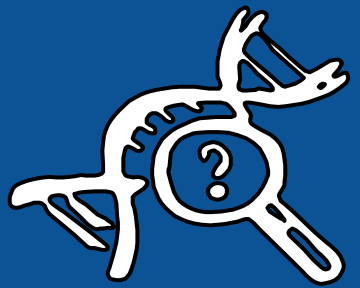
Архитектура ЭВМ и язык ассемблера

Семинар #30:

1. Изучение кодогенерации с помощью дизассемблера.
2. Знаковое/беззнаковое переполнение, регистр EFLAGS.
3. Операции над 64-битными числами.

30.01.2026

Изучение кодогенерации с помощью дизассемблера



Пересылка данных и переносимый код

```
static  uint8_t var_08bit;
static uint16_t var_16bit;
static uint32_t var_32bit;
```

```
var_08bit = 0x01U;
var_16bit = 0x0001U;
var_32bit = 0x00000001U;
```

```
080491f1  e82affffff          call    __x86.get_pc_thunk.bx
080491f6  81c30a2e0000        add     ebx, 0x2e0a
080491fc  c6835c00000001      mov     byte [ebx+0x5c], 0x1  {var_08bit}
08049203  66c7835e00000001…   mov     word [ebx+0x5e], 0x1  {var_16bit}
0804920c  c783600000000100…   mov     dword [ebx+0x60], 0x1  {var_32bit}
```

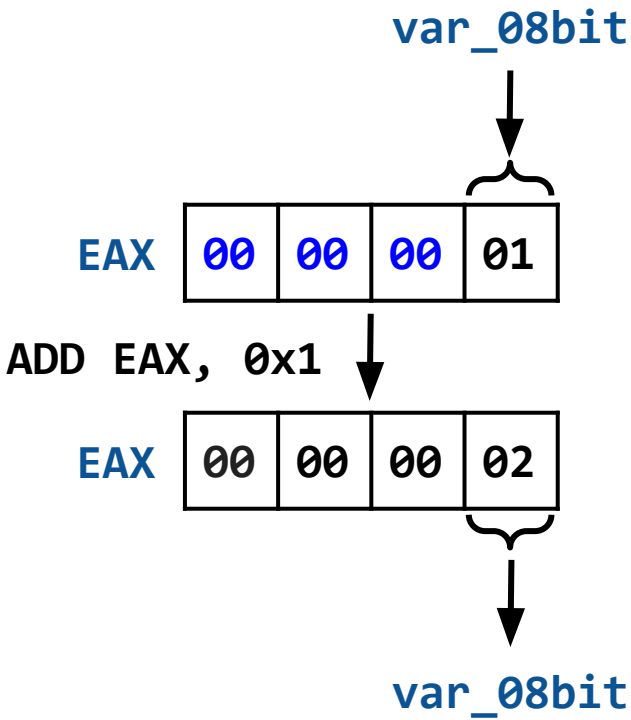
0x080491f6 (EIP) + 0x2e0a (.bss) + 0x5c (var_08bit) = 0x0804C05C

```
0804c05c  uint8_t var_08bit = 0x0
```

Сложение и вычитание

```
var_08bit += 1;
var_16bit -= 1;
var_32bit *= 10;
var_32bit /= 5;
```

```
movzx   eax, byte [ebx+0x5c]
add     eax, 0x1
mov     byte [ebx+0x5c], al
movzx   eax, word [ebx+0x5e]
sub     eax, 0x1
mov     word [ebx+0x5e], ax
```



Умножение и деление

```
// Беззнаковые 32-битные числа.
static uint32_t u32_a;
static uint32_t u32_b;
static uint32_t u32_c;
static uint32_t u32_d;

u32_c = u32_a * u32_b;
u32_d = u32_a / u32_b;

// Знаковые 32-битные числа.
static int32_t s32_a;
static int32_t s32_b;
static int32_t s32_c;
static int32_t s32_d;

s32_c = s32_a * s32_b;
s32_d = s32_a / s32_b;
```

```
mov     edx, dword [ebx+0x70]
mov     eax, dword [ebx+0x74]
imul    eax, edx
mov     dword [ebx+0x78], eax
mov     eax, dword [ebx+0x70]
mov     edi, dword [ebx+0x74]
mov     edx, 0x0
div     edi
mov     dword [ebx+0x7c], eax
mov     edx, dword [ebx+0x80]
mov     eax, dword [ebx+0x84]
imul    eax, edx
mov     dword [ebx+0x88], eax
mov     eax, dword [ebx+0x80]
mov     edi, dword [ebx+0x84]
cdq
idiv    edi
mov     dword [ebx+0x8c], eax
```

Знаковое/беззнаковое умножение

Почему и знаковое, и беззнаковое умножение через `IMUL`?

Беззнаковое умножение

`EAX = 0x200` `EDX = 0x200`

`IMUL EAX, EDX`

$EAX = (EAX * EDX) \bmod 2^{32} = 0x40000$

Знаковое умножение

`EAX = 2` `EDX = -2 = 0xFFFFFFFE = 0x10000000 - 2`

`IMUL EAX, EDX`

$EAX = (EAX * EDX) \bmod 2^{32} = (2 * (0x10000000 - 2)) \bmod 2^{32} =$

$= (2 * 0x10000000 - 4) \bmod 2^{32} = 0xFFFFFFFC = -4$

Удобное представление знаковых чисел!

Знаковое расширение для деления

```
// Знаковые 32-битные числа.
static int32_t s32_a;
static int32_t s32_b;
static int32_t s32_c;
static int32_t s32_d;

s32_c = s32_a * s32_b;
s32_d = s32_a / s32_b;
```

```
mov     eax, dword [ebx+0x80]
mov     edi, dword [ebx+0x84]
cdq
idiv    edi
mov     dword [ebx+0x8c], eax
```

CWD/CDQ/CQO—Convert Word to Doubleword/Convert Doubleword to Quadword

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
99	CWD	ZO	Valid	Valid	DX:AX := sign-extend of AX.
99	CDQ	ZO	Valid	Valid	EDX:EAX := sign-extend of EAX.
REX.W + 99	CQO	ZO	Valid	N.E.	RDX:RAX:= sign-extend of RAX.

Умножение: пример оптимизации #1

```
var_08bit += 1;
var_16bit -= 1;
var_32bit *= 10;
var_32bit /= 5;
```

```
mov     edx, dword [ebx+0x60]
mov     eax, edx
shl     eax, 0x2
add     eax, edx
add     eax, eax
mov     dword [ebx+0x60], eax
```

```
EDX = var_32bit;
EAX = EDX;
EAX <<= 2;
EAX += EDX;
EAX += EAX;
var_32bit = EAX;
```

Длительность инструкций (для моего процессора):

MOV: 1 такт
SHL: 1 такт
ADD: 1 такт
MUL: 4 такта

MOV+MUL: 5 тактов, 7 байт
MOV+SHL+ADD+ADD: 4 такта, 9 байт

Деление: пример оптимизации #2

```
var_08bit += 1;
var_16bit -= 1;
var_32bit *= 10;
var_32bit /= 5;
```

```
mov     eax, dword [ebx+0x60]
mov     edx, 0xcccccccd
mul     edx
mov     eax, edx
shr     eax, 0x2
mov     dword [ebx+0x60], eax
```

EDX = 0xCCCCCCCD =
= 0x400000001 / 5

EDX:EAX = EDX * EAX =
= 0x400000001 / 5 * EAX =
= (0x400000001 * EAX)/5

Следовательно:
EDX = (4 * EAX)/5

Итого: **EAX = (4 * EAX)/5/4**
EAX = EAX / 5

MOV+MUL+MOV+SHR: 1+4+1+1=7 тактов
MOV+DIV: 1+9=10 тактов

Битовые сдвиги и оптимизация константы

```
res0 = (ushifted << 10U);
res1 = (ushifted >> 10U);
res2 = (sshifted << 10U);
res3 = (sshifted >> 10U);
```

```
static const uint32_t ushifted = 10U;
static const int32_t sshifted  = 10;

static uint32_t res0, res1;
static  int32_t res2, res3;
```

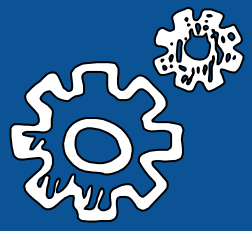
```
mov     eax, 0xa
shl     eax, 0xa  {0x2800}
mov     dword [ebx+0x64], eax  {0x2800}  {res0}
mov     eax, 0xa
shr     eax, 0xa  {0x0}
mov     dword [ebx+0x68], eax  {0x0}  {res1}
mov     eax, 0xa
shl     eax, 0xa  {0x2800}
mov     dword [ebx+0x6c], eax  {0x2800}  {res2}
mov     eax, 0xa
sar     eax, 0xa  {0x0}
mov     dword [ebx+0x70], eax  {0x0}  {res3}
```

Константы подставлены в код!

Для моего процессора:
MOV из памяти: 2 такта, 6 байт.
MOV константы: 1 такт, 5 байт.

Какие ещё оптимизации здесь возможны? (две штуки)

Знаковое/беззнаковое переполнение, регистр EFLAGS



Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

1. Не происходит **никаких переполнений**:

2. Происходит **только беззнаковое** переполнение:

3. Происходит **только знаковое** переполнение.

4. Происходит **как знаковое, так и беззнаковое** переполнение.

Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

1. Не происходит **никаких переполнений**:
   ```
   -3 + 1 = -2, 253 + 1 = 254
   11111101 + 00000001 = 11111110
   ```
2. Происходит **только беззнаковое** переполнение:

3. Происходит **только знаковое** переполнение.

4. Происходит **как знаковое, так и беззнаковое** переполнение.

Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

1. Не происходит **никаких переполнений**:
   `-3 + 1 = -2, 253 + 1 = 254`
   `11111101 + 00000001 = 11111110`
2. Происходит **только беззнаковое** переполнение:
   `255 + 1 = 0, -1 + 1 = 0`
   `11111111 + 00000001 = 00000000`
3. Происходит **только знаковое** переполнение.

4. Происходит **как знаковое, так и беззнаковое** переполнение.

Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

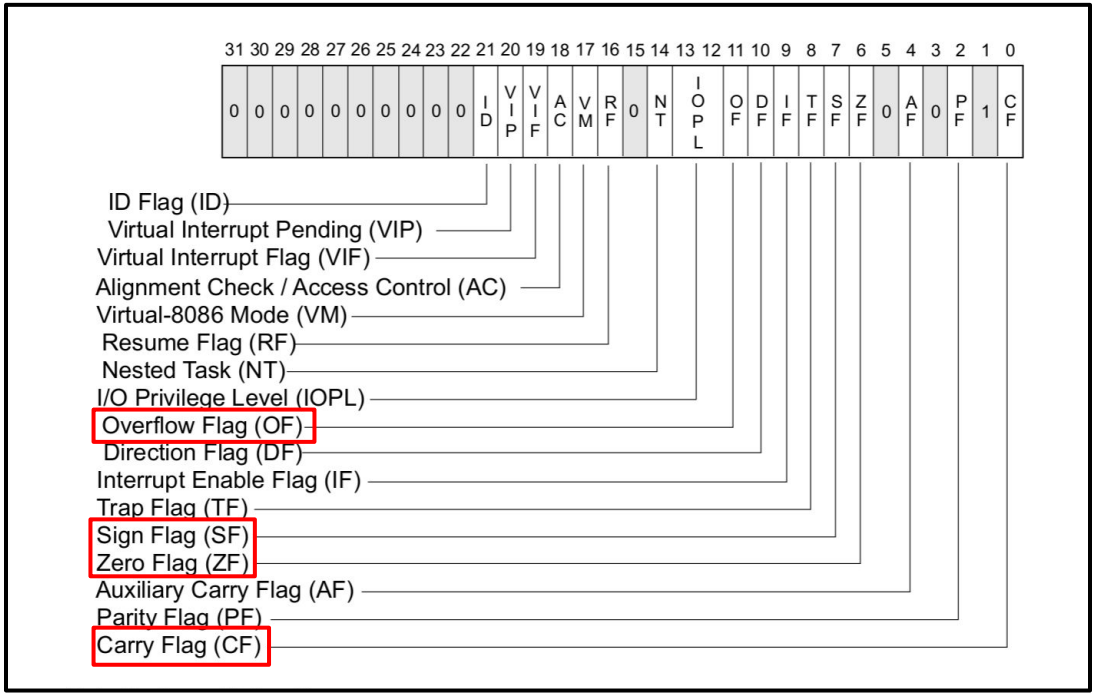
1. Не происходит **никаких переполнений**:
   `-3 + 1 = -2, 253 + 1 = 254`
   `11111101 + 00000001 = 11111110`
2. Происходит **только беззнаковое** переполнение:
   `255 + 1 = 0, -1 + 1 = 0`
   `11111111 + 00000001 = 00000000`
3. Происходит **только знаковое** переполнение.
   `127 + 127 = 254, 127 + 127 = -2`
   `01111111 + 01111111 = 11111110`
4. Происходит **как знаковое, так и беззнаковое** переполнение.

Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

1. Не происходит **никаких переполнений**:
   ```
   -3 + 1 = -2, 253 + 1 = 254
   11111101 + 00000001 = 11111110
   ```
2. Происходит **только беззнаковое** переполнение:
   ```
   255 + 1 = 0, -1 + 1 = 0
   11111111 + 00000001 = 00000000
   ```
3. Происходит **только знаковое** переполнение.
   ```
   127 + 127 = 254, 127 + 127 = -2
   01111111 + 01111111 = 11111110
   ```
4. Происходит **как знаковое, так и беззнаковое** переполнение.
   ```
   128 + 128 = 0, -128 + -128 = 0
   10000000 + 10000000 = 00000000
   ```

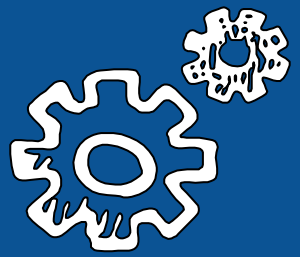
Регистр EFLAGS



OF = “знаковое переполнение” **ZF** = “результат равен 0”

CF = “беззнаковое переполнение” **SF** = “результат отрицателен”

Операции над 64-битными числами



Сложение 64-битных чисел

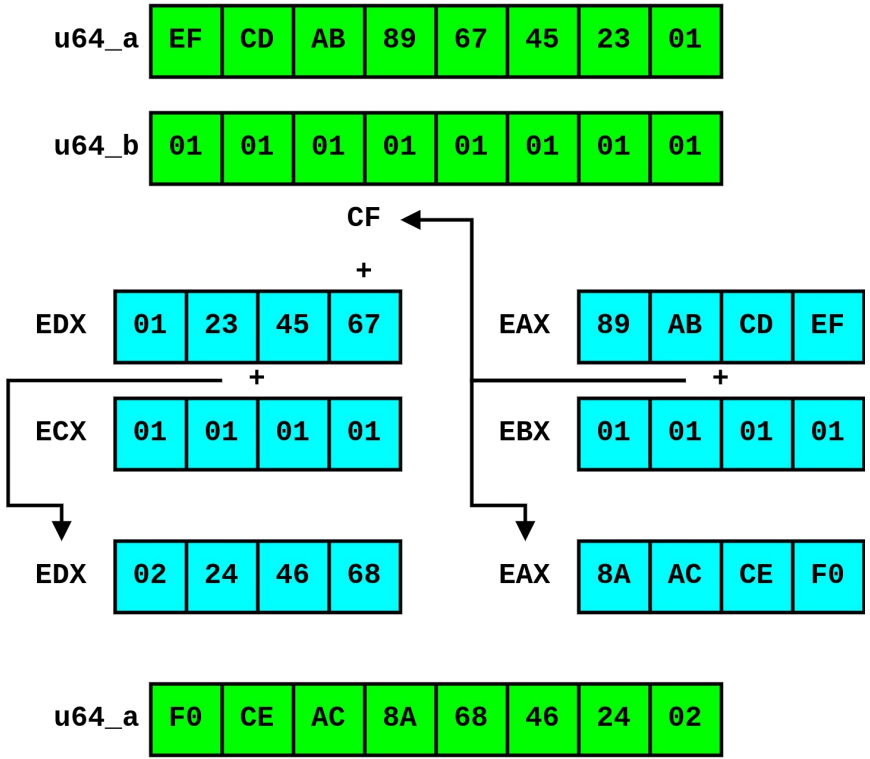
```
; edx:eax = qword [u64_a]
mov     eax, dword [u64_a + 0]
mov     edx, dword [u64_a + 4]

; ecx:ebx = qword [var64bit_b]
mov     ebx, dword [u64_b + 0]
mov     ecx, dword [u64_b + 4]

; eax = eax + ebx
; Выставляем флаг переноса CF (Carry Flag)
add     eax, ebx

; edx = edx + ecx + CF
; Используем флаг переноса CF
adc     edx, ecx

; qword [var64bit_a] = edx:eax
mov     dword [u64_a + 0], eax
mov     dword [u64_a + 4], edx
```



Умножение 64-битных чисел

```
// Беззнаковые 64-битные числа.
static uint64_t u64_a;
static uint64_t u64_b;
static uint64_t u64_c;
static uint64_t u64_d;

u64_c = u64_a * u64_b;
u64_d = u64_a / u64_b;
```

```
mov     esi, dword [ebx+0x90]  {u64_a.1618}
mov     edi, dword [ebx+0x94]  {u64_a.1618+4}
mov     eax, dword [ebx+0x98]  {u64_b.1619}
mov     edx, dword [ebx+0x9c]  {u64_b.1619+4}
mov     ecx, edi
imul    ecx, eax
mov     dword [ebp-0x1c {var_20}], ecx
mov     ecx, edx
imul    ecx, esi
add     ecx, dword [ebp-0x1c {var_20}]
mul     esi
add     ecx, edx
mov     edx, ecx
mov     dword [ebx+0xa0], eax  {u64_c.1620}
mov     dword [ebx+0xa4], edx  {u64_c.1620+4}
```

Умножение 64-битных чисел

$$u64\_a = A * 2^{32} + B$$

$$u64\_b = C * 2^{32} + D$$

$$RSLT = (u64\_a * u64\_b) \bmod 2^{64} =$$

$$AC * 2^{64} + (BC + AD) * 2^{32} + BD$$

$$IMUL_1: X_1 = (A * D) \bmod 2^{32}$$

$$IMUL_2: X_2 = (B * C) \bmod 2^{32}$$

$$MUL: X_3 = (B * D)$$

$$RSLT = (X_1 + X_2) * 2^{32} + X_3$$

Назначение регистров:

B=ESI A=EDI D=EAX C=EDX

X_1=ECX X_2=ECX X_3,RSLT=EDX:EAX

```
mov     esi, dword [ebx+0x90]  {u64_a.1618}
mov     edi, dword [ebx+0x94]  {u64_a.1618+4}
mov     eax, dword [ebx+0x98]  {u64_b.1619}
mov     edx, dword [ebx+0x9c]  {u64_b.1619+4}
mov     ecx, edi
imul    ecx, eax
mov     dword [ebp-0x1c {var_20}], ecx
mov     ecx, edx
imul    ecx, esi
add     ecx, dword [ebp-0x1c {var_20}]
mul     esi
add     ecx, edx
mov     edx, ecx
mov     dword [ebx+0xa0], eax  {u64_c.1620}
mov     dword [ebx+0xa4], edx  {u64_c.1620+4}
```

Деление 64-битных чисел

```
// Беззнаковые 64-битные числа.
static uint64_t u64_a;
static uint64_t u64_b;
static uint64_t u64_c;
static uint64_t u64_d;

u64_c = u64_a * u64_b;
u64_d = u64_a / u64_b;
```

```
mov     eax, dword [ebx+0x90]  {u64_a.1618}
mov     edx, dword [ebx+0x94]  {u64_a.1618+4}
mov     esi, dword [ebx+0x98]  {u64_b.1619}
mov     edi, dword [ebx+0x9c]  {u64_b.1619+4}
push    edi {var_30}
push    esi {var_34}
push    edx {var_38}
push    eax {var_3c}
call    __udivdi3
add     esp, 0x10
mov     dword [ebx+0xa8], eax  {u64_d.1621}
mov     dword [ebx+0xac], edx  {u64_d.1621+4}
```

Вызов встроенной функции компилятора!

`__udivdi3`, `__divdi3` – код на C.

`__udivdi3` – деление в столбик в двоичной системе счисления.

`__divdi3` – деление в столбик, много граничных случаев.

Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com