Basic Assembly

Bitwise Instructions

Assembly language programming
By xorpd

xorpd.net

Objectives

- We will learn about the Logical instructions:
  - NOT, AND, OR, XOR.
- We will learn about some instructions that move bits around:
  - Simple bit shifting.
  - Shifting with respect to the sign.
  - Rotating bits.

Logical Instructions

- So far we have considered the bits inside our registers to have numerical meaning.
  - Unsigned or Signed numbers.
- We could think about other meanings to bits:
  - **True for 1, and False for 0**. (Boolean values)
- We would like to perform meaningful operations between values of **True** and **False**.

NOT

NOT	
0	1
1	0

- NOT dest
- The NOT instruction allows to “flip” every bit.
  - Changes 1 into 0, and 0 into 1.
  - Unary operation – Works on one bit every time.
- Examples:
  - not eax
  - not ch

```
mov     ch, 10011101b
not     ch
```

1	0	0	1	1	1	0	1
0	1	1	0	0	0	1	0

- Not the same as the NEG instruction.
  - (The NEG instruction also adds 1 after the bit flip !)

AND

AND	0	1
0	0	0
1	0	1

- AND dest,src
- Binary operation
  - (produces one bit from every two bits).
  - Produces 1 if first argument is 1, **and** second argument is 1.
  - Produces 0 otherwise.
- Example:

```
mov     al,11110000b
mov     dh,11001100b
and     al,dh

; al == 11000000b
```

1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	0
1	1	0	0	0	0	0	0

OR

OR	0	1
0	0	1
1	1	1

- OR dest,src
- Binary Operation
  - Produces 1 if first argument is 1, **or** second argument is 1.
  - Produces 0 otherwise.
- Example:

```
mov       al,11110000b
mov       dh,11001100b
or        al,dh

; al == 11111100b
```

1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	0
1	1	1	1	1	1	0	0

XOR

XOR	0	1
0	0	1
1	1	0

- XOR dest,src
- Binary Operation
  - Produces 1 if first argument is 1, **or** second argument is 1, but **not both** at the same time. (Exclusive)
- Example:

```
mov     al,11110000b
mov     dh,11001100b
xor     al,dh

; al == 00111100b
```

1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	0
0	0	1	1	1	1	0	0

Zeroing using XOR

XOR	0	1
0	0	1
1	1	0

- The following piece of code is very common:

```
xor      eax,eax
; eax == 0
```

- Used because `xor eax,eax` has shorter encoding than `mov eax,0`.

Encoding	Instruction
31 c0	**xor** eax,eax
B8 00 00 00 00	mov eax,0

Bit Shifting

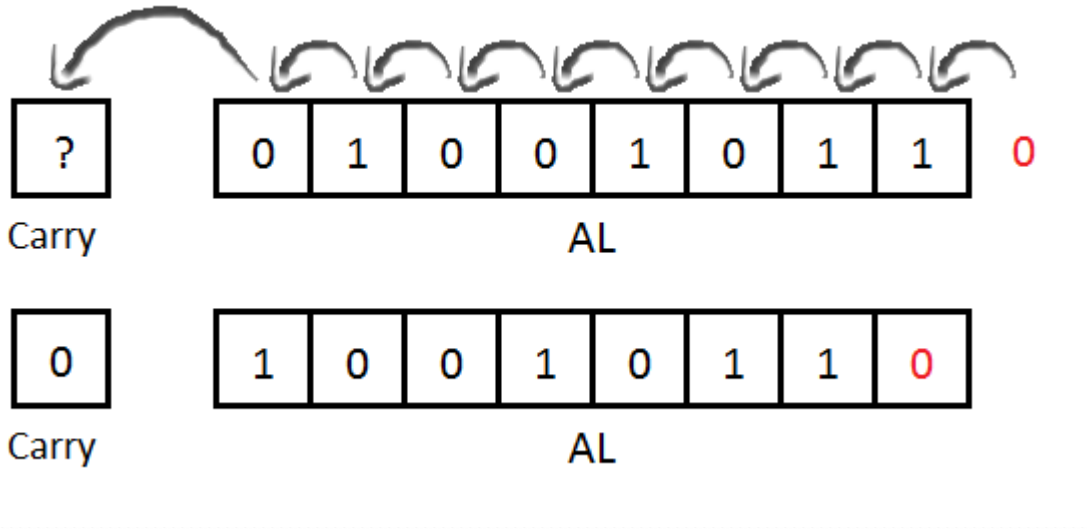
- So far we could only invoke boolean operators on "parallel" bits.
  - Example: Bit 3 of eax together with bit 3 of edx.
  - What if we want to AND bit 3 of eax and bit 5 of edx?
- We want to have finer control over individual bits.
- We will learn how to move bits around.

SHL,SHR

- Shift Left and Shift Right.
- **SHL dest,k**
  - Shift the bits inside dest k bits to the left. Insert zeroes from the right.
- **SHR dest,k**
  - Shifts the bits inside dest k bits to the right. Insert zeroes from the left.
- Example:

```
mov     al,01001011b
shl     al,1

; al == 10010110b
; CF == 0
```



SHL,SHR (Cont.)

- Boundaries:
  - The Carry flag will contain the last bit that was “kicked out” of the boundary.
  - The new inserted bit will be 0.
- Some constraints:
  - The k argument can only be:
    - A small number.
    - The CL register.
      - `shr eax,dh` will not assemble. (“Invalid Operand”)
      - `shr eax,cl` will assemble.

SHL,SHR (Example)

```
mov     dl,01001100b
mov     cl,1
shr     dl,cl

; dl == 00100110
; CF == 0

shr     dl,2
; dl == 00001001
; CF == 1

shl     dl,3
; dl == 01001000
; CF == 0
```

Arithmetic shifting

- In the unsigned numbers world:
  - Left shift is multiplication by 2.
    - In base 10, left shift is multiplication by 10!
  - Right shift is division by 2.

- Signed arithmetic using shifts:
  - Examples:

```
mov     bl,0xFB ; -0x5
shl     bl,1
; bl == 0xF6 == -0xa
; shl multiplied by 2!
```

```
mov     bl,0xFB ; -0x5
shr     bl,1
; bl == 0x7d
; shr didn't calculate
; division correctly.
```

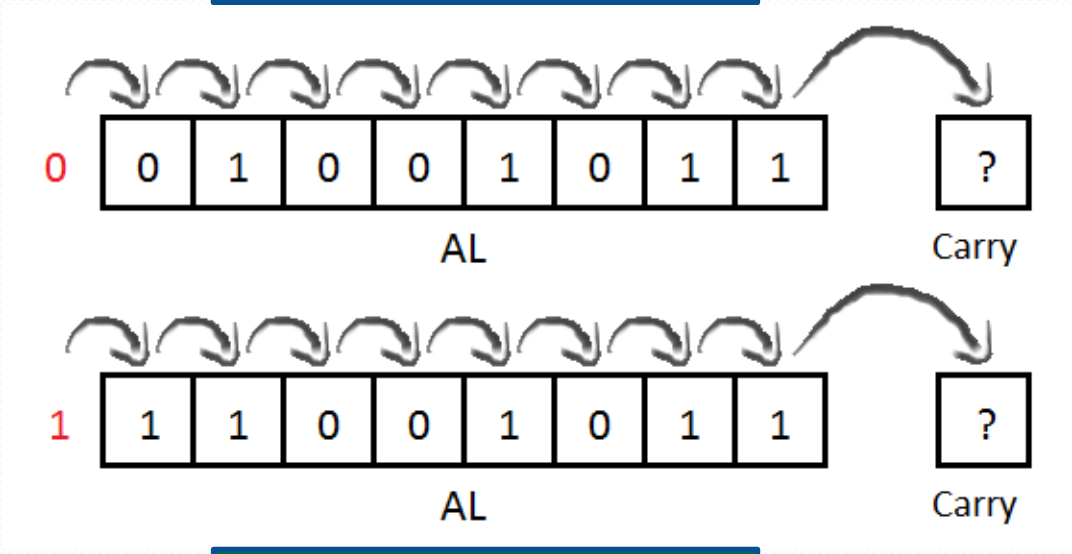
  - We can multiply by 2 using SHL , even with signed numbers.
  - SHR doesn't divide by 2 for signed negative numbers.
    - We should find an alternative.

SAL,SAR

- SHR inserts zeroes from the left.
  - Maybe we should insert 1's instead when a negative number is given?
    - To preserve the sign bit of the original number.
- **SAR dest,k**
  - Shift the bits of dest k bits to the right.
  - Insert ones or zeroes from the left, according to the original sign bit of dest.
    - For positive numbers: Insert zeroes, just like in SHR.
    - For negative numbers: Insert ones.
- **SAL dest,k**
  - Just another name for the SHL instruction.

SAR (Example)

```
mov     al,01001011b
sar     al,1
; al == 00100101
; CF == 1
```




```
mov     al,11001011b
sar     al,1
; al == 11100101
; CF == 1
```

ROL,ROR

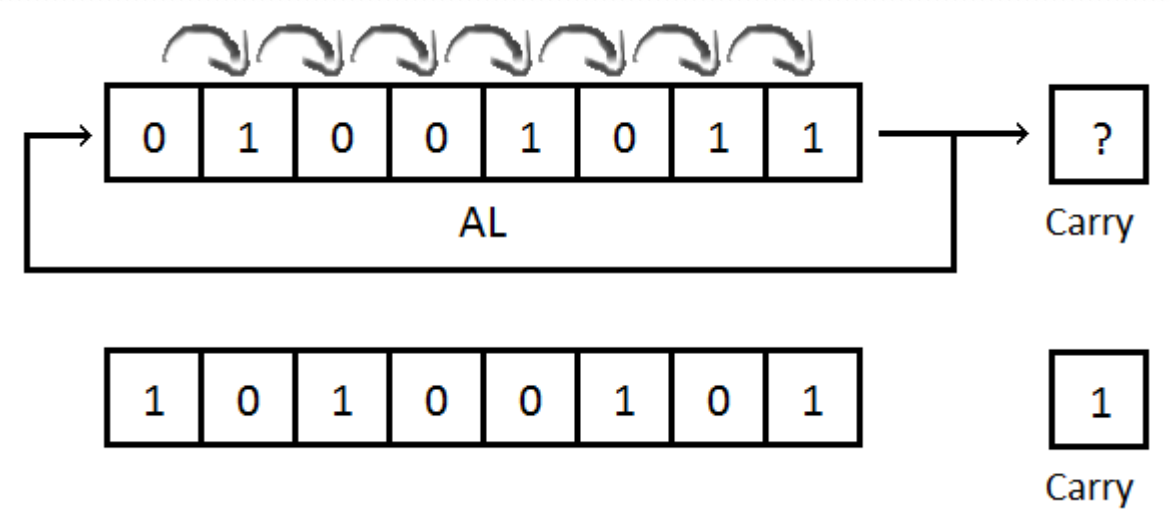
- Rotate Left, Rotate Right.
- **ROL dest,k**
  - Rotate the bits of dest k times to the left.
- **ROR dest,k**
  - Rotate the bits of dest k times to the right.

- The carry flag contains a copy of the last bit which was shifted from one end to another.

- k should be one of the following:
  - A small number.
  - CL register.

ROL,ROR (Example)

```
mov     al,01001011b
ror     al,1
; al == 10101101
; CF == 1
```



Summary

- Logical instructions:
  - NOT, AND, OR, XOR.

- Bit Shifting instructions:
  - SHL, SHR – Simple bit shifting.
  - SAL, SAR – Arithmetic shifting for signed numbers.
  - ROL, ROR – Rotate bits.