

# Introduction to 8086 Assembly

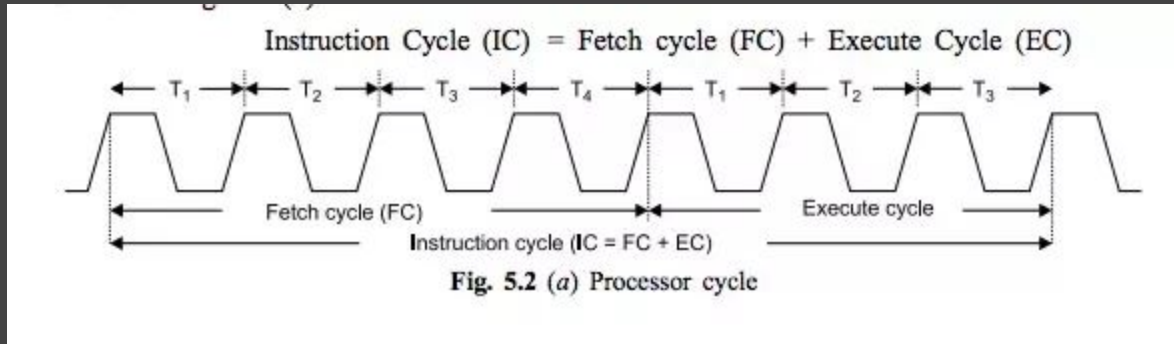
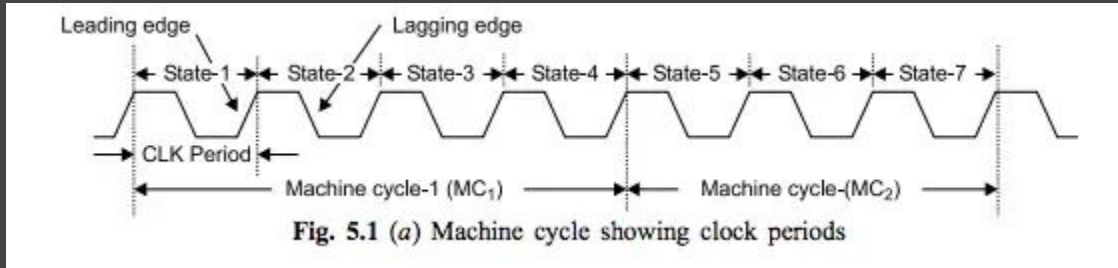
## Lecture 8

### Bit Operations

# Clock cycle & instruction timing



K. N. Toosi  
University of Technology



# Clock cycle & instruction timing



K. N. Toosi  
University of Technology

See

- 8086: [http://www.oocities.org/mc\\_introtocomputers/Instruction\\_Timing.PDF](http://www.oocities.org/mc_introtocomputers/Instruction_Timing.PDF)
- <https://zsmith.co/intel.html> ([http://zsmith.co/intel\\_a.html](http://zsmith.co/intel_a.html))
- [http://www.agner.org/optimize/instruction\\_tables.pdf](http://www.agner.org/optimize/instruction_tables.pdf)



# Logical shift (unsigned shift)

*SHL* *reg/mem*, *immed8/CL* (shift left, MSB -> CF)

*SHR* *reg/mem*, *immed8/CL* (shift right, LSB -> CF)

```
mov ax, 0A74Ch
```

```
shl ax, 1
```

```
shr ax, 3
```

```
mov cl, 10
```

```
shl eax, cl
```



# Logical shift (unsigned shift)

**SHL**: shift left

**SHR**: shift right

original number	mov al, 11011001b	1	1	0	1	1	0	0	1	
shift 1 bit left	SHL al, 1	1	0	1	1	0	0	1	0	al <<= 1

original number	mov al, 11011001b	1	1	0	1	1	0	0	1	
shift 1 bit right	SHR al, 1	0	1	1	0	1	1	0	0	al >>= 1

# Example: Shift left



```
segment .data                                simple_shl.asm
msg:  db "shift must be <= 32", 10, 0

segment .text
:
call read_int
mov ebx, eax

call read_int
cmp eax, 32
ja err_lbl
```

```
simple_shl.asm (cont.)

mov cl, al

shl ebx, cl
mov eax, ebx

call print_int
call print_nl

jmp endl

err_lbl:
mov eax, msg
call print_string

endl:
```

# Example: Shift right



```
segment .data
```

```
simple_shr.asm
```

```
msg: db "shift must be <= 32", 10, 0
```

```
segment .text
```

```
:
```

```
call read_int
```

```
mov ebx, eax
```

```
call read_int
```

```
cmp eax, 32
```

```
ja err_lbl
```

```
simple_shr.asm (cont.)
```

```
mov cl, al
```

```
shr ebx, cl
```

```
mov eax, ebx
```

```
call print_int
```

```
call print_nl
```

```
jmp endl
```

```
err_lbl:
```

```
mov eax, msg
```

```
call print_string
```

```
endl:
```

# Fast multiplication/division by powers of 2



K. N. Toosi  
University of Technology

`SHL eax, 3`      `eax *= 23`

`SHR eax, 10`      `eax /= 210`



# What about signed numbers? (SHL)



K. N. Toosi  
University of Technology

## SHL

- **Positive**
  - 00111101
  - 01111010
- **Negative**
  - 11111111            (-1)

# What about signed numbers? (SHL)



K. N. Toosi  
University of Technology

## SHL

- **Positive**
  - 00111101
  - 01111010
- **Negative**
  - 11111111           (-1)
  - 11111110           (-2)

# What about signed numbers? (SHL)



K. N. Toosi  
University of Technology

## SHL

- **Positive**
  - 00111101
  - 01111010
- **Negative**
  - 11111111            (-1)
  - 11111110            (-2)
  - As long as it remains negative

# What about signed numbers? (SHL)



K. N. Toosi  
University of Technology

## SHL

- **Positive**
  - 00111101
  - 01111010
  - As long as it remains positive
- **Negative**
  - 11111111            (-1)
  - 11111110            (-2)
  - As long as it remains negative

# What about signed numbers? (SHR)



K. N. Toosi  
University of Technology

## SHR

- Positive
  - 00111101
  - 00011110

# What about signed numbers? (SHR)



K. N. Toosi  
University of Technology

## SHR

- **Positive**
  - 00111101
  - 00011110
- **Negative**
  - 11111100           (-4)
  - 01111110

# What about signed numbers? (SHR)



K. N. Toosi  
University of Technology

## SHR

- **Positive**
  - 00111101
  - 00011110
- **Negative**
  - 11111100           (-4)
  - 01111110           (126)

# What about signed numbers? (SHR)



K. N. Toosi  
University of Technology

## SHR

- **Positive**
  - 00111101
  - 00011110
- **Negative**
  - 11111100 (-4)
  - 01111110 (126)
  - if filled with 1's from left
    - 11111110 (-2)



# What about signed numbers? (SHR)



K. N. Toosi  
University of Technology

## SHR

- Positive
  - 00111101
  - 00011110
- Negative
  - 11111100           (-4)
  - 01111110           (126)
  - if filled with 1's from left
    - 11111110       (-2)
- fill with signed bit from left!

# Arithmetic Shift (signed shift)



K. N. Toosi  
University of Technology

**SAL** (Shift Arithmetic Left)

**SAR** (Shift Arithmetic Right)



# Arithmetic Shift (signed shift)

**SAL:** an alias for SHL

**SAR:** fills with sign bit from left (copies sign bit)

original number	mov al, 11011001b	1	1	0	1	1	0	0	1	
shift 1 bit left	SAL al, 1	1	0	1	1	0	0	1	0	al <<= 1

original number	mov al, 11011001b	1	1	0	1	1	0	0	1	
shift 1 bit right	SAR al, 1	1	1	1	0	1	1	0	0	al >>= 1



# Arithmetic Shift (signed shift)

**SAL:** an alias for SHL

**SAR:** fills with sign bit from left (copies sign bit)

```
mov eax, 10  
sar eax, 1
```

```
mov eax, 10  
sar eax, 2
```

```
sar eax, 20
```

```
mov eax, -1  
sar eax, 1
```

# Example: Shift arithmetic right



```
segment .data simple_sar.asm
msg: db "shift must be <= 32", 10, 0

segment .text
:
call read_int
mov ebx, eax

call read_int
cmp eax, 32
ja err_lbl
```

```
simple_sar.asm (cont.)

mov cl, al

sar ebx, cl
mov eax, ebx

call print_int
call print_nl

jmp endl

err_lbl:
mov eax, msg
call print_string

endl:
```

# Practice:



K. N. Toosi  
University of Technology

```
call read_int

mov ebx, 0
mov ecx, 32
startloop:
shl eax, 1
jnc l1
inc ebx
l1:
loop startloop

mov eax, ebx
call print_int
call print_nl
```

# Practice: counting 1 bits



```
call read_int count_bits1.asm
```

```
mov ebx, 0
```

```
mov ecx, 32
```

```
startloop:
```

```
shl eax, 1
```

```
jnc l1
```

```
inc ebx
```

```
l1:
```

```
loop startloop
```

```
mov eax, ebx
```

```
call print_int
```

```
call print_nl
```

```
call read_int count_bits2.asm
```

```
mov ebx, 0
```

```
mov ecx, 32
```

```
startloop:
```

```
rol eax, 1
```

```
jnc l1
```

```
inc ebx
```

```
l1:
```

```
loop startloop
```

```
mov eax, ebx
```

```
call print_int
```

```
call print_nl
```

# Practice: counting 1 bits



```
call read_int count_bits1.asm
```

```
mov ebx, 0
```

```
mov ecx, 32
```

```
startloop:
```

```
shl eax, 1
```

```
jnc l1
```

```
inc ebx
```

```
l1:
```

```
loop startloop
```

```
mov eax, ebx
```

```
call print_int
```

```
call print_nl
```

```
call read_int count_bits2.asm
```

```
mov ebx, 0
```

```
mov ecx, 32
```

```
startloop:
```

```
rol eax, 1
```

```
jnc l1
```

```
inc ebx
```

```
l1:
```

```
loop startloop
```

```
mov eax, ebx
```

```
call print_int
```

```
call print_nl
```



# Practice: counting 1 bits



K. N. Toosi  
University of Technology

call read\_int count\_bits1.asm

```
mov ebx, 0
mov ecx, 32
```

startloop:

```
shl eax, 1
jnc l1
inc ebx
```

l1:

```
loop startloop
```

```
mov eax, ebx
call print_int
call print_nl
```

call read\_int count\_bits2.asm

```
mov ebx, 0
mov ecx, 32
```

startloop:

```
rol eax, 1
jnc l1
inc ebx
```

l1:

```
loop startloop
```

```
mov eax, ebx
call print_int
call print_nl
```

call read\_int count\_bits3.asm

```
mov ebx, 0
mov ecx, 32
```

startloop:

```
rol eax, 1
adc ebx, 0
loop startloop
```

```
mov eax, ebx
call print_int
call print_nl
```

# Rotate instructions

- ROL `reg/mem, immed8/CL`
- ROR `reg/mem, immed8/CL`
- RCL `reg/mem, immed8/CL`
- RCR `reg/mem, immed8/CL`



# Bitwise operations

- `AND src, dst`
- `OR src, dst`
- `XOR src, dst`
- `NOT dst`



# Practice:

- `OR eax, 100b`
- `AND bx, 0FFDFh`
- `XOR cx, 100b`
- `OR ax, 0F0h`
- `XOR bx, 0FFFFh`



# Practice:



K. N. Toosi  
University of Technology

test\_and.asm

```
call read_int
```

```
and eax, 01111b
```

```
call print_int
```

```
call print_nl
```

# Practice: change the n-th bit



K. N. Toosi  
University of Technology

- read  $a$ ,  $n$  from input
- change the  $n$ -th bit of  $a$ 
  - **set** (turn on, set to 1)
  - **unset** (turn off, clear, set to 0)
  - **flip** (switch, complement, invert)

# Practice: change the n-th bit



- read a, n from input
- change the n-th bit of a
  - **set** (turn on, set to 1)
  - **unset** (turn off, clear, set to 0)
  - **flip** (switch, complement, invert)

setbit0.asm

```
call read_int  
mov ebx, eax
```

```
call read_int
```

```
; write code here
```

# Practice: set the n-th bit



- read a, n from input
- change the n-th bit of a
  - **set** (turn on, set to 1)

setbit.asm

```
call read_int  
mov ebx, eax
```

```
call read_int  
mov cl, al
```

```
mov eax, 1  
shl eax, cl
```

```
or ebx, eax
```



# Practice: unset the n-th bit



- read a, n from input
- change the n-th bit of a
  - **unset** (turn off, clear, set to 0)

setbit.asm

```
call read_int  
mov ebx, eax
```

```
call read_int  
mov cl, al
```

```
mov eax, 1  
shl eax, cl  
not eax
```

```
and ebx, eax
```

# Practice: flip the n-th bit

- read a, n from input
- change the n-th bit of a
  - **flip** (switch, complement, invert)



setbit.asm

```
call read_int  
mov ebx, eax
```

```
call read_int  
mov cl, al
```

```
mov eax, 1  
shl eax, cl  
xor ebx, eax
```

# Check the n-th bit



```
segment .data                                     checkbit1.asm
msg1:  db "bit 7 is on", 10, 0
msg2:  db "bit 7 is off", 10, 0

segment .text
:
  call read_int
  and ax, 10000000b
  jnz onlbl

  mov eax, msg2
  jmp endl

onlbl:
  mov eax, msg1
endl:
  call print_string
```

# Check the n-th bit



```
segment .data                                     checkbit1.asm
msg1:  db "bit 7 is on", 10, 0
msg2:  db "bit 7 is off", 10, 0

segment .text
:
  call read_int
  and ax, 10000000b
  jnz onlbl

  mov eax, msg2
  jmp endl

onlbl:
  mov eax, msg1
endl:
  call print_string
```

but AX gets changed here!



# Check the n-th bit



```
segment .data
```

```
checkbit1.asm
```

```
msg1: db "bit 7 is on", 10, 0
```

```
msg2: db "bit 7 is off", 10, 0
```

```
segment .text
```

```
:
```

```
call read_int
```

```
and ax, 10000000b
```

```
jnz onlbl
```

```
mov eax, msg2
```

```
jmp endl
```

```
onlbl:
```

```
mov eax, msg1
```

```
endl:
```

```
call print_string
```

```
segment .data
```

```
checkbit2.asm
```

```
msg1: db "bit 7 is on", 10, 0
```

```
msg2: db "bit 7 is off", 10, 0
```

```
segment .text
```

```
:
```

```
call read_int
```

```
test ax, 10000000b
```

```
jnz onlbl
```

```
mov eax, msg2
```

```
jmp endl
```

```
onlbl:
```

```
mov eax, msg1
```

```
endl:
```

```
call print_string
```

# Practice:

- `XOR eax, eax`



**K. N. Toosi**  
University of Technology

# Practice:



K. N. Toosi  
University of Technology

- `XOR eax, eax`
  
- `XOR eax, ebx`
- `XOR ebx, eax`
- `XOR eax, ebx`

# Practice:



K. N. Toosi  
University of Technology

- `XOR eax, eax`
  
- `XOR eax, ebx`
- `XOR ebx, eax`
- `XOR eax, ebx`
- `≡ XCHG eax, ebx`



# Parity FLAG



K. N. Toosi  
University of Technology

After a math/bit operation

- PF = 0 odd number of set (=1) bits in first byte (LSB)
- PF = 1 even number of set (=1) bits in first byte (LSB)

# Appendix A of the book



K. N. Toosi  
University of Technology

- instructions
- which flags affected

(-> Carter, Paul A. *PC Assembly Language*. Lulu. com, 2007.)