

BYTE AND ENDIANNESS



0:05 / 8:53



BINARY

0

1

Byte = 8 bits, ISO/IEC 2382-1:1993

Bit Position

7

6

5

4

3

2

1

0

1	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---

1	0	1	0
---	---	---	---

$$1 * 2^3 + 1 * 2^1 = 10$$

a

1	1	0	0
---	---	---	---

$$1 * 2^3 + 1 * 2^2 = 12$$

c

0xac

```
char a = 'a';
```

ASCII: American Standard Code for Information Interchange

'a': 0x61

0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

0x61


```
char a[2] = "ab";
```

`'a':0x61` `'b':0x62`

0x61

0x62

Byte 1

Byte 2

Low memory address

High memory address

相对big-endian

LITTLE ENDIAN: *LEAST* SIGNIFICANT BYTE → *LOW* ADDRESS

```
int i = 1100;
```

int: 4 bytes,

$i = 0x44c = 0x\ 00\ 00\ 04\ 4c$



Byte 1

Byte 2

Byte 3

Byte 4

Low memory address

High memory address

LITTLE ENDIAN: *LEAST* SIGNIFICANT BYTE -> *LOW* ADDRESS

```
int i = 1100;
```

int: 4 bytes,

$i = 0x44c = 0x\ 00\ 00\ 04\ 4c$

4c

Byte 1

04

Byte 2

00

Byte 3

00

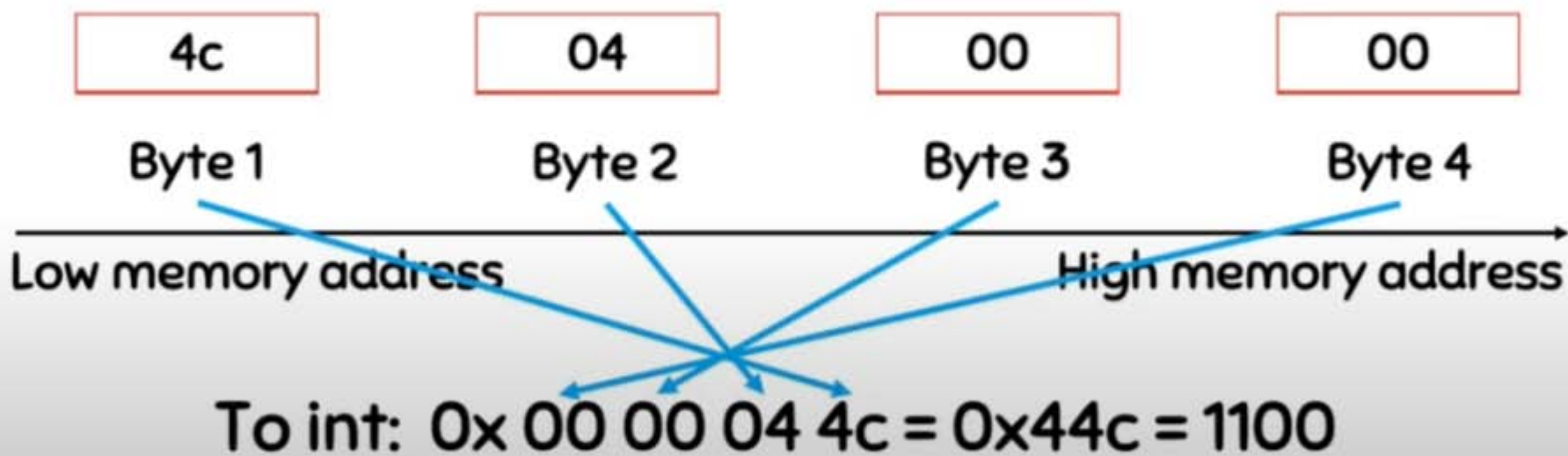
Byte 4

Low memory address

High memory address

LITTLE ENDIAN: *LEAST* SIGNIFICANT BYTE -> *LOW* ADDRESS

```
char s[4] = "\x4c\x04\x00\x00";
```



LITTLE ENDIAN: *LEAST* SIGNIFICANT BYTE -> *LOW* ADDRESS

```
struct.pack("<i", 1100)
```


LITTLE ENDIAN: *LEAST* SIGNIFICANT BYTE -> *LOW* ADDRESS

```
struct.unpack("<i", "\x4c\x04\x00\x00")[0]
```

X86-64 ASSEMBLY 101

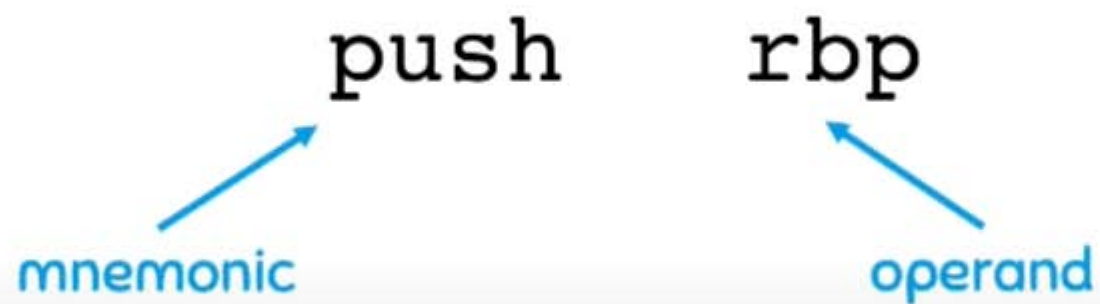


0:11 / 15:38



push rbp

mnemonic operand



Push the value of **register** rbp to the **stack**

REGISTER

Register is a location that a CPU is able to visit quickly.

“CPU-defined variable”



2:13 / 15:38



64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

STACK: WHERE THE **PUSH** GOES

- × First in, last out
- × Push to the **low** address
- × `push rbp`
- × push the value of `rbp` to the stack

XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYY

mov rbp, rsp

mnemonic operands

SYNTAX: INTEL VS AT&T

Intel syntax

mov

rbp, rsp



AT&T syntax

movq

%rsp, %rbp



OPERANDS

mov rax, 0
 register immediate

mov rdx, qword ptr [rcx]
 absolute

mov esi, dword ptr [rbp+4*rax-48]
 scaled indexed

MORE INSTRUCTIONS

```
cmp    rax, rbx  
jge    xxx
```

Intel syntax

if $rax \geq rbx$ then jump xxx

MORE INSTRUCTIONS

```
cmpl    %rbx, %rax  
jge     xxx
```

AT&T syntax

if rax >= rbx then jump xxx

PREFIX -- EXAMPLE

```
mov ecx, eax  
and ecx, 3  
rep movs byte ptr es:[edi], byte ptr[esi]
```

repeat until ecx is equal to 0

ASSEMBLY LEARNING TIPS

- × Similar to learning a foreign language
- × You don't have to learn all the instructions (vocabulary) --- use instruction reference as your "dictionary"
- × Read more (dis)assembly

USEFUL RESOURCES

x86-64 instruction reference: <https://www.felixcloutier.com/x86/>

x86-64 cheat sheet:

https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf