

Низькорівневе програмування та програмування мікроконтролерів

З використанням мови програмування **Assembler**

Математичні операції

класрум **im3ozqq4**

Математичні Операції

Додавання. Базовий синтаксис

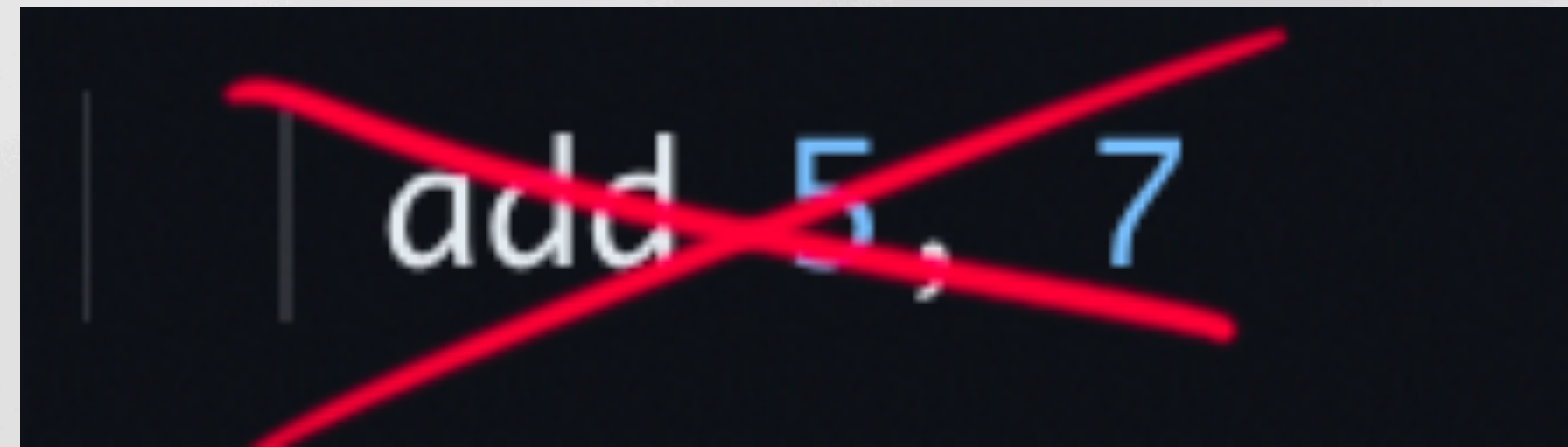
```
add dst, src
```

Семантично

```
dst = dst + src
```


Математичні Операції

Додавання



```
add 5, 7
```

Можливо хотілося б мати такий синтаксис —
але такого синтаксису нема;)

Математичні Операції

Додавання.

Щоб щось додати, потрібно його покласти в регістр

```
mov ax, 5  
add ax, 10
```


Математичні Операції

Додавання.

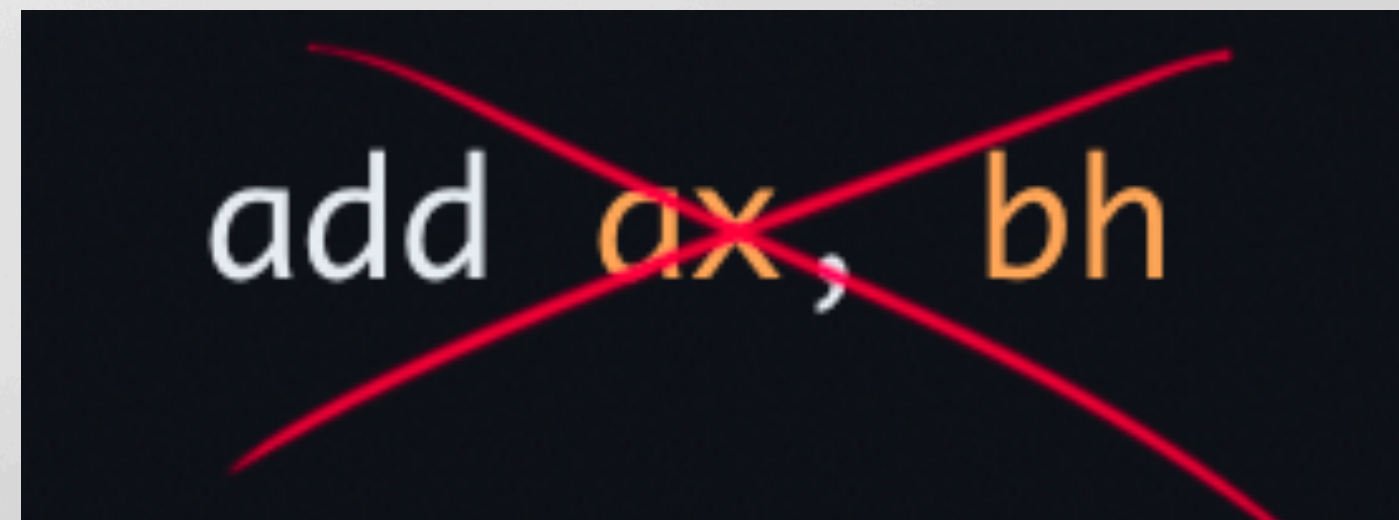
Щоб щось додати, потрібно його покласти в регістр
Або додати один регістр до іншого

```
mov ax, 5  
mov bx, 3  
add ax, bx
```

Математичні Операції

Додавання.

Регістри мають бути одного розміру



`add ax, bh`

Не скомпілюється

Математичні Операції

Додавання.

Також є можливість додавати безпосередньо дані з пам'яті

```
add ax, [0x0040100b]
```

Математичні Операції

Додавання.

Також є можливість додавати безпосередньо дані з пам'яті

```
add [0x00401000], ax
```


Математичні Операції

Додавання.
Навіть так

```
add [0x0040100a], 1
```

Математичні Операції

Додавання.
Навіть так

```
add [0x0040100a], 1
```

Але не працює (((

Математичні Операції

Бо розмір...

```
add byte    [0x0040100a], 1  
add word    [0x0040100a], 1  
add dword   [0x0040100a], 1  
add qdword  [0x0040100a], 1
```

Математичні Операції

Додавання.

Розмір має значення

Це принципово різні речі

```
add al, [0x0040100b]
```

```
add ax, [0x0040100b]
```

```
add eax, [0x0040100b]
```

```
add rax, [0x0040100b]
```


Математичні Операції

Переповнення

```
mov al, 250  
add al, 10
```

rax : 0xfa

rax : 0x04

Математичні Операції

Переповнення

```
mov al, 250  
add al, 10
```

rax : 0xfa

rax : 0x04

Якщо виникає переповнення,
то встановлюється флаг

CF = 1

Флаг — це 1 біт
Може бути або **0** або **1**

Про флаги, їх кількість та використання —
на наступній лекції

Математичні Операції

Переповнення

```
mov al, 250  
add al, 10  
adc al, 1
```

rax : 0xfa

rax : 0x04

rax : 0x06

Математичні Операції

Додавання з урахуванням флагу переповнення

```
adc dst, src
```

Семантично

```
dst = dst + src + CF
```

Математичні Операції

Додавання з урахуванням флагу переповнення.
Всі можливі комбінації

```
adc ax, bx
adc ax, [edx]
adc [ecx], ax
adc ax, 1
adc [ecx], 1
```


Математичні Операції

Віднімання

```
mov ax, 5  
mov bx, -3  
add ax, bx
```

Математичні Операції

Віднімання

```
mov ax, 5  
sub ax, 3
```


Математичні Операції

Віднімання. Загальний синтаксис

```
sub dst, src
```

Семантично

```
dst = dst - src
```

Математичні Операції

Багатий синтаксис

```
sub ax, bx  
sub ax, [edx]  
sub [ecx], ax  
sub ax, 1  
sub [ecx], 1
```


Математичні Операції

Від'ємні значення

```
mov al, 5  
sub al, 7
```

```
rax : 0x05
```

```
rax : 0xfe
```

Математичні Операції

Від'ємні значення

| | |
|-----|----------|
| -4 | 11111100 |
| -3 | 11111101 |
| -2 | 11111110 |
| -1 | 11111111 |
| 0 | 00000000 |
| 1 | 00000001 |
| 2 | 00000010 |
| 255 | 11111111 |

Математичні Операції

Як відрізнити

```
mov al, -2  
mov al, 254
```

```
rax : 0xfe
```

```
rax : 0xfe
```

Математичні Операції

Як відрізнити

```
mov al, -2  
mov al, 254
```

```
rax : 0xfe
```

```
rax : 0xfe
```

НІЯК

Математичні Операції

Але можна відрізнити

```
mov al, 253
```

```
add al, 1
```

```
mov al, 2
```

```
sub al, 4
```

```
rax : 0xfe
```

```
rax : 0xfe
```

Математичні Операції

Але можна відрізнити

```
mov al, 253  
add al, 1
```

```
mov al, 2  
sub al, 4
```

rax : 0xfe

CF = 0

rax : 0xfe

CF = 1

Математичні Операції

Оскільки є CF, то його теж можна враховувати при необхідності, по аналогії з ADC

```
mov ax, 0  
sub ax, 1  
sbb ax, 1
```

```
rax : 0x00
```

```
rax : 0xffff
```

```
rax : 0xfffd
```

```
CF = 1
```

```
CF = 0
```

Математичні Операції

Віднімання з урахуванням CF

```
sbb dst, src
```

Семантично

```
dst = dst - src - CF
```


Математичні Операції: inc

Дуже часто, особливо при операціях з пам'яттю
Нам потрібно щось на кшталт

```
add si, 1
```

Технічно це додавання 1, але

Математичні Операції: inc

Має відповідну команду

```
inc si
```

Технічно, це **add si, 1**, але семантично “наступний”

```
si = si + 1
```


Математичні Операції: dec

Дуже часто, особливо при операціях з пам'яттю
Нам потрібно щось на кшталт

```
sub si, 1
```

Технічно це додавання 1, але

Математичні Операції: dec

Має відповідну команду

```
dec si
```

Технічно, це **sub si, 1**, але семантично “попередній”

```
si = si - 1
```


Чи є різниця?

```
add ax, 1
```

```
inc ax
```

Чи є різниця?

```
add ax, 1
```

```
inc ax
```

ТАК

Чи є різниця?

```
add ax, 1
```

```
inc ax
```

Перевіряє “перехід через 0”,
установлює флаг **CF**

НЕ Перевіряє “перехід через 0”,
НЕ установлює флаг **CF**

Чи є різниця?

```
mov al, 255  
add al, 1
```

```
mov al, 255  
inc al
```

```
rax : 0x00
```

```
CF = 1
```

```
rax : 0x00
```

```
CF = 0
```


Зміна знаку

В звичайних мовах програмування $y = -x$

```
neg al
```

Зміна знаку

Приклад

```
mov al, 1  
neg al  
neg al
```

```
rax : 0x01
```

```
rax : 0xff
```

```
rax : 0x01
```


Зміна знаку

Як відрізнити **-1** та **255**?

Зміна знаку

Як відрізнити **-1** та **255**?

НІЯК

Зміна знаку

Як відрізнити **-1** та **255**?

НІЯК

Тому після

neg **al**

завжди

CF = **1**

Зміна знаку

Як відрізнити **-1** та **255**?

НІЯК

Тому після `neg al` завжди

`CF = 1`

Окрім ситуації коли

Зміна знаку

Як відрізнити **-1** та **255**?

НІЯК

Тому після `neg al` завжди

`CF = 1`

Окрім ситуації коли

`al = 0`

Множення

Синтаксис

```
mul reg
```


Множення: нюанс

Якщо при додаванні,
наприклад $255 + 255 = 510$

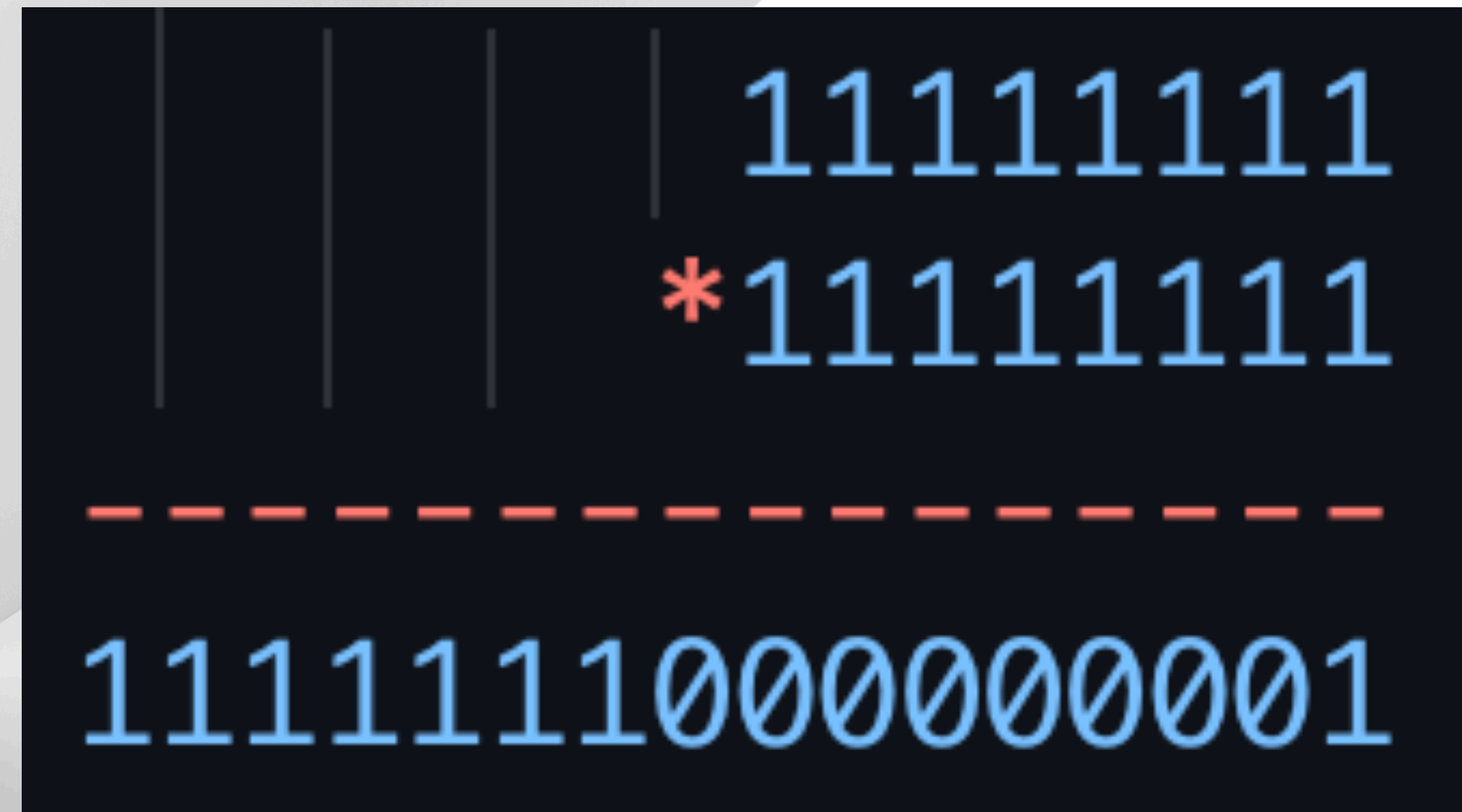
У нас "з'являється максимум
1 зайвий біт",
який ми кладемо
В регістр (благ CF)

```
11111111
+11111111
-----
111111110
```

Множення: нюанс

То при множенні,
наприклад $255 * 255 = 65025$

У нас “з’являється
ЩЕ один регістр”.
Тут флагом **CF**
не обмежитися.



Множення: нюанс

8 bit * 8 bit = 16 bits
16 bit * 16 bit = 32 bits
32 bit * 32 bit = 64 bits
64 bit * 64 bit = 128 bits

Потрібен ще один “зайвий” регістр

Множення: нюанс

Відповідно, маємо наступну логіку:

| | | |
|----------------------|----------------------|----------------------------------|
| <code>mul r8</code> | <code>mul bl</code> | <code>ax = al * bl</code> |
| <code>mul r16</code> | <code>mul bx</code> | <code>dx:ax = ax * bx</code> |
| <code>mul r32</code> | <code>mul ecx</code> | <code>edx:eax = eax * ecx</code> |
| <code>mul r64</code> | <code>mul rcx</code> | <code>rdx:rax = rax * rcx</code> |

Множення

Тут не все просто, оскільки існує знак

$$P * P = P$$

$$P * N = N$$

$$N * P = N$$

$$N * N = P$$

І з цим треба бути дуже обережно

Множення зі знаком

Має трохи інший синтаксис

```
imul reg
```

Також один з операндів завжди **al, ax, eax, rax**
Але всі деталі і нюанси аналогічні

```
mul reg
```


Ділення

Знову є нюанс.

Оскільки у нас нема float, тільки int, то:

$$20 / 3 = 6 + \text{залишок } 2$$

Тобто:

$$A / B = C + D$$

Ділення

div r8

div bl

ax / bl = al + rem ah

div r16

div bx

dx:ax / bx = ax + rem dx

div r32

div ecx

edx:eax / eax = eax + rem edx

div r64

div rcx

rdx:rax / rax = rax + rem rdx

Ділення зі знаком

Має трохи інший синтаксис

```
idiv reg
```

Також один з операндів завжди **al, ax, eax, rax**

Але всі деталі і нюанси аналогічні

```
div reg
```