

Lecture 20: 80x86 Assembly Programming II

Seyed-Hosein Attarzadeh-Niaki

Based on the slides by Hongzi Zhu

Microprocessors and Assembly

1

Review

- Assembly statement
- Model definition
- Segments definition
- Building programs
- Control transfer instructions
 - Short, near and far
- Data types and definition

Microprocessors and Assembly

2

Outline

- Addition and subtraction
- Multiplication and division (unsigned)
- BCD arithmetic
- Rotate instructions

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

Microprocessors and Assembly

5

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
SUM       ORG    0008H
          DW     ?
```

Exercise: Draw the layout of the data segment in memory

```
        .CODE
MAIN    PROC    FAR
        MOV     AX,@DATA
        MOV     DS,AX
        MOV     CX,COUNT
        MOV     SI,OFFSET DATA
        MOV     AX,00
BACK:    ADD     AL,[SI]
        JNC     OVER
        INC     AH
OVER:    INC     SI
        DEC     CX
        JNZ     BACK
        MOV     SUM,AX
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
        END     MAIN
```

What's this for?

CX is the loop counter
SI is the data pointer
AX will hold the sum
add the next byte to AL
if no carry, continue
else accumulate carry in AH
increment data pointer
decrement loop counter
if not finished, go add next byte
store sum
go back to DOS

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

Microprocessors and Assembly

6

Addition Example of Multi-byte Nums

```

TITLE    PROG3-2 (EXE) MULTIWORD ADDITION
PAGE     60,132
MODEL    SMALL
STACK    64

;-----
; DATA
DATA1    DQ    548FB9963CE7H
DATA2    DQ    3FCD4FA23B8DH
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     DI                               ;point to next word of operand2
INC     DI                               ;point to next word of operand2
INC     BX                               ;point to next word of sum
INC     BX
LOOP    BACK                            ;if not finished, continue adding
MOV     AH,4CH                          ;go back to DOS
INT     21H
ENDP
MAIN     END

```

LOOP xxxx ;is equivalent to the following two instructions

```

DEC     CX
JNZ     xxxx

```

Microprocessors and Assembly 7

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
 - take the 2's complement of the *src*
 - add it to the *dest*
 - 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   1001 1100      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  CF=1 (step 3) the result is negative
-22  0110 1110      0 1101 1110

```

Microprocessors and Assembly

9

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.

Microprocessors and Assembly

10

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**;

Microprocessors and Assembly

11

Unsigned Multiplication Example

- Byte x byte (reg addr)

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

- Byte x byte (reg ind)

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

- Word x word

```
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
```

Microprocessors and Assembly

12

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

Microprocessors and Assembly

13

Unsigned Division Example

• Byte/byte

```
MOV    AL,DATA7
SUB     AH,AH
DIV     DATA8
MOV     QOUT1,AL
MOV     REMAIN1,AH
```

• Word/word

```
MOV     AX,10050
SUB     DX,DX
MOV     BX,100
DIV     BX
MOV     QOUT2,AX
MOV     REMAIN2,DX
```

• Word/byte

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

• Dword/word

```
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
```

Microprocessors and Assembly

14

Signed Multiplication and Division (Read the Book)

• IMUL

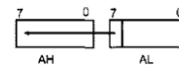
Multiplication	Operand 1	Operand 2	Result
byte × byte	AL	register or memory	AX ¹
word × word	AX	register or memory	DXAX ²
word × byte	AL=byte CBW	register or memory	DXAX ²

Notes:

1. CF = 1 and OF = 1 if AH has part of the result, but if the result is not large enough to need the AH, the sign bit is copied to the unused bits and the CPU makes CF = 0 and OF = 0 to indicate that.
2. CF = 1 and OF = 1 if DX has part of the result, but if the result is not large enough to need the DX, the sign bit is copied to the unused bits and the CPU makes CF = 0 and OF = 0 to indicate that.

One can use the J condition to find out which of the conditions above has occurred. The rest of the flags are undefined.

CBW will copy D7 (the sign flag) to all bits of AH.



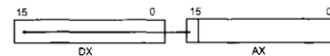
• IDIV

Division	Numerator	Denominator	Quotient	Remainder
byte/byte	AL = byte CBW	register or memory	AL	AH
word/word	AX = word CWD	register or memory	AX	DX
word/byte	AX = word	register or memory	AL ¹	AH
doubleword/word	DXAX = doubleword	register or memory	AX ²	DX

Notes:

1. Divide error interrupt if $-127 > AL > +127$.
2. Divide error interrupt if $-32767 > AX > +32767$.

CWD sign extends AX. It copies D15 of AX to all bits of the DX register.



Microprocessors and Assembly

15

Logic Instructions

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

Microprocessors and Assembly

16

Basic Logic Operations

- AND dest, src
- OR dest, src
- XOR dest, src
- NOT *operand*
 - Bit-wise logic
 - Operand can be a register or in memory
- Bit-wise logic
- dest can be a register or in memory; src can be a register, in memory, or immediate

Microprocessors and Assembly

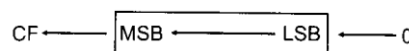
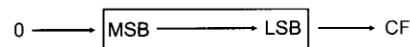
17

Logical SHIFT

- SHR dest, times
 - dest can be a register or in memory
 - 0->MSB->...->LSB->CF
 - Fixed:


```
SHR xx, 1
```
 - Variable:


```
MOV CL, times
SHR xx, CL
```
- SHL dest, times
 - All the same except in **reverse** direction



Microprocessors and Assembly

18

Example: BCD & ASCII Numbers Conversion

- BCD: Binary Coded Decimal
 - Digits 0~9 in binary representation
 - Unpacked, packed
- ASCII
 - Code for all characters that you can read or write
 - Digit characters '0'~'9'

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

Microprocessors and Assembly

19

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
```

Microprocessors and Assembly

20

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
```

Microprocessors and Assembly

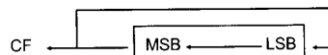
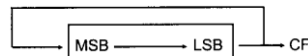
21

ROTATE

- ROR *dest, times*
 - *dest* can be a register, in memory
 - Fixed:


```
ROR xx, 1
```
 - Variable:


```
MOV CL, times
ROR xx, CL
```
- ROL *dest, times*
 - All the same except in **reverse** direction



Microprocessors and Assembly

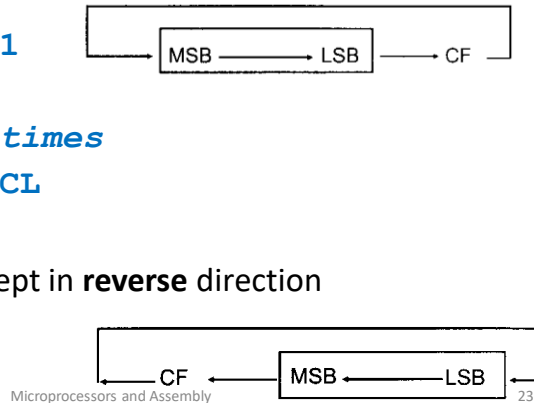
22

ROTATE Cont.

- RCR *dest, times*
 - *dest* can be a register, in memory
 - Fixed:

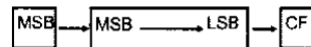

```
RCR xx, 1
```
 - Variable:


```
MOV CL, times
RCR xx, CL
```
- RCL *dest, times*
 - All the same except in **reverse** direction



Arithmetic Shift

- Used for signed numbers
 - the sign bit is copied to the shifted bits
- SAR (shift arithmetic right)
 - SAR *destination, count*
- SAL (shift arithmetic left)
 - Same as SHL (shift left)



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 label1 ; jump if above, CF = ZF = 0

Jump 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 <op2	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

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

Example:

Given the ASCII table, write an algorithm to convert lowercase letters in a string into uppercase letters and implement your algorithm using 8086 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	☐

29

Answer

```

.MODEL SMALL
.STACK 64

;-----
; DATA
DATA1 DB 'mY NAME is jOe'
        ORG 0020H
DATA2 DB 14 DUP(?)
;-----

; CODE
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 ;go back to DOS
      INT 21H
MAIN ENDP
END MAIN

```

BCD Arithmetic

Addition

- Addition of two BCD numbers is not a BCD!
- **DAA** (decimal adjust for addition) fixes this!

Subtraction

- Similar (but different) problem
- **DAS**

DATA1 DB 47H	Hex	BCD
DATA2 DB 25H	29	0010 1001
DATA3 DB ?	+ 18	+ 0001 1000
MOV AL,DATA1	41	0100 0001
MOV BL,DATA2		
ADD AL,BL	+ 6	+ 0110
DAA		
MOV DATA3,AL	47	0100 0111

Read the book for
BCD multiplication
and division.

Microprocessors and Assembly

31

Self-Learning

- Signed arithmetic
- XLAT Instruction & Look-up Tables
- Chapter 6

Microprocessors and Assembly

32

Next Lecture

- Memory and IO interfacing