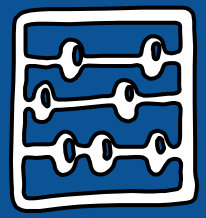
Архитектура ЭВМ и язык ассемблера

Семинар #31:

1. Побитовые операции и битовые маски.
2. Условные ветвления.
3. Условные перемещения, вычисление модуля числа.
4. Игра: декомпиляция в уме.

05.02.2025

Побитовые операции и битовые маски



Побитовые операции и битовые маски

Операция	Пример	Вычисление
Логическое И	`and eax, ebx`	`eax = eax & ebx` `0001 = 0011 & 0101`
Логическое ИЛИ	`or eax, ebx`	`eax = eax | ebx` `0111 = 0011 | 0101`
Логическое НЕ	`not eax`	`eax = ~eax` `0011 = ~1100`
Исключающее ИЛИ	`xor eax, ebx`	`eax = eax ^ ebx` `0110 = 0011 ^ 0101`
Сдвиг влево	`shl eax, bl`	`eax = eax << bl`
Сдвиг вправо	`shr eax, bl`	`eax = eax >> bl`

Побитовые операции и битовые маски

Операция	Пример	Вычисление
Бит N	`mov eax, 1` `shl eax, N`	`eax = (1 << N)` `N = 5, eax = 00100000`

Биты (N-1:0)	`mov eax, 1`
	`shl eax, N`
	`sub eax, 1`
		`eax = (1 << N) - 1`
		`N = 5, eax = 00011111`
		`N = 3, eax = 00000111`

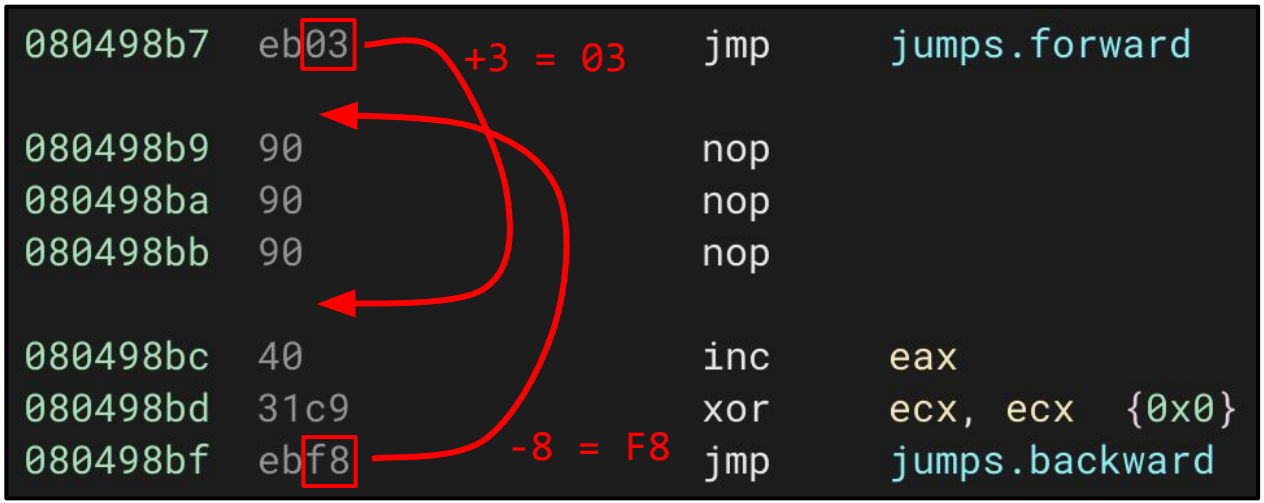
Битовая маска (N-1:K)	`mov eax, 1`
	`shl eax, N-K`
	`sub eax, 1`
	`shl eax, K`
		`eax = ((1 << (N-K)) - 1) << K`
		`N = 6, K = 3, eax = 00111000`
		`N = 5, K = 0, eax = 00011111`

Извлечение битов из регистра	`mov eax, 1`
	`shl eax, N-K`
	`sub eax, 1`
	`shl eax, K`
	`and eax, ebx`
		`N = 6, K = 3`
		`mask = 00111000`
		`ebx = 01011101`
		`eax = 00011000`

Условные ветвления

Безусловное ветвление

```
jumps:
    jmp .forward
.backward:
    nop
    nop
    nop
.forward:
    inc eax
    xor ecx, ecx
    jmp .backward

    xor eax, eax
    ret
```



Условное ветвление

```
mov     eax, dword [s32_c]

cmp     eax, 0
jge     .do_not_negate
neg     eax
.do_not_negate:

mov     dword [s32_d], eax
```

```
section .data
s32_c   dd      -204
s32_d   dd      0
```

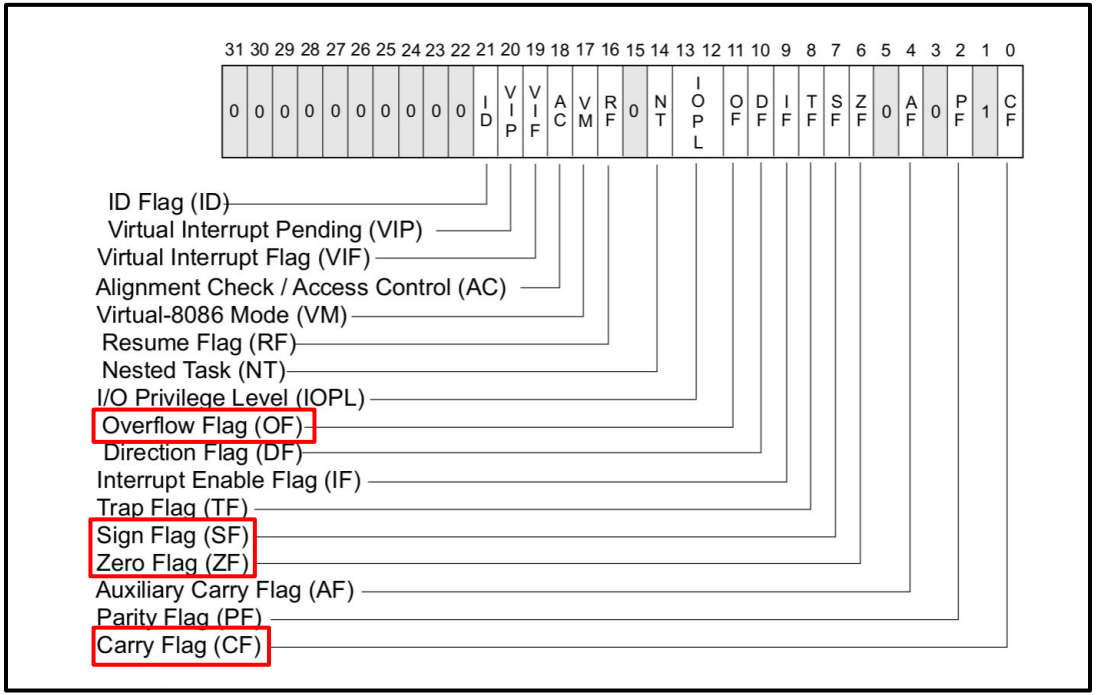
Вычисление модуля числа

`CMP (CoMPare)`:
- Вычисляет: `EAX - 0`.
- Не сохраняет результат.
- Записывает `EFLAGS`.

`JGE (Jump Greater Equal)`:
- Проверяет `EFLAGS`.
- Выполняет прыжок по адресу (запись в `EIP`).

Адрес задан меткой `.do_not_negate`

Регистр EFLAGS



OF = “знаковое переполнение”

CF = “беззнаковое переполнение”

ZF = “результат равен 0”

SF = “результат отрицателен”

Инструкции для условных ветвлений

КОП		Флаги	Описание
JO		OF = 1	overflow
JS		SF = 1	sign
JZ	JE	ZF = 1	zero/equal
JC	JB	CF = 1	below
	JBE	CF = 1 или ZF = 1	below or equal
JNC	JAE	CF = 0	above or equal
	JA	CF = 0 и ZF = 0	above
	JL	SF ≠ OF	less
	JLE	SF ≠ OF или ZF = 1	less or equal
	JGE	SF = OF	greater or equal
	JG	SF = OF и ZF = 0	greater
JECXZ		ECX = 0	

less/greater – знаковое сравнение

below/above – беззнаковое сравнение

Примеры ветвлений

```
static uint32_t u32_a, u32_b;
static  int32_t s32_c, s32_d;

// Пример #1.
if (u32_a > u32_b)
{
    return;
}

// Пример #2.
if (u32_a != u32_b)
{
    return;
}

// Пример #3.
if (u32_a <= u32_b && s32_c >= s32_d)
{
    return;
}
```

```
    ; Пример #1
    mov     eax, dword [u32_a]
    mov     ebx, dword [u32_b]
    cmp     eax, ebx
    jbe     .not_a_above_b
    ret
.not_a_above_b:

    ; Пример #2
    mov     eax, dword [u32_a]
    mov     ebx, dword [u32_b]
    cmp     eax, ebx
    je      .not_a_neq_b
    ret
.not_a_neq_b:
```

Примеры ветвлений

```
static uint32_t u32_a, u32_b;
static  int32_t s32_c, s32_d;

// Пример #1.
if (u32_a > u32_b)
{
    return;
}

// Пример #2.
if (u32_a != u32_b)
{
    return;
}

// Пример #3.
if (u32_a <= u32_b && s32_c >= s32_d)
{
    return;
}
```

```
    ; Пример #3
    mov     eax, dword [u32_a]
    mov     ebx, dword [u32_b]
    cmp     eax, ebx
    jg      .skip_if3

    mov     ecx, dword [s32_c]
    mov     edx, dword [s32_d]
    cmp     ecx, edx
    jl      .skip_if3
    ret
.skip_if3:
```

Примеры ветвлений

```
// Пример #4.
if (s32_c == s32_d)
{
    return;
}

// Пример #5.
if (s32_c > s32_d)
{
    return;
}

// Пример #6.
if (s32_c >= s32_d || u32_a > u32_b)
{
    return;
}
```

```
    ; Пример #4
    mov     ecx, dword [s32_c]
    mov     edx, dword [s32_d]
    cmp     ecx, edx
    jne     .not_c_eq_d
    ret
.not_c_eq_d:

    ; Пример #5
    mov     ecx, dword [s32_c]
    mov     edx, dword [s32_d]
    cmp     ecx, edx
    jle     .not_c_gt_d
    ret
.not_c_gt_d:
```

Примеры ветвлений

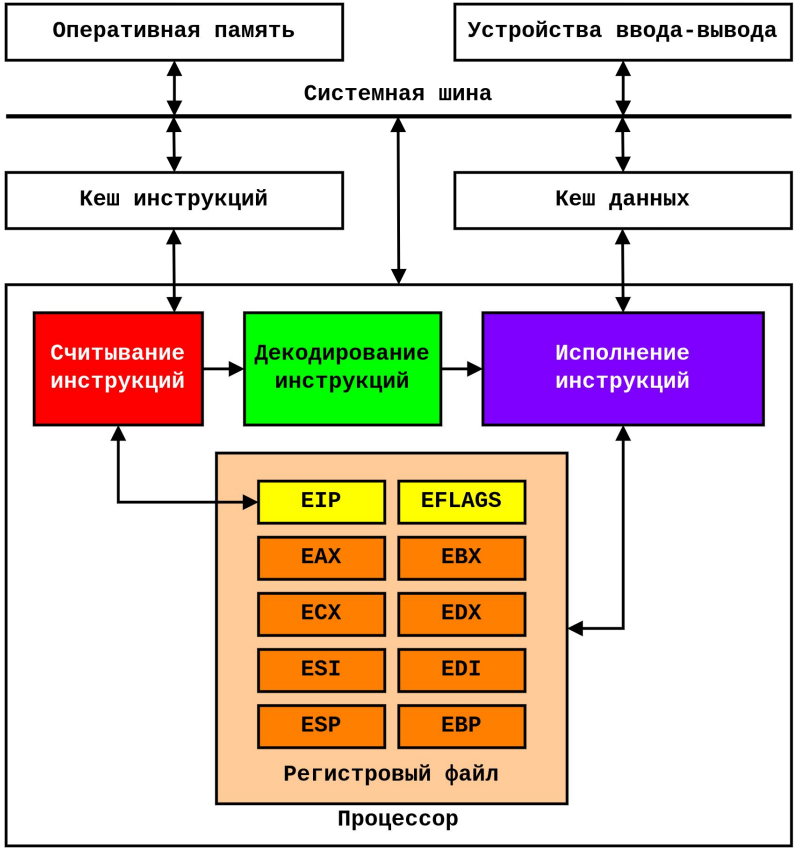
```
// Пример #4.
if (s32_c == s32_d)
{
    return;
}

// Пример #5.
if (s32_c > s32_d)
{
    return;
}

// Пример #6.
if (s32_c >= s32_d || u32_a > u32_b)
{
    return;
}
```

```
    ; Пример #6
    mov     ecx, dword [s32_c]
    mov     edx, dword [s32_d]
    cmp     ecx, edx
    jge     .enter_if6

    mov     eax, dword [u32_a]
    mov     ebx, dword [u32_b]
    cmp     eax, ebx
    ja      .enter_if6

    jmp     .skip_if6
.enter_if6:
    ret
.skip_if6:
```

Простейшая модель процессора

Алгоритм работы процессора:

1. Считывание инструкции по EIP (**fetch**)
2. Декодирование инструкции. (**decode**)
3. Считывание операндов.
4. Исполнение (**execute**):
   - Обновление регистров общего назначения, EIP, EFLAGS.
5. Запись результата в память.



Условные перемещения

Условные перемещения

```
mov     eax, dword [s32_c]

; ebx = -eax
mov     ebx, 0
sub     ebx, eax

cmp     eax, 0
cmovl   eax, ebx

mov     dword [s32_d], eax
```

Вычисление модуля числа

`CMP (CoMPare)`:
- Вычисляет: `EAX - 0`.
- Не сохраняет результат.
- Записывает `EFLAGS`.

`CMOVL (Conditional MOV Less)`:
- Проверяет `EFLAGS`.
- При выполнении условия выполняет `mov eax, ebx`

Пересылка только
между регистрами!

Игра: декомпиляция в уме

Раунд 0/3

```
mov     eax, dword [a]
cmp     eax, dword [b]
cmovb   eax, dword [b]
mov     dword [x], eax
```

(A)
```
int a, b, x;
x = (a < b)? b : a;
```

(B)
```
unsigned a, b, x;
x = (a < b)? b : a;
```

(C)
```
int a, b, x;
x = (a > b)? b : a;
```

(D)
```
unsigned a, b, x;
x = (a > b)? b : a;
```

Раунд 0/3

```
mov     eax, dword [a]
cmp     eax, dword [b]
cmovb   eax, dword [b]
mov     dword [x], eax
```

(A)

```
int a, b, x;
x = (a < b)? b : a;
```

(B)

```
unsigned a, b, x;
x = (a < b)? b : a;
```

(C)

```
int a, b, x;
x = (a > b)? b : a;
```

(D)

```
unsigned a, b, x;
x = (a > b)? b : a;
```

Раунд 1/3

```
mov     eax, dword [a]
mov     edx, dword [a + 4]
add     eax, dword [b]
adc     edx, dword [b + 4]
sub     eax, dword [c]
sbb     edx, dword [c + 4]
mov     dword [a], eax
mov     dword [a + 4], eax
```

(A)

```
uint64_t a, b, c;
a = a + b - c;
```

(B)

```
int64_t a, b, c;
a = a + b - c;
```

(C)

```
uint32_t a[2], b[2], c[2];
a[0] = a[0] + b[0] - c[0];
a[1] = a[1] + b[1] - c[1];
```

(C)

```
int32_t a[2], b[2], c[2];
a[0] += b[0] - c[0];
a[1] += b[1] - c[1];
```

Раунд 1/3

```
mov     eax, dword [a]
mov     edx, dword [a + 4]
add     eax, dword [b]
adc     edx, dword [b + 4]
sub     eax, dword [c]
sbb     edx, dword [c + 4]
mov     dword [a], eax
mov     dword [a + 4], eax
```

(A)

```
uint64_t a, b, c;
a = a + b - c;
```

(B)

```
int64_t a, b, c;
a = a + b - c;
```

(C)

```
uint32_t a[2], b[2], c[2];
a[0] = a[0] + b[0] - c[0];
a[1] = a[1] + b[1] - c[1];
```

(C)

```
int32_t a[2], b[2], c[2];
a[0] += b[0] - c[0];
a[1] += b[1] - c[1];
```

Раунд 2/3

```
    cmp     dword [a], 0
    je      L1
    mov     eax, dword [b]
    cdq
    idiv    dword [a]
    jmp     L2
L1:
    mov     eax, 0
L2:
    mov     dword [x], eax
```

(A)

```
uint32_t a, b, x;
x = a ? (b % a) : 0;
```

(B)

```
uint32_t a, b, x;
x = (a == 0)? 0 : (b / a);
```

(C)

```
int32_t a, b, x;
x = a ? (b / a) : 0;
```

Раунд 2/3

```
    cmp     dword [a], 0
    je      L1
    mov     eax, dword [b]
    cdq
    idiv    dword [a]
    jmp     L2
L1:
    mov     eax, 0
L2:
    mov     dword [x], eax
```

(A)

```
uint32_t a, b, x;
x = a ? (b % a) : 0;
```

(B)

```
uint32_t a, b, x;
x = (a == 0)? 0 : (b / a);
```

(C)

```
int32_t a, b, x;
x = a ? (b / a) : 0;
```

Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com