Контра по асэмбляру

Номер 1

`A = 0x5FC31847`

В памяти число выглядит как `[0x47], [0x18], [0xC3], [0x5F]`, последовательность байтов `big-endian` - сначала младший байт, в конце старший.

`MOVSX EAX, WORD [A + 1]`, выдергивает два байта (`sizeof(word) == 2`), начиная с позиции 1, т.е. `[0x18], [0xC3]`. Как число это выглядит так: `0xC318` (помним про big-endian).

Так как `MOV_S_X` (т.е. знаковый перенос, а не беззнаковый `MOVZX`), а у `0xC318` первый знаковый бит == 1, то `EAX == 0xFFFFC318`.

`AL` (A lower) - первые 8 байт, `AH` (A high) - вторые 8 байт, `AX` - первые 16 байт. `AH == 0xC3`.

`192 == 0xC0, AH - 192 = 0x03`, получаем `EAX == 0xFFFF0318`.

Флаги посчитались не для всего `EAX`, а именно для операции `AH - 192`.

1. `CF` - вышли ли мы за пределы беззнакового пространства. `0xC3 = 195, 195 - 192 = 3`. **Ответ = 0**.

2. `OF` - вышли ли мы за пределы знакового пространства при операции (если представить все числа как знаковые). В этом представлении `195 == 256-195 == -61, 192 == 256-192 == -64. -61 - (-64) == 3`. **Ответ = 0**.

3. `SF` - может ли число одновременно быть представлено как знаково и как беззнаково (т.е. равен ли старший бит 1?) **Ответ = 0**.

4. `ZF` - равен ли результат 0. **Ответ = 0**.

Номер 2

Условие:
`short *pa, pb[2], x; *pa++ = pb[1] / x;`
Побочные эффекты:

1. `*pa = pb[1] / x`

2. `pa++`

Память:

```
section .bss
pa resd 1 ; short*
pb resw 2 ; short[2]
x resw 1 ; short
```

Код:

```
movsx eax, word [pb + 1]
movsx ecx, word [x]
cdq
idiv ecx
mov edx, word [pa]
mov word [edx], eax
```

Номер 3

```
struct tree {
    short x; // +2 bytes
    // +2 bytes align
    struct tree* l; // +4 bytes
    struct tree* r; // +4 bytes
};

num_pos_leaf:
    push ebp
    mov ebp, esp
    sub esp, 8 ; выравнивание по 16 байт

    mov edx, [ebp + 8] ; edx = t

    ; проверка на то что это лист
    mov eax, [edx + 4]
    cmp eax, 0
    jne .non_leaf
    mov eax, [edx + 8]
    cmp eax, 0
    jne .non_leaf

    ; это лист!
    ; запишем в eax 1, если число x положительное (его старший бит == 0)
    ; для этого сделаем сдвиг направо на 7 бит и xor 1
    ; eax = ((x >> 7) ^ 1)
    ; Внимание - 0 НЕ положительное!
    movzx eax, word [edx] ; eax = x
    cmp eax, 0
    je .end
    shr eax, 7
    xor eax, 1
    je .end

.non_leaf:
    ; идем направо и налево
    mov eax, 0
```

```
    mov ecx, [edx + 4] ; go to *l
    call num_pos_leaf
    mov ecx, [edx + 8] ; go to *r
    mov edx, eax ; сохраняем старое значение eax
    call num_pos_leaf
    add eax, edx ; eax = num_pos_leaf(t->l) + num_pos_leaf(t->r)

.end:
    leave
    ret
```

Номер 4

-fomit-frame-pointer - Не используем `push ebp` и `leave`, следим за указателем esp сами.

fastcall - Первые два аргумента помещаются в `ecx` и `edx` соответственно, остальные на стек. Стек чистим сами, как и в **stdcall**.

```
diff:
    ; ecx == x
    ; edx == y
    ; dword [esp + 4] == n
    mov eax, [esp + 4]
    sub esp, 4
.loop:
    dec eax

    ; сравнение ecx[eax] и edx[eax]
    push ecx
    push edx
    movzx ecx, byte [ecx + eax]
    movzx edx, byte [edx + eax]
    cmp ecx, edx
    jne .non_equal

    ; [esp]++ через eax
    push eax
    mov eax, [esp]
    inc eax
    mov dword [esp], eax
    pop eax

.non_equal:
    pop edx
    pop ecx
```

```
   cmp eax, 0
   jne .loop

   mov eax, [esp]
   add esp, 4

   ret 4 ; чистим сами лишние 4 байта для аргументов
```

Номер 5

```
struct matrix {
   int n; // +4 bytes
   unsigned char *v; // +4 bytes
};

id_k:
   ; [esp + 4] == k
   ; [esp + 5] == n

   push ebp
   mov ebp, esp
   ; [esp + 8] == k
   ; [esp + 9] == n

   ; [ebp + 8] == k
   ; [ebp + 9] == n

   ; Структура фрейма:
   ; 4 - ebp
   ; 24 - доп память
   ; --- 8 байт -> [esp + 16] - struct matrix m ([esp+20] - m.n, [esp+16] - m.v)
   ; --- 16 байт -> [esp + 8] - ассемблер это жопа бомжа
   ; --- 4 байта -> [esp + 4] - второй аргумент calloc
   ; --- 4 байта -> [esp] - первый аргумент calloc
   ; 4 - адрес вызова calloc
   ; сумма = 32 - делится на 16

   sub esp, 24

   ; первый аргумент calloc
   mov eax, [ebp + 9]
   imul eax, eax
   mov dword [esp], eax

   ; второй аргумент calloc
   mov dword [esp + 4], 1
```

```
    ; m.n = n
    mov eax, [ebp + 9]
    mov dword [esp + 20], eax

    ; вызов calloc, сохранение в m.v
    call calloc
    mov dword [esp + 16], eax

    ; ecx = n, цикл
    mov ecx, [ebp + 9]
.loop
    cmp ecx, 1
    je .NAXYI
    dec ecx

    ; m.v[i * n + i] = k
    mov eax, ecx
    inc eax
    imul eax, dword [ebp + 9] ; eax = i * n + i
    mov edx, [esp + 16] ; edx = m.v
    mov al, byte [ebp + 8]
    mov byte [edx + eax], al ; m.v[i * n + i] = k

    jmp .loop
.NAXYI:
    ; Че делать-то дальше??? Вроде все, m никуда не девается
    mov eax, [esp + 20] ; eax = m

    leave
    ret
```