Computer Organization &
Assembly Language Programming

CSE 2312
Lecture 26 Loop and Boolean Operations

Junzhou Huang, Ph.D.
Department of Computer Science and Engineering
NEG (negate) Instruction

- NEG
  - Reverses the sign of an operand.
  - Operand can be a register or memory operand.

```assembly
.SECT .DATA
valB BYTE -1
valW WORD +32767

.SECT .TEXT
  MOV AL,valB ! AL = -1
  NEG AL ! AL = +1
  NEG valW ! valW = -32767
```
NEG Instruction and the Flags

- The processor implements NEG using the following internal operation:

  \[
  \text{SUB} \ 0, \text{operand}
  \]

  Any nonzero operand causes the Carry flag to be set.

  .SECT .DATA
  valB BYTE 1

  .SECT .TEXT
  NEG valB ! CF = 1
  NEG [valB + 1] ! CF = 0
Implementing Arithmetic Expressions

• Please translate mathematical expressions into assembly language. For example:

\[
Rval = -Xval + (Yval - Zval)
\]

Rval WORD ?
Xval WORD 26
Yval WORD 30
Zval WORD 40

.CODE

SECT .TEXT
    MOV AX, Xval
    NEG AX ; AX = -26
    MOV BX, Yval
    SUB BX, Zval ; BX = -10
    ADD AX, BX
    MOV Rval, AX ; -36
Your Turn

• Translate the following expression into assembly language. Do not permit Xval, Yval, or Zval to be modified:

\[ Rval = Xval - (-Yval + Zval) \]

Assume that all values are words.

```
MOV BX,Yval
NEG BX
ADD BX,Zval
MOV AX,Xval
SUB AX,BX
MOV Rval,Ax
```
INC and DEC Instructions

- **INC and DEC**
  - Add 1, subtract 1 from destination operand
  -Operand may be register or memory

- **INC destination**
  - Logic: \( \text{destination} \leftarrow \text{destination} + 1 \)

- **DEC destination**
  - Logic: \( \text{destination} \leftarrow \text{destination} - 1 \)
INC and DEC Example

```assembly
.SECT .DATA
myWord   WORD 1000h
myDword  DW 10000000h

.SECT .TEXT
INC myWord   ! 1001h
DEC myWord   ! 1000h
INC myDword  ! 10000001h

MOV AX,00FFh
INC AX       ! AX = 0100h
MOV AX,00FFh
INC AL       ! AX = 0000h
```
Your Turn

• Show the value of the destination operand after each of the following instructions executes:

```assembly
.SECT .DATA
myByte BYTE 0FFh, 0

.SECT .TEXT
MOV AL, myByte ! AL = FFh
MOV AH, [myByte+1] ! AH = 00h
DEC AH ! AH = FFh
INC AL ! AL = 00h
```
SHL Instructions

• **SHL**
  – The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.

  ![Diagram of SHL operation]

  – Operand types for SHL:

    | Instruction          | Description                  |
    |----------------------|------------------------------|
    | **SHL reg, imm8**    | Shift destination register   |
    | **SHL mem, imm8**    | Shift memory operand, imm8   |
    | **SHL reg, CL**      | Shift destination register, CL|
    | **SHL mem, CL**      | Shift memory operand, CL      |

  (Same for all shift and rotate instructions)
**SHR Instructions**

- **SHR**
  - The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.

  ![Shift Right Diagram](image)

  Shifting right \(n\) bits divides the operand by \(2^n\)

```
MOV DL, 80
SHR DL, 1 ; DL = 40
SHR DL, 2 ; DL = 10
```
Your Turn

• Indicate the hexadecimal value of AL after each shift:

\[
\begin{align*}
\text{MOV AL, } & 6Bh & ! \quad AL = 6Bh \\
\text{SHR AL, } & 1 & ! \quad AL = 35h \\
\text{SHL AL, } & 3 & ! \quad AL = A8h \quad (1A8)
\end{align*}
\]
Binary Multiplication

- **Binary Multiplication**
  - Multiply 123 * 36

  - We already know that SHL performs unsigned multiplication efficiently when the multiplier is a power of 2.
  - You can factor any binary number into powers of 2.

- **Fast Computation**
  - Multiply AX * 36, factor 36 into \(32 + 4 = 2^5 + 2^2\) and use the distributive property of multiplication to carry out the operation:
  - Shifting left 1 bit multiplies a number by 2

\[
\begin{array}{c}
01111011 & 123 \\
\times 00100100 & 36 \\
\hline
01111011 & 123 \text{ SHL 2} \\
+ 01111011 & 123 \text{ SHL 5} \\
\hline
0001000101001100 & 4428
\end{array}
\]

Before: \(000000101\) = 5
After: \(00001010\) = 10
Example

• For example
  – Multiply AX * 36, factor 36 into 32 + 4 = 2^5 + 2^2
  – Use the distributive property of multiplication to carry out the operation:

\[
\begin{align*}
AX \times 36 & = AX \times (32 + 4) \\
& = (AX \times 32) \times (AX \times 4)
\end{align*}
\]

\[
\begin{array}{l}
\text{MOV AX,123} \\
\text{MOV BX, AX} \\
\text{SHL AX,5} \quad \text{! mult by } 2^5 \\
\text{SHL BX,2} \quad \text{! mult by } 2^2 \\
\text{ADD AX, BX}
\end{array}
\]
Your Turn

• Multiply AX by 26, using shifting and addition instructions.

  – Hint: 26 = 16 + 8 + 2 = $2^4 + 2^3 + 2$.

```assembly
MOV AX, 2 ! test value
MOV DX, AX
SHL DX, 4 ! AX * 16
PUSH EDX ! save for later
MOV DX, AX
SHL DX, 3 ! AX * 8
SHL AX, 1 ! AX * 2
ADD AX, DX ! AX * 10
POP EDX ! recall AX * 16
ADD AX, DX ! AX * 26
```
Loop Instruction

• **JMP**
  – The LOOP instruction creates a counting loop
  – Syntax: LOOP target

• **Logic**
  – ECX ← ECX – 1
  – if ECX != 0, jump to target

• **Implementation**
  – The assembler calculates the distance, in bytes, between the offset of the following instruction and the offset of the target label. It is called the relative offset.
  – The relative offset is added to EIP.
Loop Example

- **Example**
  - The following loop calculates the sum of the integers 5 + 4 + 3 + 2 + 1:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Machine code</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>66 B8 0000</td>
<td>MOV AX,0</td>
</tr>
<tr>
<td>00000004</td>
<td>B9 00000005</td>
<td>MOV ECX,5</td>
</tr>
<tr>
<td>00000009</td>
<td>66 03 C1</td>
<td>L1:ADD AX,CX</td>
</tr>
<tr>
<td>0000000C</td>
<td>E2 FB</td>
<td>LOOP L1</td>
</tr>
<tr>
<td>0000000E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When LOOP is assembled, the current location = 0000000E (offset of the next instruction). –5 (FBh) is added to the current location, causing a jump to location 00000009:

00000009 ← 0000000E + FB
Your Turn

What will be the final value of AX?
10

How many times will the loop execute?
4,294,967,296
Summing an Integer Array

The following code calculates the sum of an array of 16-bit integers.

```assembly
.SECT .DATA
intarray WORD 100h,200h,300h,400h

.SECT .TEXT
    MOV EDI,OFFSET intarray ; address of intarray
    MOV ECX,LENGTHOF intarray ; loop counter
    MOV AX,0 ; zero the accumulator
    L1:
        ADD AX,[EDI] ; add an integer
        ADD EDI,TYPE intarray ; point to next integer
        LOOP L1 ; repeat until ECX = 0
```
Boolean and Comparison Instructions

- CPU Status Flags
- AND Instruction
- OR Instruction
- XOR Instruction
- NOT Instruction
- Applications
- TEST Instruction
- CMP Instruction
Status Flags - Review

• **The Zero flag**
  – Set when the result of an operation equals zero.

• **The Carry flag**
  – Set when an instruction generates a result that is too large (or too small) for the destination operand.

• **The Sign flag**
  – Set if the destination operand is negative, and it is clear if the destination operand is positive.

• **The Overflow flag**
  – Set when an instruction generates an invalid signed result (bit 7 carry is XORed with bit 6 Carry).

• **The Parity flag**
  – Set when an instruction generates an even number of 1 bits in the low byte of the destination operand.

• **The Auxiliary Carry flag**
  – Set when an is set when an operation produces a carry out from bit 3 to bit 4
AND Instruction

- **AND**
  - Performs a Boolean AND operation between each pair of matching bits in two operands
  - Syntax: `AND destination, source`
OR Instruction

- **OR**
  - Performs a Boolean OR operation between each pair of matching bits in two operands
  - Syntax: **OR destination, source**

```
0 0 1 1 1 0 1 1
OR 0 0 0 0 1 1 1 1
unchanged ——— 0 0 1 1 1 1 1 1
set
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ∨ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
XOR Instruction

- **XOR**
  - Performs a Boolean exclusive-OR operation between each pair of matching bits in two operands
  - Syntax: XOR destination, source
  - XOR is a useful way to toggle (invert) the bits in an operand.

XOR Instruction Table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ⊕ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR Example:

```
0 0 1 1 0 1 1
XOR 0 0 0 1 1 1 1
```

unchanged 0 0 1 1 0 1 0 0
inverted

```
XOR
```
NOT Instruction

- NOT
  - Performs a Boolean NOT operation on a single destination operand
  - Syntax: \textbf{NOT destination}

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{X} & \textbf{\neg X} \\
\hline
\text{T} & \text{F} \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{l}
\textbf{NOT} \quad 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \\
\hline
\text{1} \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \quad \text{inverted}
\end{tabular}
\end{center}
Bit-Mapped Set

• Set Complement
  MOV AX, SetX
  NOT AX

• Set Intersection
  MOV AX, SetX
  AND AX, SetY

• Set Union
  MOV AX, SetX
  OR AX, SetY
Applications (1 of 5)

• Task
  – Binary bits Convert the character in AL to upper case.

• Solution
  – Use the AND instruction to clear bit 6.

  MOV AL, 'a'
  ! AL = 01100001b
  AND AL, 11011111b
  ! AL = 01000001b
Applications (2 of 5)

• Task
  – Convert a binary decimal byte into its equivalent ASCII decimal digit.

• Solution
  – Use the OR instruction to set bits 5 and 6.

\[
\begin{align*}
\text{MOV AL, 6} & \quad ! \ AL = 00000110b \\
\text{OR AL, 00110000b} & \quad ! \ AL = 00110110b
\end{align*}
\]

The ASCII digit '6' = 00110110b
Applications (3 of 5)

• Task
  – Turn on the keyboard CapsLock key

• Solution
  – Use the OR instruction to set bit 6 in the keyboard flag byte at 0040:0017h in the BIOS data area.

```
    MOV AX, 40h            ! BIOS segment
    MOV DS, AX
    MOV BX, 17h            ! keyboard flag byte
    OR BYTE PTR [BX], 01000000b ! CapsLock on
```

This code only runs in Real-address mode, and it does not work under Windows NT, 2000, or XP.
Test Instruction

• TEST
  – Performs a nondestructive AND operation between each pair of matching bits in two operands
  – No operands are modified, but the Zero flag is affected.

• Example
  – jump to a label if either bit 0 or bit 1 in AL is set.

\[
\text{TEST AL,00000011b} \\
\text{JNZ ValueFound}
\]

• Example
  – jump to a label if neither bit 0 nor bit 1 in AL is set.

\[
\text{TEST AL,00000011b} \\
\text{JZ ValueNotFound}
\]
Please bring your computer next class!