Lecture 05: Assembly Language Programming (2)

Reference Book:

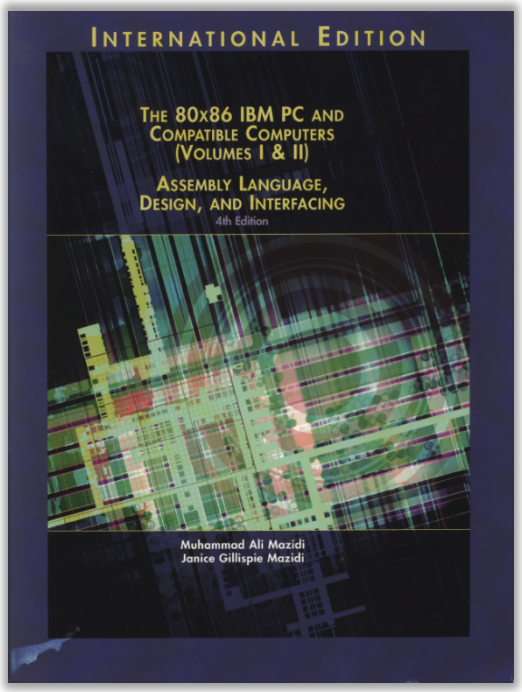
The 80x86 IBM PC and Compatible Computers

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

Unsigned Addition/Subtraction

- **INC** dest; *dest = dest + 1*
  - Dest can be a register or in memory
  - Dest **cannot** be an immediate
  - Change ZF, SF, AF, OF, PF
  - **Does not change CF**
- **DEC** dest; *dest = dest - 1*
  - Dest can be a register or in memory
  - Dest **cannot** be an immediate
  - Change ZF, SF, AF, OF, PF
  - **Does not change CF**

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
```

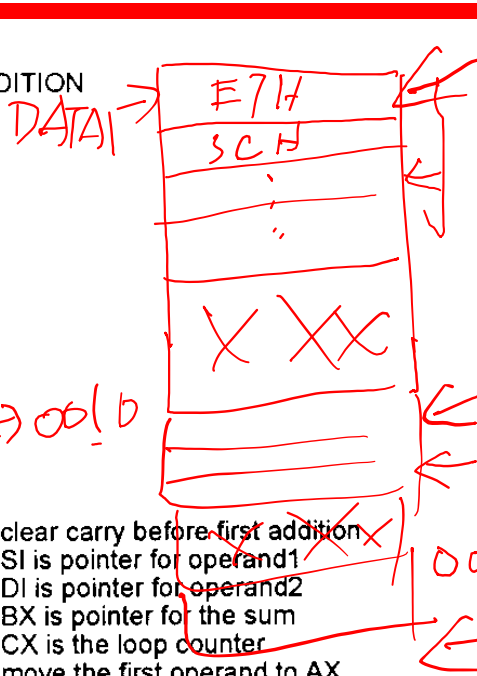
Draw the layout of the data segment in memory

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              ;go
            INT     21H
MAIN        ENDP
            END     MAIN
```



DEC CX/BX;

DATA2 => 0010

CLC => CF => 0

add SI, 2

```
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
MOV   BH,23H    - BH  - 23 - 0010 0011
SUB   AL,BH             1C
```

```
 0011 1111
+1101 1101
1 0001 1100
```

(2's complement)
CF=0 (step 3)

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

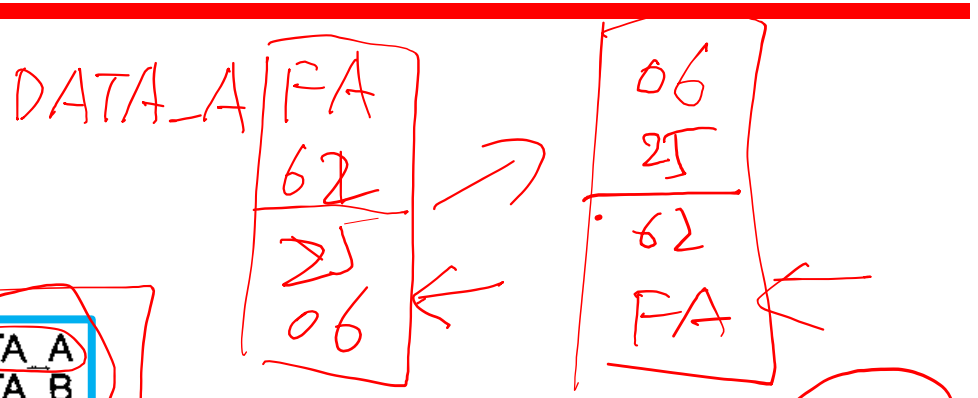
```
  4C      0100 1100
- 6E      0110 1110   2's comp
- 22
```

```
 0100 1100
+1001 0010
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.

Question: how would you change the above code if the CPU uses big endian?

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
  - Update SF, ZF, PF; AF is undetermined
  - Clear CF and OF (set CF and OF to zero) ,

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
  - Update SF, ZF, PF; AF is undetermined
  - Clear CF and OF (set CF and OF to zero)

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
  - Update SF, ZF, PF; AF is undetermined
  - Clear CF and OF (set CF and OF to zero)

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

NOT

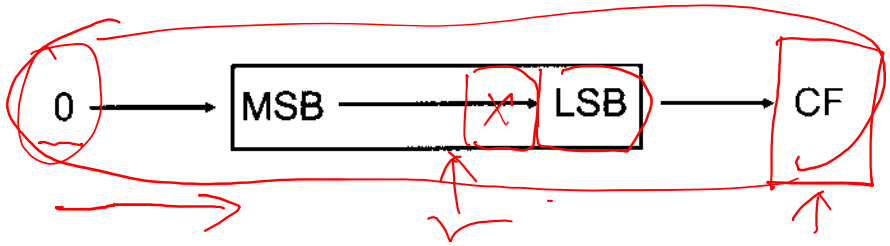
- NOT *operand*
  - Bit-wise logic
  - Operand can be a register or in memory
  - Does not change the flag register

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:

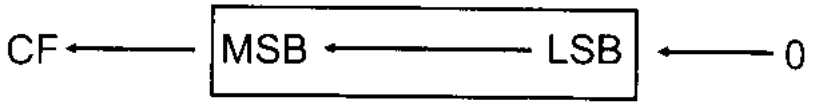
```
SHR xx, 1
```



  - 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

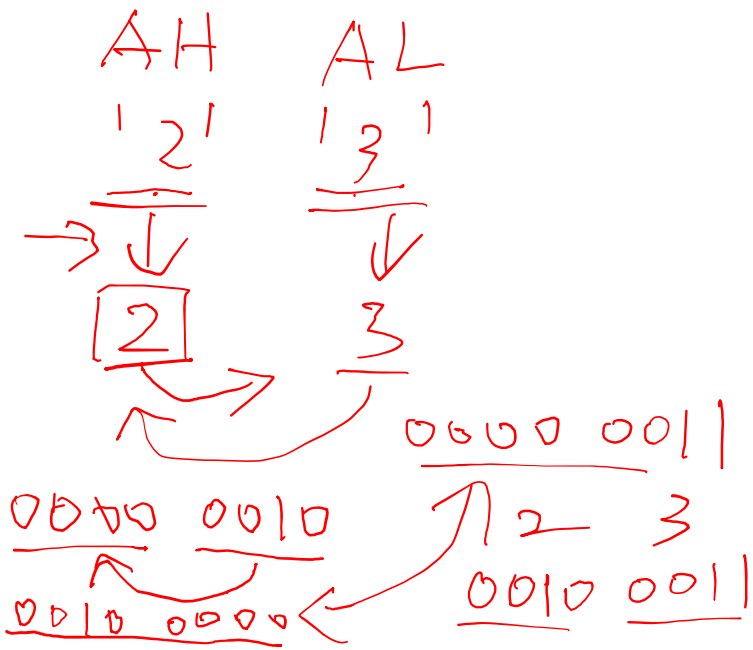
- Simply remove the higher 4 bits "0011"
- E.g.,

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

ASCII -> Packed BCD Conversion

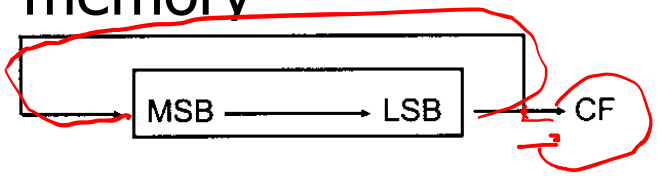
- First convert ASCII to unpacked BCD
- Then, combine two unpacked into one packed
- E.g.,

```
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:

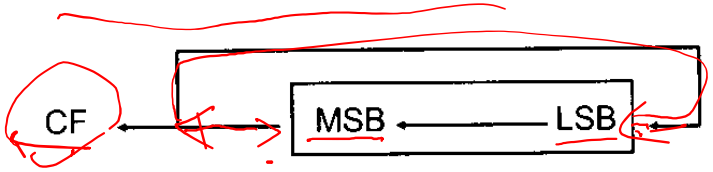



```
ROR xx, 1
```

  - *Times* >1:

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

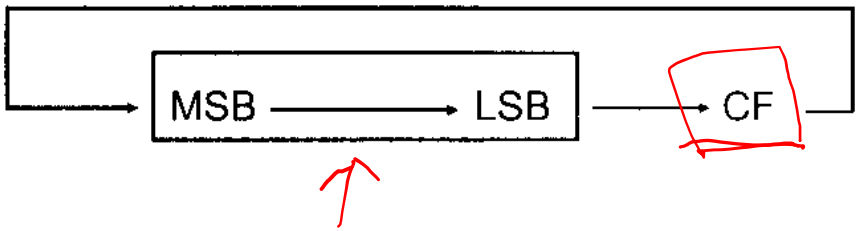
- ROL *dest, times*
  - All the same except in **reverse** direction



ROTATE Cont.

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

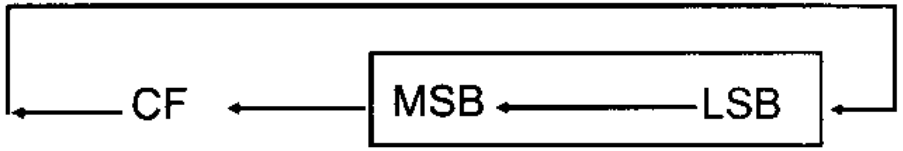
```
RCR xx, 1
```



  - *Times* >1:

```
MOV 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

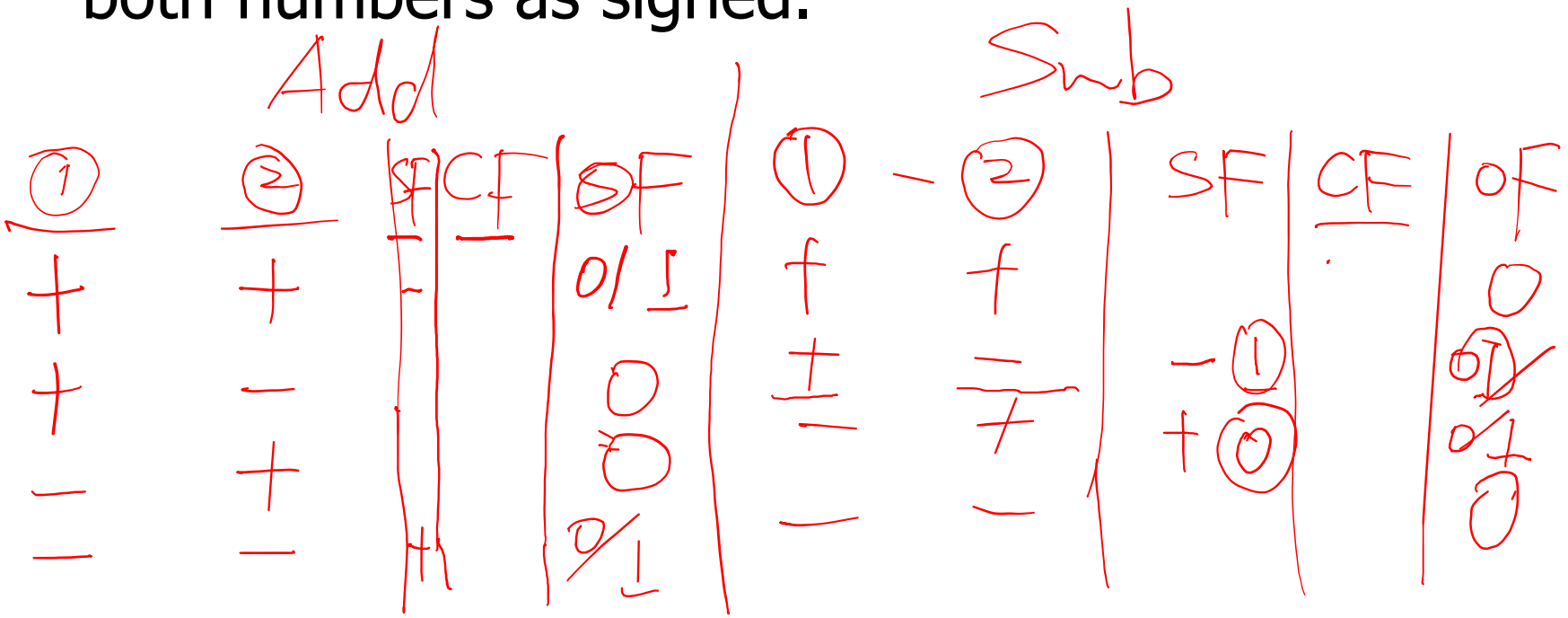
COMPARE of Signed Numbers

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

destination > source	OF == SF and ZF == 0
destination = source	ZF == 1
destination < source	OF != SF (OF == negative of SF)

Unsigned vs Signed Number

- Execution: treated as unsigned numbers
- Interpretation: CF is updated by treating both numbers as unsigned, OF is updated by treating both numbers as signed.



Jump Based on Signed Comparison

These flags are based on signed comparison

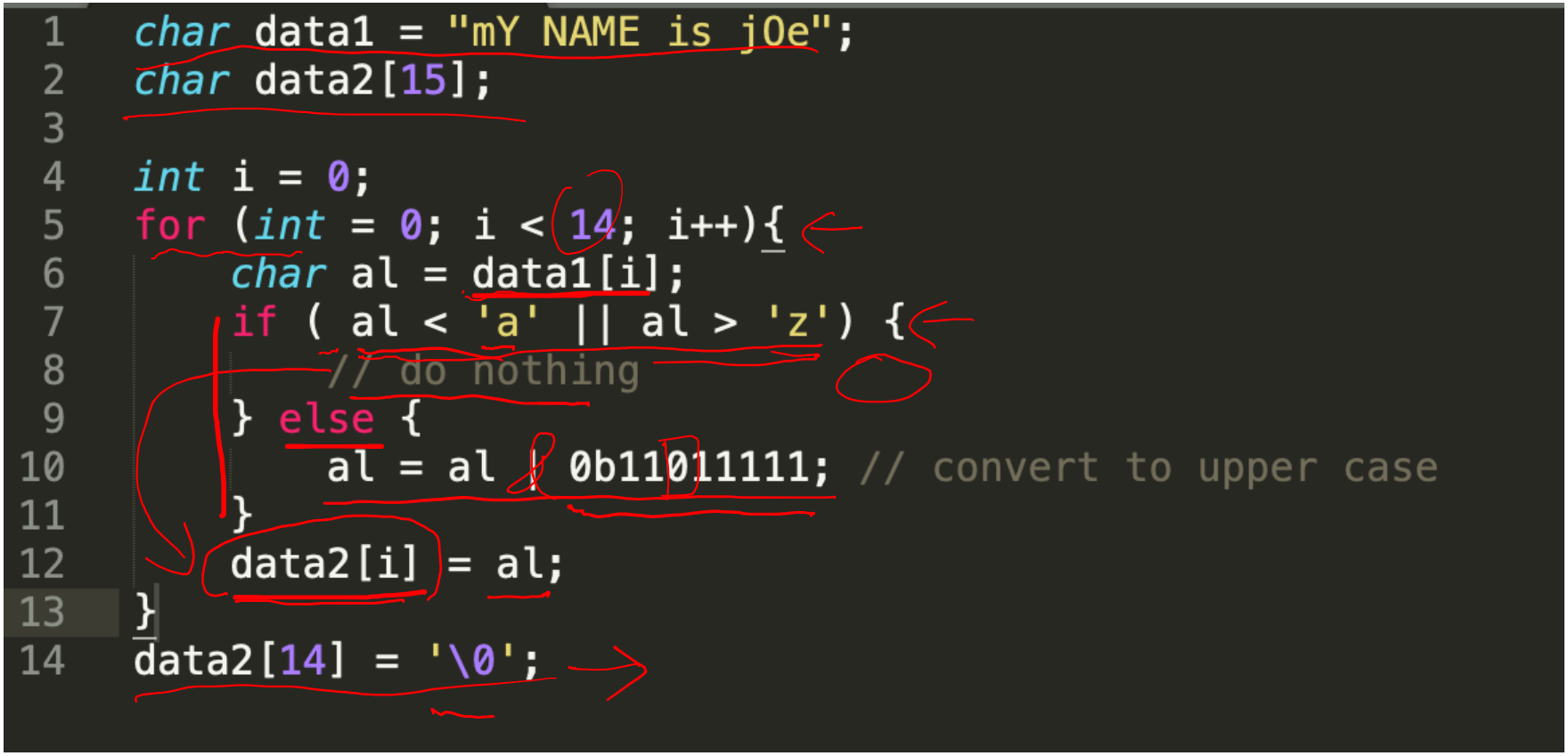
Mnemonic	Description	Flags/Registers
JG	Jump if GREATER op1>op2	SF = OF AND ZF = 0
JNLE	Jump if not LESS THAN or equal op1>op2	SF = OF AND ZF = 0
JGE	Jump if GREATER THAN or equal op1>=op2	SF = OF
JNL	Jump if not LESS THAN op1>=op2	SF = OF
JL	Jump if LESS THAN op1<op2	SF <> OF
JNGE	Jump if not GREATER 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	π	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	⌂

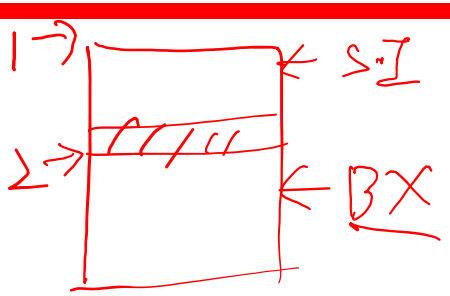
Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]



```
1   char data1 = "mY NAME is jOe";
2   char data2[15];
3
4   int i = 0;
5   for (int = 0; i < 14; i++){
6       char al = data1[i];
7       if ( al < 'a' || al > 'z') {
8           // do nothing
9       } else {
10          al = al | 0b11011111; // convert to upper case
11      }
12      data2[i] = al;
13  }
14  data2[14] = '\0';
```

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