Низькорівневе програмування та програмування мікроконтролерів

З використанням мови програмування **Assembler.**
Регістри. Операції з регістрами. Операції з пам’яттю

класрум **im3ozqq4**

Регістри

Not modified for 8-bit operands Not modified for 16-bit operands Zero-extended for 32-bit operands			Low 8-bit	16-bit	32-bit	64-bit
		AH†	AL	AX	EAX	RAX
		BH†	BL	BX	EBX	RBX
		CH†	CL	CX	ECX	RCX
		DH†	DL	DX	EDX	RDX
			SIL‡	SI	ESI	RSI
			DIL‡	DI	EDI	RDI
			BPL‡	BP	EBP	RBP
			SPL‡	SP	ESP	RSP

Регістр **IP**

Регістр **IP (Instruction Pointer)**
використовується для зберігання адреси
наступної інструкції, яку має виконати процесор,
забезпечуючи послідовність виконання програмного коду.

Програміст не може змінити цей регістр.
Він використовується процесором для власних потреб

Регістр **IP**

```
004000b0  ba14000000      mov edx, 0x14
004000b5  48 8d351510…    lea rsi, [rip+0x4010d1]
004000bc  bf01000000      mov edi, 1
004000c1  b801000000      mov eax, 1
004000c6  0f 05           syscall
004000c8  31ff            xor edi, edi
004000ca  b83c000000      mov eax, 0x3c
004000cf  0f 05           syscall
```

```
rax : 0x00
rbx : 0x00
rcx : 0x00
rdx : 0x14
rsp : 0x4fffffffed0
rbp : 0x00
rsi : 0x00
rdi : 0x00
rip : 0x4000b5
```

```
004000b0  ba14000000      mov edx, 0x14
004000b5  48 8d351510…    lea rsi, [rip+0x4010d1]
004000bc  bf01000000      mov edi, 1
004000c1  b801000000      mov eax, 1
004000c6  0f 05           syscall
004000c8  31ff            xor edi, edi
004000ca  b83c000000      mov eax, 0x3c
004000cf  0f 05           syscall
```

```
rax : 0x00
rbx : 0x00
rcx : 0x00
rdx : 0x14
rsp : 0x4fffffffed0
rbp : 0x00
rsi : 0x4010d1
rdi : 0x01
rip : 0x4000c1
```

Операції з пам’яттю. **MOV**

Загальний синтаксис:

MOV dst, src

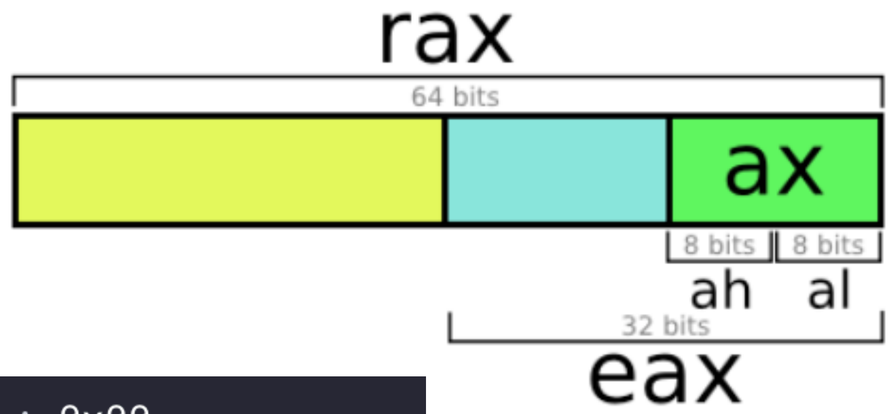
Після виконання цієї операції
Вміст **dst** дорівнює вмісту **src**

Тобто семантично можна представити переміщення даних

dst ← src

Операції з пам’яттю. **MOV**

Запис значення в регістр

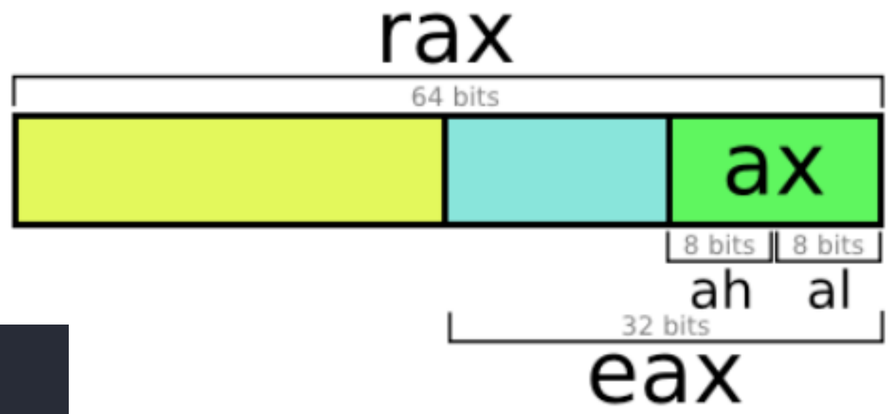



```
00401000  _start:
00401000  b012          mov al, 0x12
00401002  66 b83412     mov ax, 0x1234
00401006  b878563412    mov eax, 0x12345678
0040100b  48 b8efcd…    movabs rax, 0x1234567890abcdef
00401015  b000          mov al, 0
00401017  48 c7c701…    mov rdi, 1
0040101e  48 8d3425…    lea rsi
```

```
rax : 0x00
rbx : 0x00
rcx : 0x00
rdx : 0x4fffffffff7
rsp : 0x4fffffffed0
rbp : 0x00
rsi : 0x00
rdi : 0x00
```

Операції з пам’яттю. **MOV**

Запис значення в регістр




```
_start:
b012          mov al, 0x12
```

```
rax : 0x00
rbx : 0x00
```

```
_start:
b012          mov al, 0x12
66 b83412     mov ax, 0x1234
```

```
rax : 0x12
rbx : 0x00
rcx : 0x00
```

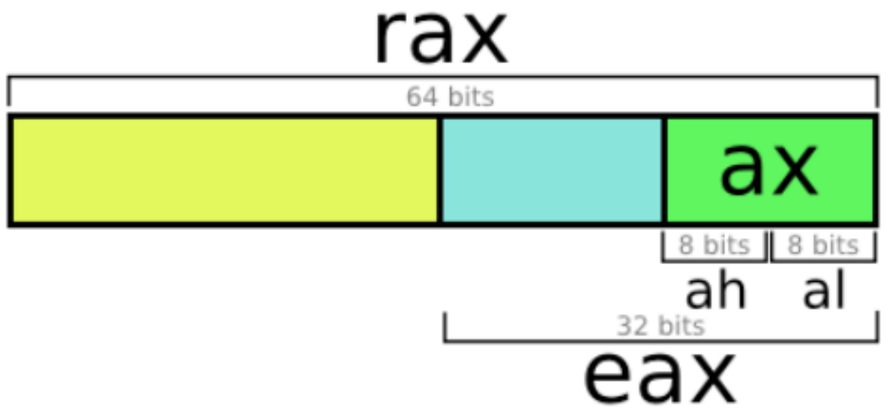
```
_start:
b012          mov al, 0x12
66 b83412     mov ax, 0x1234
b878563412    mov eax, 0x12345678
```

```
rax : 0x1234
rbx : 0x00
rcx : 0x00
rdx : 0x4ffff
```


класрум **im3ozqq4**

Операції з пам’яттю. **MOV**

Запис значення в регістр

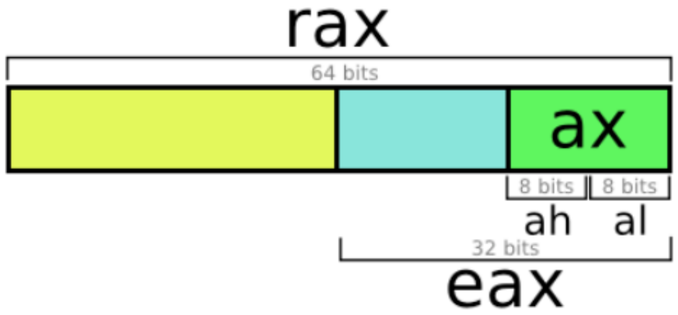



```
_start:
b012          mov al, 0x12
66 b83412     mov ax, 0x1234
b878563412    mov eax, 0x12345678
48 b8efcd…    movabs rax, 0x1234567890abcdef
b000          mov al, 0
```

```
rax : 0x1234567890abcd00
rbx : 0x00
rcx : 0x00
rdx : 0x4fffffffffff7
rsp : 0x4fffffffed0
rbp : 0x00
```

Операції з пам’яттю. **MOV**

Перенос значення одного регістра в інший




```
00401000  _start:
00401000  66 b83412          mov ax, 0x1234
00401004  bb88776655        mov ebx, 0x55667788
00401009  66 89c1            mov cx, ax
0040100c  66 89c2            mov dx, ax
```

```
rax : 0x1234
rbx : 0x55667788
rcx : 0x1234
rdx : 0x4fffffff1234
rsp : 0x4fffffffed0
```

Операції з пам’яттю. **MOV**

Читання з пам’яті в зазначений регістр

```
mov eax, [0x40102b]
```

Адреса вказується в квадратних скобках

Операції з пам’яттю. **MOV**

Оскільки регістр EAX має довжину 4 байта то саме 4 байта і буде скопійовано

```
00401000  8b04252b104000      mov eax
00401007  48 c7c701000000     mov rdi, 1
0040100e  48 8d342500204000   lea rsi
00401016  48 c7c20e000000     mov rdx, 14
0040101d  e808000000          call print
00401022  48 31ff             xor rdi, rdi
00401025  e80f000000          call exit
0040102a  print:
0040102a  55                  push rbp
0040102b  48 89e5             mov rbp, rsp
0040102e  48 c7c001000000     mov rax, 1
```

```
mov eax, [0x40102b]
```

До виконання

```
rax : 0x00
```

Після виконання

```
rax : 0x48e58948
```

Операції з пам’яттю. **MOV**

Для зручності, та можливості обчислень,
можливо адресу пам’яті записати в інший регістр,
та потім звернутися за адресою в іншому регістрі.

```
mov edx, 0x40102b
mov eax, [edx]
```

Операції з пам’яттю. **MOV**

Відповідно, існує зворотня операція
Запис вмісту регістра за адресою в пам’яті

```
mov al, 12
mov [0x401034], al
```

Операції з пам’яттю. **MOV**

Так само можна спочатку адресу записати в інший регістр.
І потім звернутися до цього регістру

```
mov al, 12
mov edx, 0x40102b
mov [edx], al
```

Операції з пам’яттю. **MOV**

Але є нюанс:
Ми [дуже легко] можемо записати в неправильну адресу пам’яті і наша програма зависне

класрум **im3ozqq4**

Операції з пам’яттю. **MOV**

таким чином маємо 4 можливих комбінації:

```
mov ax, 12

mov bx, ax

mov [ecx], ax

mov ax, [ecx]
```


класрум **im3ozqq4**

Операції з пам’яттю. **MOV**

Оскільки адреса в пам’яті описуються мінімум 4 байтами, то для опису адреси можливо використовувати тільки достатньо “широкі” регістри:

EAX, EBX, ECX, EDX
RAX, RBX, RCX, RDX

Операції з пам’яттю. **MOV**

Операція переносу з пам’яті в пам’ять минуючи процесор — не можлива:

класрум **im3ozqq4**

Інколи потрібно звернутися за адресою пам’яті “+2 байти, +4 байти”, або більш складною конструкцію вигляду

```
mov edx, [0x123456 + EBX * 4 + ECX]
```

Але такий синтаксис не працює.
В силу обмежень команди **mov**

Операції з пам’яттю. **LEA**

load effective address

Використовується як раз для таких ситуацій

```
lea edx, [0x123456 + EBX * 4 + ECX]
```

Не зважаючи на дуже схожий синтаксис,
LEA не звертається до пам’яті,
Вона тільки робить калькуляцію

```
mov eax, 10
mov ebx, 6
lea edx, [0x1000000 + EBX * 4 + EAX]
```

До виконання

```
rdx : 0x00
```

Після виконання

```
rdx : 0x01000022
```


класрум **im3ozqq4**

Для того, щоб отримати дані за “тією” адресою Потрібно додати ще одну команду, яку ми вже знаємо

```
mov eax, 10
mov ebx, 6
lea edx, [0x1000000 + EBX * 4 + EAX]
mov cx, [eax]
```

Є один нюанс, в якості “*4” можуть бути тільки 1,2,4,8

```
mov eax, 10
mov ebx, 6
lea edx, [0x1000000 + EBX * 4 + EAX]
mov cx, [eax]
```

Якщо нам потрібно скопіювати частину пам’яті,
То для цього є відповідні команди

```
rep movsb
```

Це перший випадок, коли ми не бачимо операндів.
Як це працює?

класрум **im3ozqq4**

Тут має місце “прихована семантика” (треба читати документацію)

```
mov esi, 0x100000 ; source address
mov edi, 0x200000 ; destination address
mov ecx, 16       ; count of bytes

rep movsb
```

Відповідно існують команди

```
rep movsb ; bytes
rep movsw ; words (2 bytes)
rep movsd ; double words (4 bytes)
rep movsq ; quad words (8 bytes)
```

Наступні 4 варіанти коду виконують однакову роботу

Тому використовуємо
Те що має більше сенс

```
mov cx, 16
rep movsb

mov cx, 8
rep movsw

mov cx, 4
rep movsd

mov cx, 2
rep movsq
```