Introduction to 8086 Assembly

Lecture 8

Bit Operations

Clock cycle & instruction timing

K. N. Toosi
University of Technology

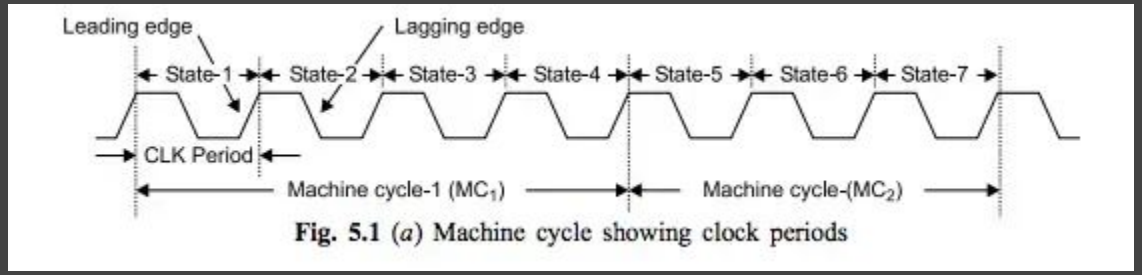


Fig. 5.1 (*a*) Machine cycle showing clock periods

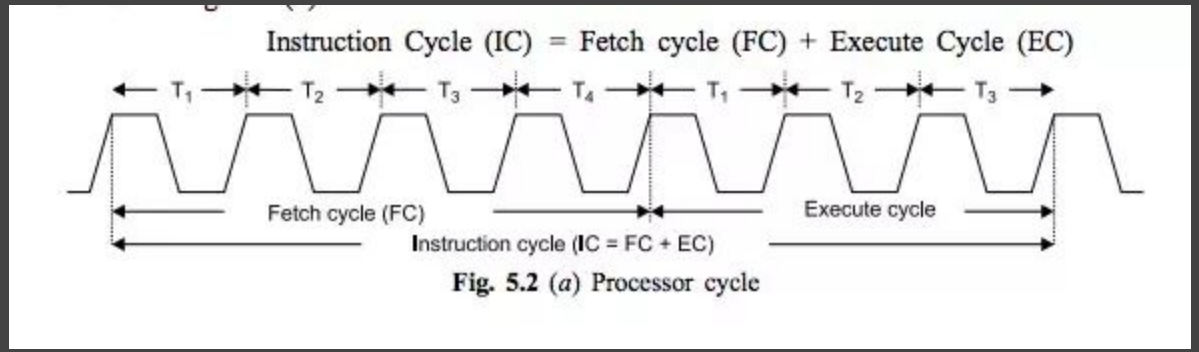
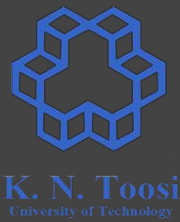


Fig. 5.2 (*a*) Processor cycle

http://www.newagepublishers.com/samplechapter/000495.pdf

Clock cycle & instruction timing



See

- 8086 Instruction Timing (http://www.oocities.org/mc_introtocomputers/)
  - http://www.oocities.org/mc_introtocomputers/Instruction_Timing.PDF
- Agner Fog's Software optimization resources - Instruction tables
  - https://www.agner.org/optimize/instruction_tables.pdf
- Instruction latencies and throughput for AMD and Intel x86 processors (Torbjorn Granlund)
  - https://gmplib.org/~tege/x86-timing.pdf
- Intel® 64 and IA-32 Architectures Optimization Reference Manual
  - https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.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

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, 3	1	1	0	0	1	0	0	0	al <<= 3

original number	mov al, 11011001b	1	1	0	1	1	0	0	1	
shift 1 bit right	SHR al, 3	0	0	0	1	1	0	1	1	al >>= 3

Example: Shift left

simple_shl.asm

```
segment .data
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 left

K. N. Toosi
University of Technology

simple_shl.asm

```
segment .data
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
```

```
CS@kntu:lecture8$ ./run.sh simple_shl
12
1
24
CS@kntu:lecture8$ ./run.sh simple_shl
7
3
56
```

Example: Shift right

simple_shr.asm

```
segment .data
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:
```

Example: Shift right

K. N. Toosi
University of Technology

simple_shr.asm

```
segment .data
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
```

```
CS@kntu:lecture8$ ./run.sh simple_shr
32
1
16
CS@kntu:lecture8$ ./run.sh simple_shr
32
3
4
CS@kntu:lecture8$ ./run.sh simple_shr
31
3
3
CS@kntu:lecture8$ ./run.sh simple_shr
37
3
4
```

Fast multiplication/division by powers of 2


```
SHL eax, 3          eax *= 2^3

SHR eax, 10         eax /= 2^10
```

What about signed numbers? (SHL)

SHL

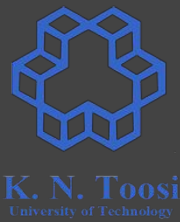
- Positive
  - 00111101
  - 01111010
  - As long as it remains positive
- Negative
  - 11111111 (-1)

What about signed numbers? (SHL)

SHL

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

What about signed numbers? (SHL)



SHL

- Positive
  - 00111101
  - 01111010
  - As long as it remains positive
- Negative
  - 11111111 (-1)
  - 11111110 (-2)
  - 11111000 (-8)

What about signed numbers? (SHL)

SHL

- Positive
  - 00111101
  - 01111010
  - As long as it remains positive
- Negative
  - 11111111 (-1)
  - 11111110 (-2)
  - 11111000 (-8)
  - As long as it remains negative

What about signed numbers? (SHR)

SHR

- Positive
  - 00111101
  - 00011110

What about signed numbers? (SHR)

SHR

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

What about signed numbers? (SHR)

SHR

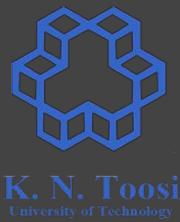
- Positive
  - 00111101
  - 00011110
- Negative
  - 11111100 (-4)
  - 01111110 (126)

What about signed numbers? (SHR)

SHR

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

What about signed numbers? (SHR)



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)

SAL (Shift Arithmetic Left)

SAR (Shift Arithmetic Right)

Arithmetic Shift (signed shift)

SAL: an alias for SHL

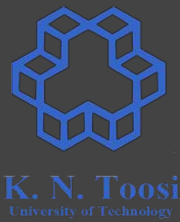
SAR: fills with sign bit from left (copies sign bit)

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

original number	mov al, 01011001b	0	1	0	1	1	0	0	1	
shift 1 bit right	SAR al, 1	0	0	1	0	1	1	0	0	al >>= 1

Arithmetic Shift (signed shift)

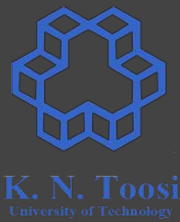


SAL: an alias for SHL

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

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

Arithmetic Shift (signed shift)

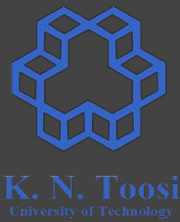


SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111111110110 = -10
sar eax, 1
sar eax, 1
sar eax, 1
sar eax, 1
sar eax, 1
```

Arithmetic Shift (signed shift)

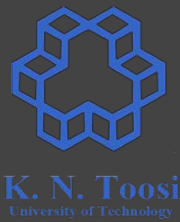


SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111110110 = -10
sar eax, 1       EAX=11111111111111111111111111011 = -5
sar eax, 1
sar eax, 1
sar eax, 1
sar eax, 1
```

Arithmetic Shift (signed shift)

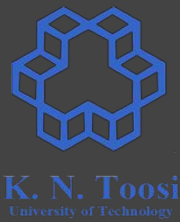


SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111110110 = -10
sar eax, 1       EAX=11111111111111111111111111011 = -5
sar eax, 1       EAX=11111111111111111111111111101 = -3
sar eax, 1
sar eax, 1
sar eax, 1
```

Arithmetic Shift (signed shift)

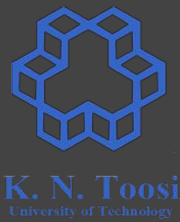


SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111110110 = -10
sar eax, 1       EAX=11111111111111111111111111011 = -5
sar eax, 1       EAX=11111111111111111111111111101 = -3
sar eax, 1       EAX=11111111111111111111111111110 = -2
sar eax, 1
sar eax, 1
```

Arithmetic Shift (signed shift)



SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111110110 = -10
sar eax, 1       EAX=11111111111111111111111111011 = -5
sar eax, 1       EAX=11111111111111111111111111101 = -3
sar eax, 1       EAX=11111111111111111111111111110 = -2
sar eax, 1       EAX=11111111111111111111111111111 = -1
sar eax, 1
```

Arithmetic Shift (signed shift)

SAL: an alias for SHL

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

```
mov eax, -10     EAX=11111111111111111111111110110 = -10
sar eax, 1       EAX=11111111111111111111111111011 = -5
sar eax, 1       EAX=11111111111111111111111111101 = -3
sar eax, 1       EAX=11111111111111111111111111110 = -2
sar eax, 1       EAX=11111111111111111111111111111 = -1
sar eax, 1       EAX=11111111111111111111111111111 = -1
```

Example: Shift arithmetic right

simple_sar.asm

```
segment .data
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:


```
        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

count_bits1.asm

```
    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
```

count_bits2.asm

```
    call read_int

    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

count_bits1.asm

```
    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
```

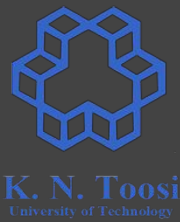
count_bits2.asm

```
    call read_int

    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



count_bits1.asm

```
    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
```

count_bits2.asm

```
    call read_int
    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
```

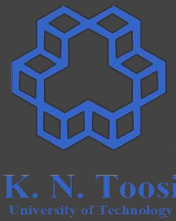
count_bits3.asm

```
    call read_int
    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

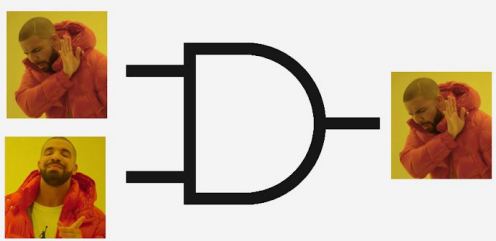
- `AND  dst, src`
- `OR  dst, src`
- `XOR  dst, src`
- `NOT  dst`

Bitwise operations

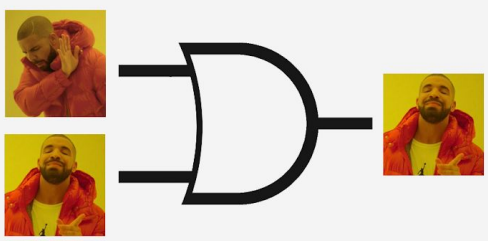
K. N. Toosi
University of Technology

Drake's Logic Gates

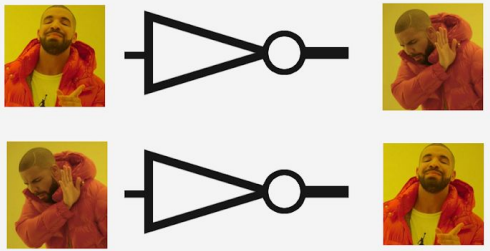
'And' gate
&

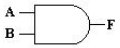


'Or' Gate
|



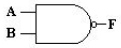
'Not' Operator
~





AND

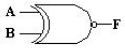
OR



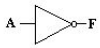
NAND

NOR

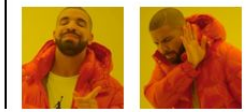
XOR

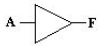


XNOR



NOT





BUFFER

Practice:

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

Practice:

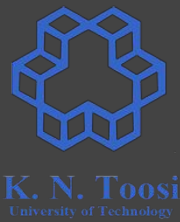
test_and.asm

```
call read_int

and eax, 01111b

call print_int
call print_nl
```

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)

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

checkbit1.asm

```
segment .data
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

checkbit1.asm

```
segment .data
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

checkbit1.asm

```
segment .data
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
```

checkbit2.asm

```
segment .data
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:

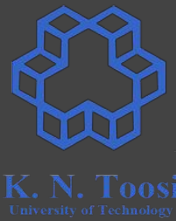
- `XOR  eax, eax`

Practice:

- `XOR  eax, eax`

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

Practice:



- `XOR  eax, eax`

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

Parity FLAG

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

- instructions
- which flags affected

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