	Description	Flags/Registers
JG	Jump if GREATER op1>op2	SF = OF AND ZF = 0
JNLE	Jump if not LESS THAN or equal op1>op2	SF = OF AND ZF = 0
JGE	Jump if GREATER THAN or equal op1>=op2	SF = OF
JNL	Jump if not LESS THAN op1>=op2	SF = OF
JL	Jump if LESS THAN op1<op2	SF <> OF
JNGE	Jump if not GREATER THAN nor equal op1<op2	SF <> OF
JLE	Jump if LESS THAN or equal op1 <= op2	ZF = 1 OR SF <> OF
JNG	Jump if NOT GREATER THAN op1 <= op2	ZF = 1 OR SF <> OF

Quiz

Given the ASCII table, write an algorithm to convert lowercase letters in a string into uppercase letters and implement your algorithm using 86 assembly language.

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Answer to Quiz

```
.MODEL SMALL
.STACK 64
;-------------------------------------
        .DATA
DATA1   DB      'mY NAME is jOe'
        ORG     0020H
DATA2   DB      14 DUP(?)
;-------------------------------------
        .CODE
MAIN    PROC    FAR
        MOV     AX,@DATA
        MOV     DS,AX
        MOV     SI,OFFSET DATA1         ;SI points to original data
        MOV     BX,OFFSET DATA2         ;BX points to uppercase data
        MOV     CX,14                   ;CX is loop counter
BACK:   MOV     AL,[SI]                 ;get next character
        CMP     AL,61H                  ;if less than 'a'
        JB      OVER            ;then no need to convert
        CMP     AL,7AH          ;if greater than 'z'
        JA      OVER            ;then no need to convert
        AND     AL,11011111B    ;mask d5 to convert to uppercase
OVER:   MOV     [BX],AL         ;store uppercase character
        INC     SI              ;increment pointer to original
        INC     BX              ;increment pointer to uppercase data
        LOOP    BACK            ;continue looping if CX > 0
        MOV     AH,4CH
        INT     21H             ;go back to DOS
MAIN    ENDP
        END     MAIN
```

XLAT Instruction & Look-up Tables

- Self-learning
  - pp. 189 in Chapter 6