Introduction to 8086 Assembly

Lecture 6

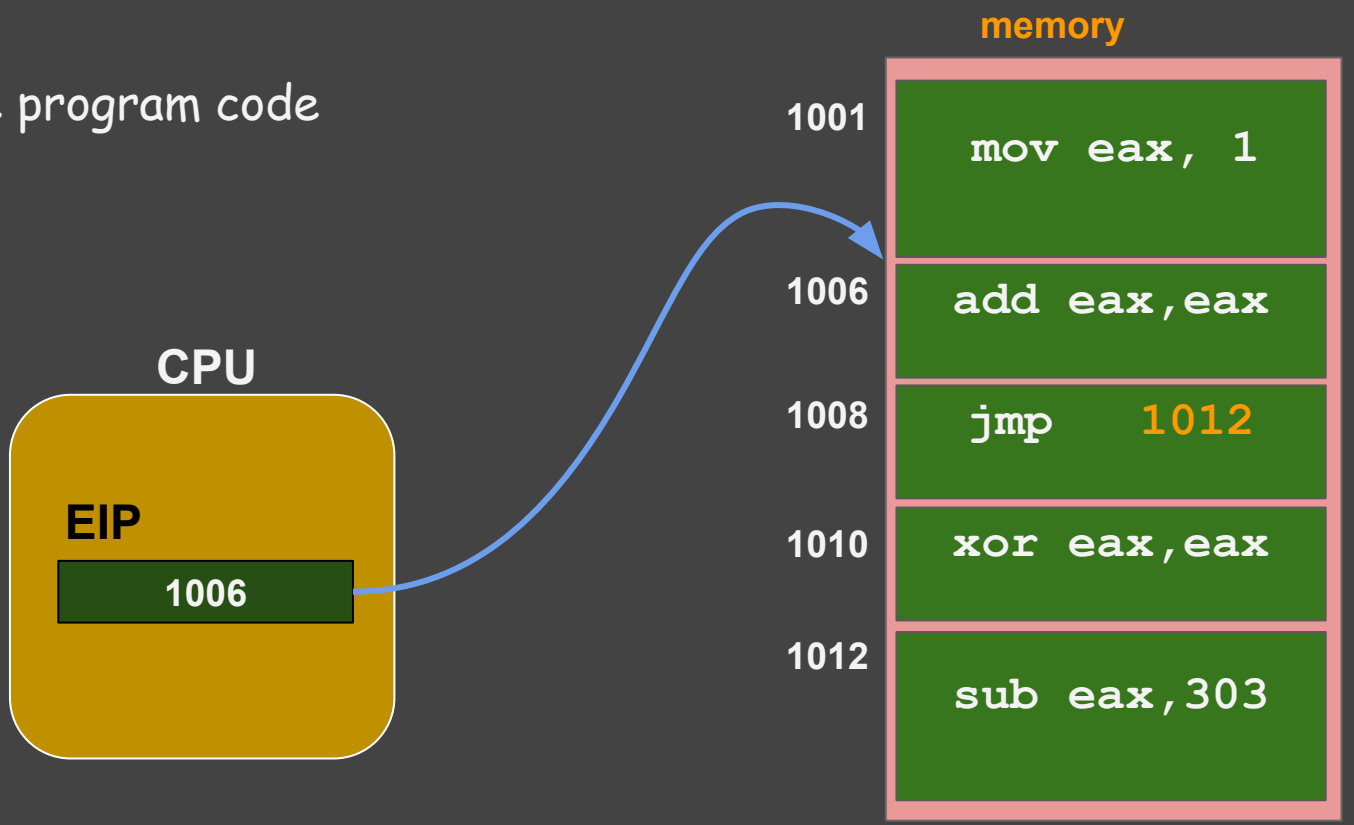
Working with memory

Why use memory?

-

Why use memory?

- Store program code
-



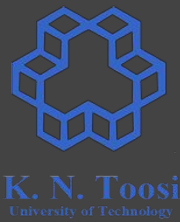
Why use memory?

- Store program code
- Registers are limited in number and size
- Program data
-

Why use memory?

- Store program code
- Registers are limited in number and size
- Program data
  - Numbers, pointers, arrays, structures, data structures,
  - Text
  - Photos
  - Audio
  - Video

Why use memory?



- Store program code
- Registers are limited in number and size
- Program data
  - Numbers, pointers, arrays, structures, data structures,
  - Text
  - Photos
  - Audio
  - Video
- Memory-mapped IO

The data segment (section)


```
segment .data

    dd 1234
    dw 13
    db -123
```

How to access data?


```
segment .data

    dd 1234
    dw 13
    db -123
```

But how to access data?

How to access data? Labels!


```
segment .data

l1:  dd 1234
     dw 13
     db -123
```

How to access data?

memory1.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl
```

How to access data?

memory1.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl
```

```
CS@kntu:lecture6$ nasm -f elf -d ELF_TYPE asm_io.asm
CS@kntu:lecture6$ gcc -m32 -c driver.c
CS@kntu:lecture6$ nasm -f elf memory1.asm
CS@kntu:lecture6$ gcc -m32 -o memory1 driver.c memory1.o asm_io.o
CS@kntu:lecture6$ ./memory1
1449279496
1234
```

How to access data?

memory1.asm

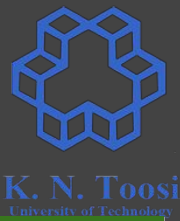
```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, l1
      call print_int
      call print_nl

      mov eax, [l1]
      call print_int
      call print_nl
```

run.sh

```
nasm -f elf -d ELF_TYPE asm_io.asm &&
gcc -m32 -c driver.c &&
nasm -f elf $1.asm &&
gcc -m32 -o $1 driver.c $1.o asm_io.o &&
./$1
```

```
CS@kntu:lecture6$ nasm -f elf -d ELF_TYPE asm_io.asm
CS@kntu:lecture6$ gcc -m32 -c driver.c
CS@kntu:lecture6$ nasm -f elf memory1.asm
CS@kntu:lecture6$ gcc -m32 -o memory1 driver.c memory1.o asm_io.o
CS@kntu:lecture6$ ./memory1
1449279496
1234
```

How to access data?



memory1.asm

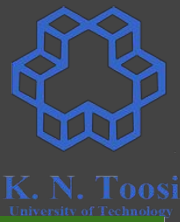
```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, l1
      call print_int
      call print_nl

      mov eax, [l1]
      call print_int
      call print_nl
```

run.sh

```
nasm -f elf -d ELF_TYPE asm_io.asm &&
gcc -m32 -c driver.c &&
nasm -f elf $1.asm &&
gcc -m32 -o $1 driver.c $1.o asm_io.o &&
./$1
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory1
134520872
1234
```

How to access data?



memory1.asm

```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, l1
      call print_int
      call print_nl

      mov eax, [l1]
      call print_int
      call print_nl
```

run.sh

```
nasm -f elf -d ELF_TYPE asm_io.asm &&
gcc -m32 -c driver.c &&
nasm -f elf $1.asm &&
gcc -m32 -o $1 driver.c $1.o asm_io.o &&
./$1
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory1
134520872
1234
```

Reading data from memory

memory1.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl
```

Data labels vs. Code labels

memory2.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, asm_main
        call print_int
        call print_nl

        mov eax, [asm_main]
        call print_int
        call print_nl
```

Data labels vs. Code labels

memory2.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, asm_main
        call print_int
        call print_nl

        mov eax, [asm_main]
        call print_int
        call print_nl
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory2
134513872
200
```

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size (not data type!)

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size (not data type!)

db	1 byte
dw	2 bytes
dd	4 bytes
dq	8 bytes
dt	10 bytes

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size

```
mov  eax, 11
call print_int
call print_nl

mov  eax, 12
call print_int
call print_nl

mov  eax, 13
call print_int
call print_nl

mov  al, [18]
call print_char
call print_nl
```

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size

```
mov  eax, 11
call print_int
call print_nl

mov  eax, 12
call print_int
call print_nl

mov  eax, 13
call print_int
call print_nl

mov  al, [18]
call print_char
call print_nl
```

```
b.nasihatkon@kntu
134520872
134520873
134520875
A
```

Definitions in the book


```
L1      db      0           ; byte labeled L1 with initial value 0
L2      dw      1000        ; word labeled L2 with initial value 1000
L3      db      110101b     ; byte initialized to binary 110101 (53 in decimal)
L4      db      12h         ; byte initialized to hex 12 (18 in decimal)
L5      db      17o         ; byte initialized to octal 17 (15 in decimal)
L6      dd      1A92h       ; double word initialized to hex 1A92
L7      resb    1           ; 1 uninitialized byte
L8      db      "A"         ; byte initialized to ASCII code for A (65)
```

Carter, *PC Assembly Language*, 2007.

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size

Putting data in memory


```
11:     db   123
12:     dw   1000
13:     db   11010b
14:     db   12o
16:     dd   1A92h
17:     dd   0x1A92
18:     db   'A'
19:     db   "B"
```

data size

Putting data in memory


```
segment .data

l1:     dd 11, 12, 13, 14, 15, 16
```

```
mov eax, [l1]
call print_int
call print_nl

mov eax, [l1+1]
call print_int
call print_nl

mov eax, [l1+2]
call print_int
call print_nl

mov eax, [l1+3]
call print_int
call print_nl


mov eax, [l1+4]
call print_int
call print_nl
```

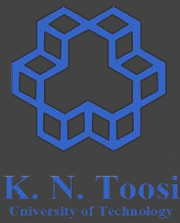
Putting data in memory

```
segment .data

l1:     dd 11, 12, 13, 14, 15, 16
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory4
11
201326592
786432
3072
12
```

```
mov eax, [l1]
call print_int
call print_nl

mov eax, [l1+1]
call print_int
call print_nl

mov eax, [l1+2]
call print_int
call print_nl

mov eax, [l1+3]
call print_int
call print_nl


mov eax, [l1+4]
call print_int
call print_nl
```

Putting data in memory




```
segment .data

l1:     dd 11, 12, 13, 14, 15, 16

l2:     dd 8, 8, 8, 8, 8, 8, 8, 8, 8

l3:     times 9   dd 8

l4:     resd  9

l5:     resw  18

l6:     resb  36
```

Argument types


```
segment .data

l1:     dd 11, 12, 13, 14, 15, 16

segment .text

mov eax, [l1]
mov [l1+4], ebx

mov eax, ebx

mov eax, l1
mov eax, 123
```

memory

register

immediate (constant)

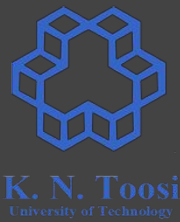
Invalid mem,mem assembly commands


```
segment .data
l1:     dd 11, 12, 13, 14
l2:     dd 100

segment .text

mov [l1], [l2]
add [l1], [l2]
sub [l1], [l2]
adc [l1], [l2]
sbb [l1], [l2]
cmp [l1], [l2]
and [l1], [l2]
or  [l1], [l2]
xor [l1], [l2]
```

Invalid mem,mem assembly commands


```
segment .data
l1:     dd 11, 12, 13, 14
l2:     dd 100

segment .text

mov [l1], [l2]
add [l1], [l2]
sub [l1], [l2]
adc [l1], [l2]
sbb [l1], [l2]
cmp [l1], [l2]
and [l1], [l2]
or  [l1], [l2]
xor [l1], [l2]
```

Operation size

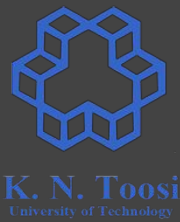



```
segment .data
l1:     dd 11, 13, 14
l2:     dd 100

segment .text

mov [l1], eax
add  eax, [l2]


sub  eax, 44
mov [l1], 44
```

Operation size


```
segment .data
l1:     dd 11, 13, 14
l2:     dd 100

segment .text

mov [l1], eax
add  eax, [l2]


sub  eax, 44
mov [l1], 44
mov dword [l1], 44
mov word [l1], 44
mov byte [l1], 44
```

Operation size




```
segment .data
l1:     dd 11, 13, 14
l2:     dd 100

segment .text

mov [l1], eax
add  eax, [l2]


sub  eax, 44
mov [l1], 44
mov dword [l1], 44
mov word [l1], 44
mov byte [l1], 44
```

What happens when we write dword?

Operation size


```
segment .data
l1:     dd 11, 13, 14
l2:     dd 100

segment .text

mov [l1], eax
add  eax, [l2]


sub  eax, 44
mov [l1], 44
mov dword [l1], 44
mov word [l1], 44
mov byte [l1], 44
```

What is the difference?


```
segment .data
l1:     dd 1324

segment .text
```

```
segment .data
l1:     resd 1

segment .text

mov dword [l1], 1324
```

Assembly command formats

- List of x86 instructions
  - http://www.felixcloutier.com/x86/
  - https://c9x.me/x86/
  - https://zsmith.co/intel.html

Storing multibyte data


```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

Storing multibyte data


```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

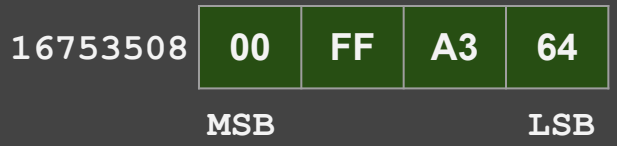
endianness1.asm

16753508	00	FF	A3	64

Storing multibyte data


```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm



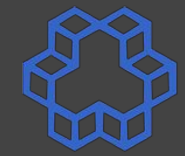
Storing multibyte data


```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

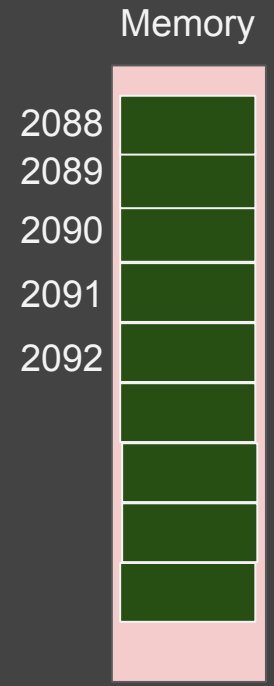
Storing multibyte data




```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

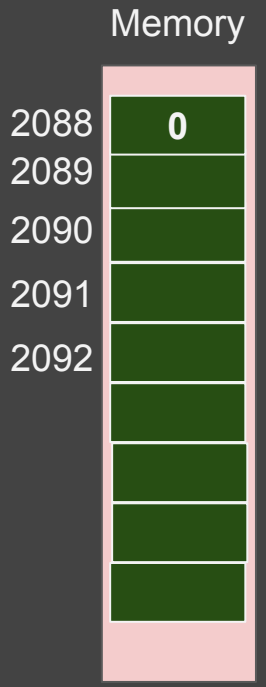


Storing multibyte data

```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

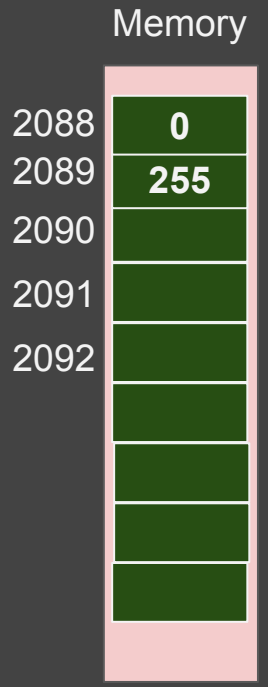


Storing multibyte data

```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

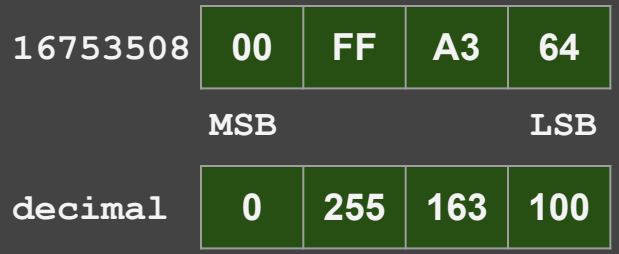
16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

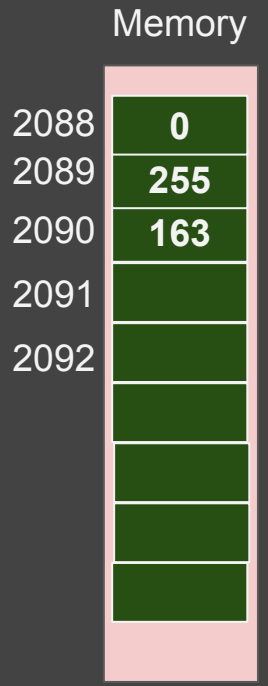


Storing multibyte data

```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm





Storing multibyte data

```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

Memory

2088	0
2089	255
2090	163
2091	100
2092	

K. N. Toosi University of Technology

Storing multibyte data

```
segment .data
a:     dd    16753508

segment .text
          :
          :

        mov eax, 0

        mov  al, [a]
        call print_int
        call print_nl
```

endianness1.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100

Memory

2088	0
2089	255
2090	163
2091	100
2092	

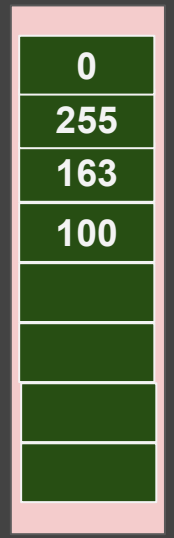
```
b.nasihatkon@kntu:lecture6$ ./run.sh endianness1
100
```

Endianness

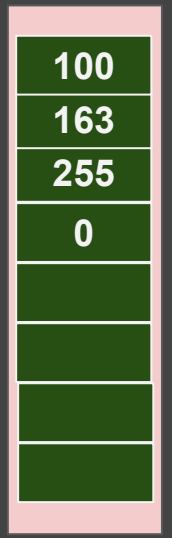
```
segment .data
a:    dd   16753508
segment .text
      :
  mov eax, 0

  mov  al, [a]
  call print_int
  call print_nl

  mov  al, [a+1]
  call print_int
  call print_nl

  mov  al, [a+2]
  call print_int
  call print_nl

  mov  al, [a+3]
  call print_int
  call print_nl
```

endianness2.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100



Big Endian



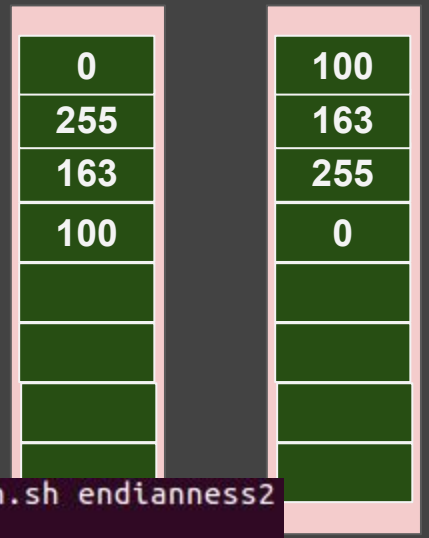
Little Endian (e.g. x86)

Endianness

```
segment .data
a:    dd   16753508
segment .text
      :
  mov eax, 0

  mov  al, [a]
  call print_int
  call print_nl

  mov  al, [a+1]
  call print_int
  call print_nl

  mov  al, [a+2]
  call print_int
  call print_nl

  mov  al, [a+3]
  call print_int
  call print_nl
```

endianness2.asm

16753508	00	FF	A3	64
	MSB			LSB
decimal	0	255	163	100



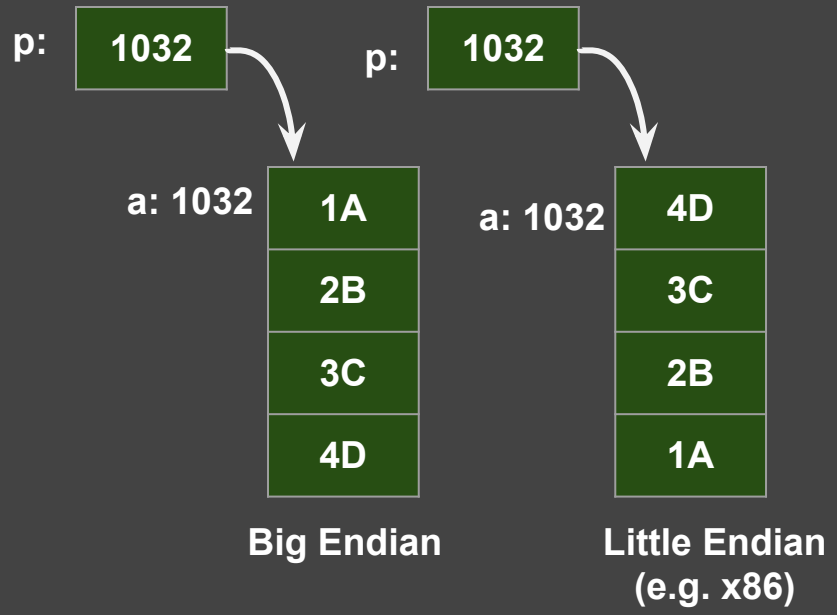

```
b.nasihatkon@kntu:lecture6$ ./run.sh endianness2
100
163
255
0
```

Little Endian (e.g. x86)

Checking endianness in C

```c
unsigned int a = 0x1A2B3C4D;
printf("%X\n", a);

unsigned char *p = (unsigned char *)(&a);
printf("%X\n", *p);
printf("%X\n", *(p+1));
printf("%X\n", *(p+2));
printf("%X\n", *(p+3));
```

test_endianness.c

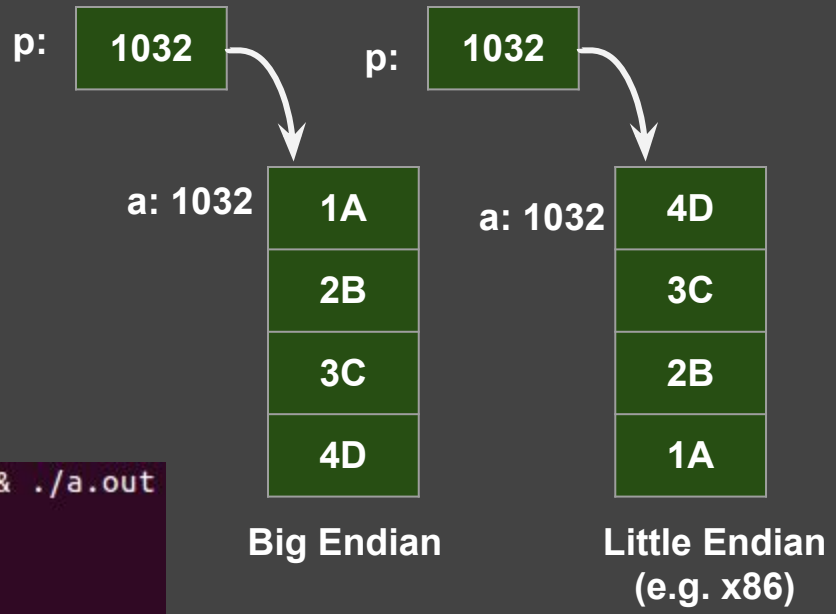


Checking endianness in C

K. N. Toosi
University of Technology

```c
unsigned int a = 0x1A2B3C4D;
printf("%X\n", a);

unsigned char *p = (unsigned char *)(&a);
printf("%X\n", *p);
printf("%X\n", *(p+1));
printf("%X\n", *(p+2));
printf("%X\n", *(p+3));
```

```
b.nasihatkon@kntu:lecture6$ gcc test_endianness.c && ./a.out
1A2B3C4D
4D
3C
2B
1A
```



Change endianness

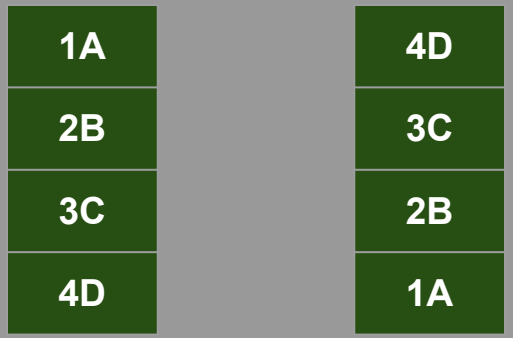
AX:

Change endianness


```
AX:

xchg  ah, al ; 16 bit
```

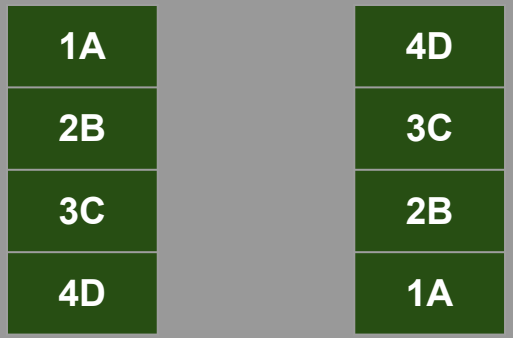
Change endianness


```
AX:

xchg  ah, al ; 16 bit

EAX:
```



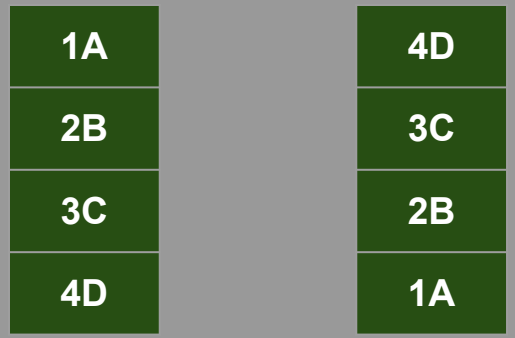
Change endianness


```
AX:

xchg  ah, al ; 16 bit

EAX:

bswap eax    ; 32 bit
```



Change endianness


```
AX:

xchg  ah, al ; 16 bit

EAX:

bswap eax    ; 32 bit

bswap rax    ; 64 bit (x64 only)
```



Remember: Data labels vs. Code labels

memory2.asm

```
segment .data
l1:     dd 1234

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        mov eax, asm_main
        call print_int
        call print_nl

        mov eax, [asm_main]
        call print_int
        call print_nl
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory2
134513872
200
```

Remember: Data labels vs. Code labels

memory2.asm

```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, asm_main
      call print_int
      call print_nl

      mov eax, [asm_main]
      call print_int
      call print_nl
```

```
CS@kntu:lecture6$ objdump -d memory2

memory2:     file format elf32-i386


Disassembly of section .init:

08048310 <_init>:
 8048310:       53                      push   %ebx
 8048311:       83 ec 08                sub    $0x8,%esp
 8048314:       e8 c7 00 00 00          call   80483e0 <__x8
```

Remember: Data labels vs. Code labels

memory2.asm

```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, asm_main
      call print_int
      call print_nl

      mov eax, [asm_main]
      call print_int
      call print_nl
```

```
CS@kntu:lecture6$ objdump -d memory2

memory2:     file format elf32-i386


Disassembly of section .init:

08048310 <_init>:
 8048310:       53                      push   %ebx
 8048311:       83 ec 08                sub    $0x8,%esp
 8048314:       e8 c7 00 00 00          call   80483e0 <  x8
```

```
080484d0 <asm_main>:
 80484d0:       c8 00 00 00             enter  $0x0,$0x0
 80484d4:       60                      pusha
 80484d5:       b8 d0 84 04 08          mov    $0x80484d0,%eax
 80484da:       e8 3e 00 00 00          call   804851d <print_int>
 80484df:       e8 a4 00 00 00          call   8048588 <print_nl>
 80484e4:       a1 d0 84 04 08          mov    0x80484d0,%eax
 80484e9:       e8 2f 00 00 00          call   804851d <print_int>
 80484ee:       e8 95 00 00 00          call   8048588 <print_nl>
 80484f3:       61                      popa
 80484f4:       c9                      leave
 80484f5:       c3                      ret
```

Remember: Data labels vs. Code labels

memory2.asm

```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, asm_main
      call print_int
      call print_nl

      mov eax, [asm_main]
      call print_int
      call print_nl
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory2
134513872
200
```

```
080484d0 <asm_main>:
 80484d0:       c8 00 00 00             enter  $0x0,$0x0
 80484d4:       60                      pusha
 80484d5:       b8 d0 84 04 08          mov    $0x80484d0,%
 80484da:       e8 3e 00 00 00          call   804851d <pri
 80484df:       e8 a4 00 00 00          call   8048588 <pri
 80484e4:       a1 d0 84 04 08          mov    0x80484d0,%e
 80484e9:       e8 2f 00 00 00          call   804851d <pri
 80484ee:       e8 95 00 00 00          call   8048588 <pri
 80484f3:       61                      popa
 80484f4:       c9                      leave
 80484f5:       c3                      ret
```

Remember: Data labels vs. Code labels

memory2.asm

```
segment .data
l1:    dd 1234

segment .text
      global  asm_main
asm_main:
      enter   0,0
      pusha

      mov eax, asm_main
      call print_int
      call print_nl

      mov eax, [asm_main]
      call print_int
      call print_nl
```

```
b.nasihatkon@kntu:lecture6$ ./run.sh memory2
134513872
200
```

```
080484d0 <asm_main>:
 80484d0:       c8 00 00 00             enter  $0x0,$0x0
 80484d4:       60                      pusha
 80484d5:       b8 d0 84 04 08          mov    $0x80484d0,%
 80484da:       e8 3e 00 00 00          call   804851d <pri
 80484df:       e8 a4 00 00 00          call   8048588 <pri
 80484e4:       a1 d0 84 04 08          mov    0x80484d0,%e
 80484e9:       e8 2f 00 00 00          call   804851d <pri
 80484ee:       e8 95 00 00 00          call   8048588 <pri
 80484f3:       61                      popa
 80484f4:       c9                      leave
 80484f5:       c3                      ret
```