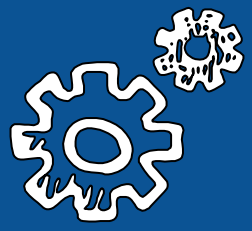
Архитектура ЭВМ и язык ассемблера

Семинар #30:

1. Регистр EFLAGS.
2. Операции над 64-битными числами.
3. Условные перемещения.

15.04.2025

Знаковое/беззнаковое переполнение, регистр EFLAGS



Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

1. Происходит **только беззнаковое** переполнение:

2. Происходит **только знаковое** переполнение.

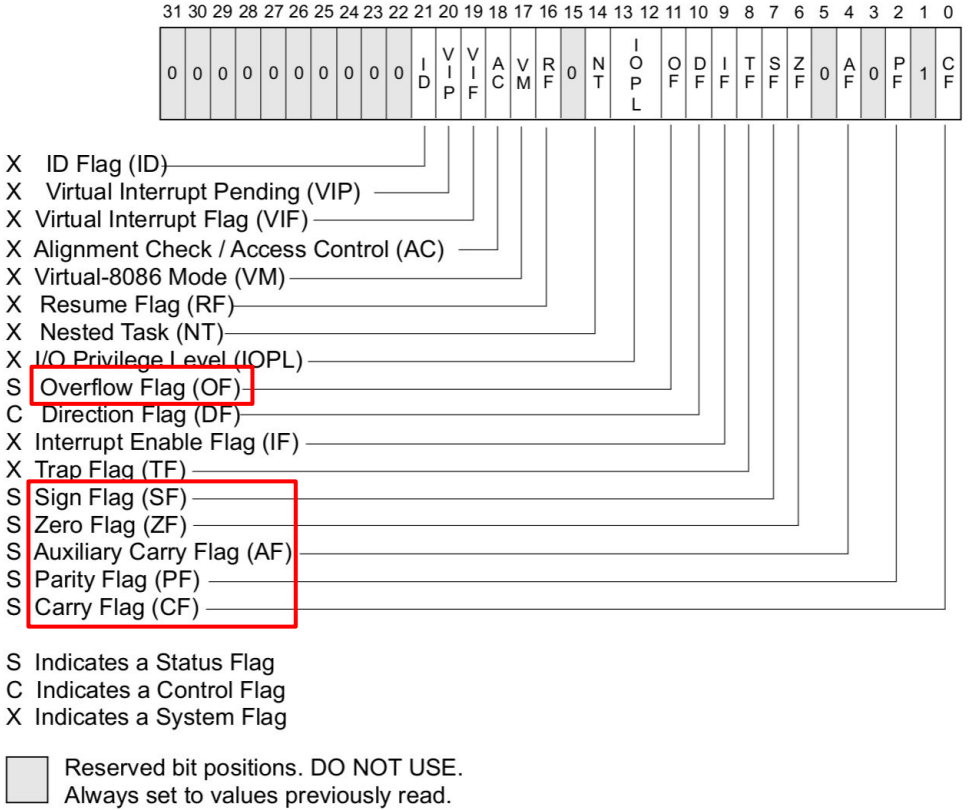
3. Происходит **как знаковое, так и беззнаковое** переполнение.

Знаковое/беззнаковое переполнение

Приведите пример 8-битных значений, при сложении которых:

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

Регистр EFLAGS



Регистр EFLAGS

3.4.3.1 Status Flags

The status flags (bits 0, 2, 4, 6, 7, and 11) of the EFLAGS register indicate the results of arithmetic instructions, such as the ADD, SUB, MUL, and DIV instructions. The status flag functions are:

CF (bit 0)	**Carry flag** — Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.
PF (bit 2)	**Parity flag** — Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.
AF (bit 4)	**Auxiliary Carry flag** — Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binary-coded decimal (BCD) arithmetic.
ZF (bit 6)	**Zero flag** — Set if the result is zero; cleared otherwise.
SF (bit 7)	**Sign flag** — Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)
OF (bit 11)	**Overflow flag** — Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.

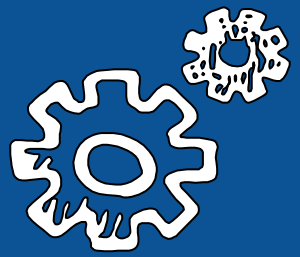
OF = "знаковое переполнение"

CF = "беззнаковое переполнение"

ZF = "результат равен 0"

SF = "результат отрицателен"

Операции над 64-битными числами



Операции с 32-битными числами

```
// 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 расширение

```
// 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.

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}
```

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;
```

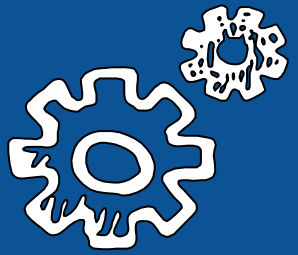
```
mov     eax, dword [ebx+0x90]  {u64_a.1618}
mov     edx, dword [ebx+0x94]  {u64_a.1618+4}
mov     esi, dword [ebx+0x98]  {u64_b.1619}
mov     edi, dword [ebx+0x9c]  {u64_b.1619+4}
push    edi {var_30}
push    esi {var_34}
push    edx {var_38}
push    eax {var_3c}
call    __udivdi3
add     esp, 0x10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

Знаковые/беззнаковые операции

```
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}
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}
```

```
mov     esi, dword [ebx+0xb0]  {s64_a.1622}
mov     edi, dword [ebx+0xb4]  {s64_a.1622+4}
mov     eax, dword [ebx+0xb8]  {s64_b.1623}
mov     edx, dword [ebx+0xbc]  {s64_b.1623+4}
mov     ecx, edi
imul    ecx, eax
mov     dword [ebp-0x1c {var_20_1}], ecx
mov     ecx, edx
imul    ecx, esi
add     ecx, dword [ebp-0x1c {var_20_1}]
mul     esi
add     ecx, edx
mov     edx, ecx
mov     dword [ebx+0xc0], eax  {s64_c.1624}
mov     dword [ebx+0xc4], edx  {s64_c.1624+4}
mov     eax, dword [ebx+0xb0]  {s64_a.1622}
mov     edx, dword [ebx+0xb4]  {s64_a.1622+4}
mov     esi, dword [ebx+0xb8]  {s64_b.1623}
mov     edi, dword [ebx+0xbc]  {s64_b.1623+4}
push    edi {var_30_1}
push    esi {var_34_1}
push    edx {var_38_1}
push    eax {var_3c_1}
call    __divdi3
add     esp, 0x10
mov     dword [ebx+0xc8], eax  {s64_d.1625}
mov     dword [ebx+0xcc], edx  {s64_d.1625+4}
```

Условные перемещения



Условные перемещения

```
; Считываем два числа из консоли.
call io_get_dec
mov ebx, eax

call io_get_dec
mov ecx, eax

; Производим сравнение.
cmp ebx, ecx

; Выполняем условное перемещение.
cmovl eax, dword [cmp_neg]
cmove eax, dword [cmp_zer]
cmovg eax, dword [cmp_pos]

; Print result:
call io_print_dec
```

Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com