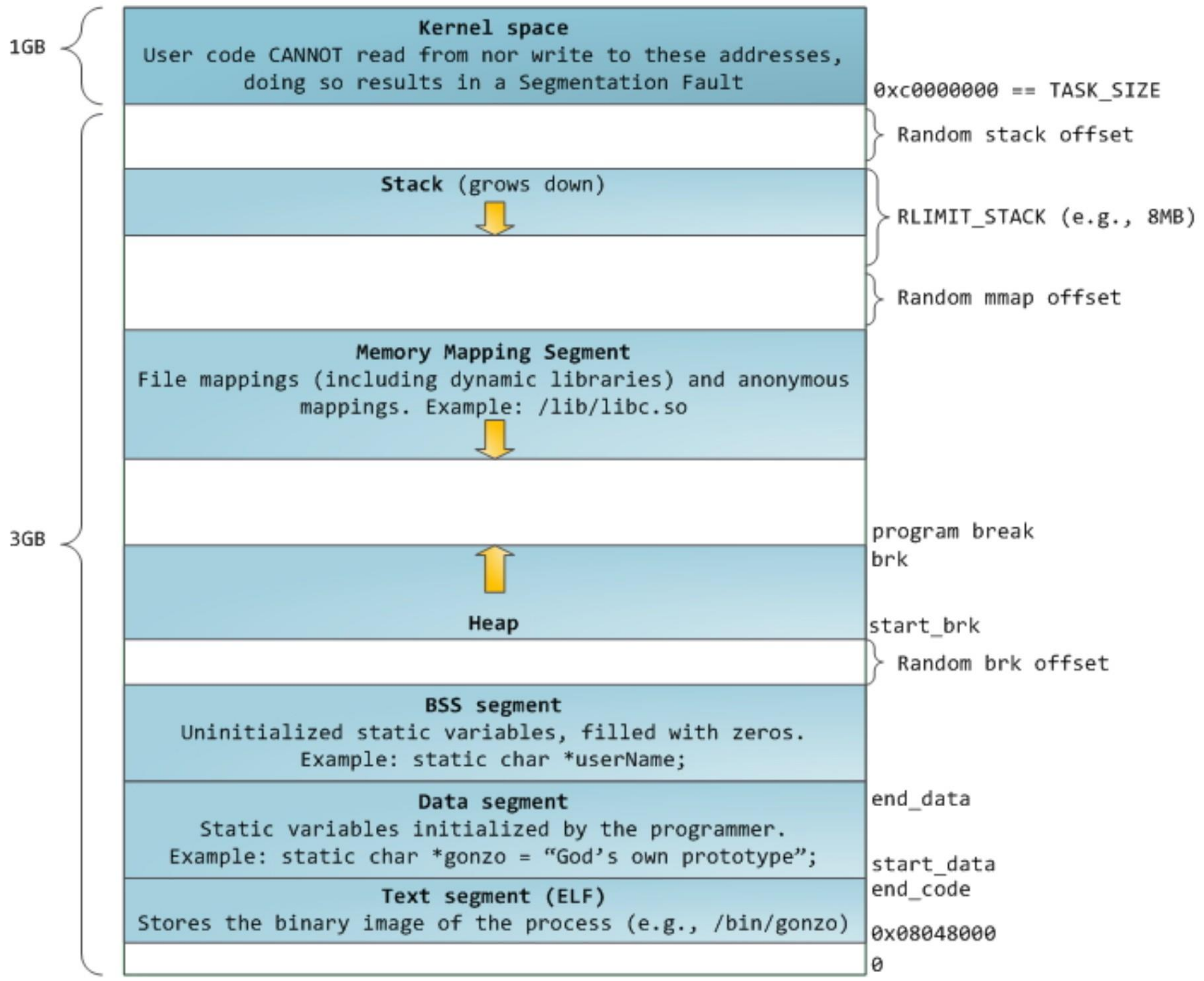
АКОС 4
Ассемблер и память. Начало

Стадии изучения ассемблера

1. Понимание
2. Read-only
3. Read-write

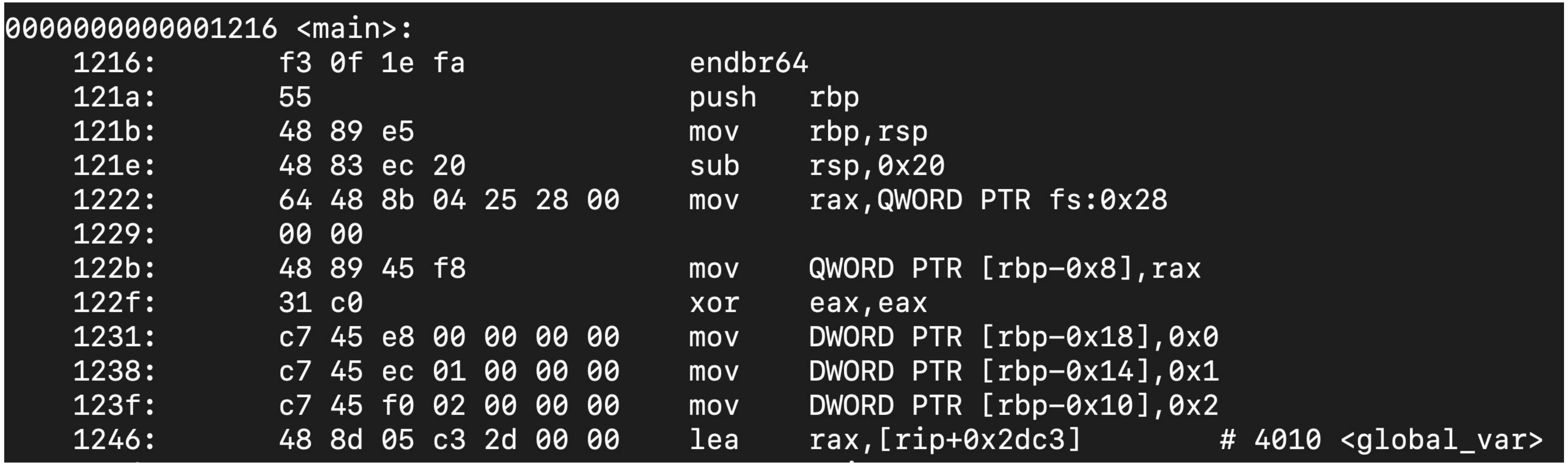
1. Отрицание
2. Гнев
3. Торг
4. Депрессия
5. Принятие

Организация памяти процесса


1GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
3GB
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros. Example: static char *userName;
end_data
Data segment
Static variables initialized by the programmer. Example: static char *gonzo = “God’s own prototype”;
start_data
end_code
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
0x08048000
0

Как посмотреть код

- objdump -d -M intel intel-mnemonic a.out



```
0000000000001216 <main>:
    1216:       f3 0f 1e fa             endbr64
    121a:       55                      push   rbp
    121b:       48 89 e5                mov    rbp,rsp
    121e:       48 83 ec 20             sub    rsp,0x20
    1222:       64 48 8b 04 25 28 00    mov    rax,QWORD PTR fs:0x28
    1229:       00 00
    122b:       48 89 45 f8             mov    QWORD PTR [rbp-0x8],rax
    122f:       31 c0                   xor    eax,eax
    1231:       c7 45 e8 00 00 00 00    mov    DWORD PTR [rbp-0x18],0x0
    1238:       c7 45 ec 01 00 00 00    mov    DWORD PTR [rbp-0x14],0x1
    123f:       c7 45 f0 02 00 00 00    mov    DWORD PTR [rbp-0x10],0x2
    1246:       48 8d 05 c3 2d 00 00    lea    rax,[rip+0x2dc3]        # 4010 <global_var>
```

Основные инструкции

Арифметические
add, sub, mul, div,

Логические
or, and, xor, inv

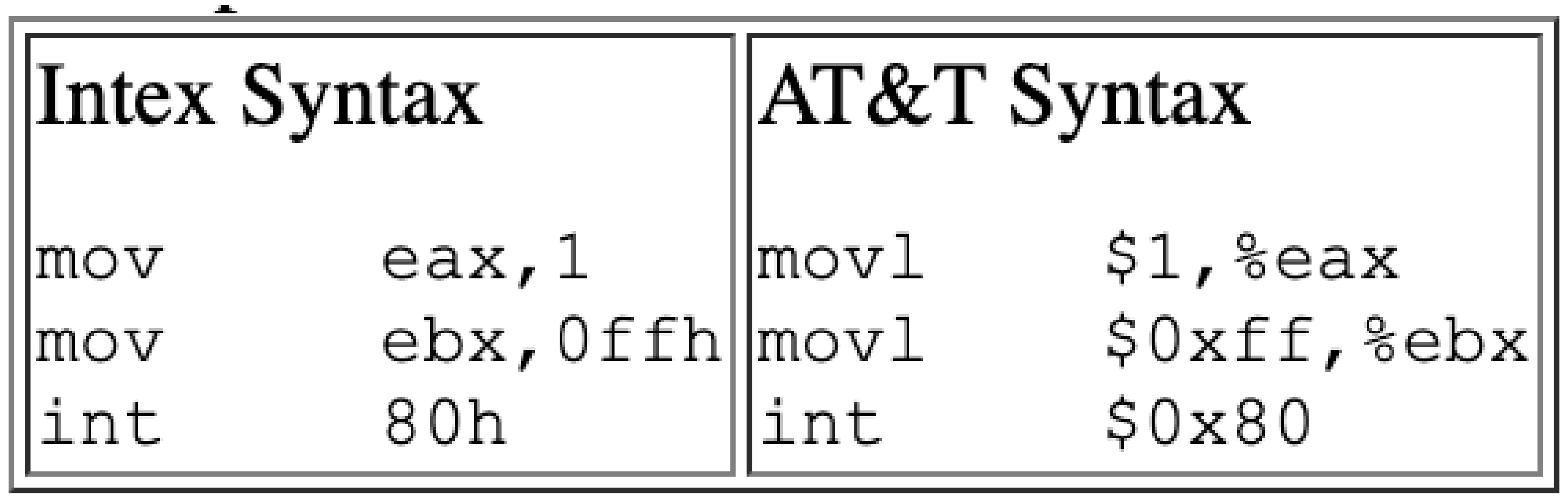
Регистры
mov

Прыжки
call, ret, jmp, je, ja, jle, …

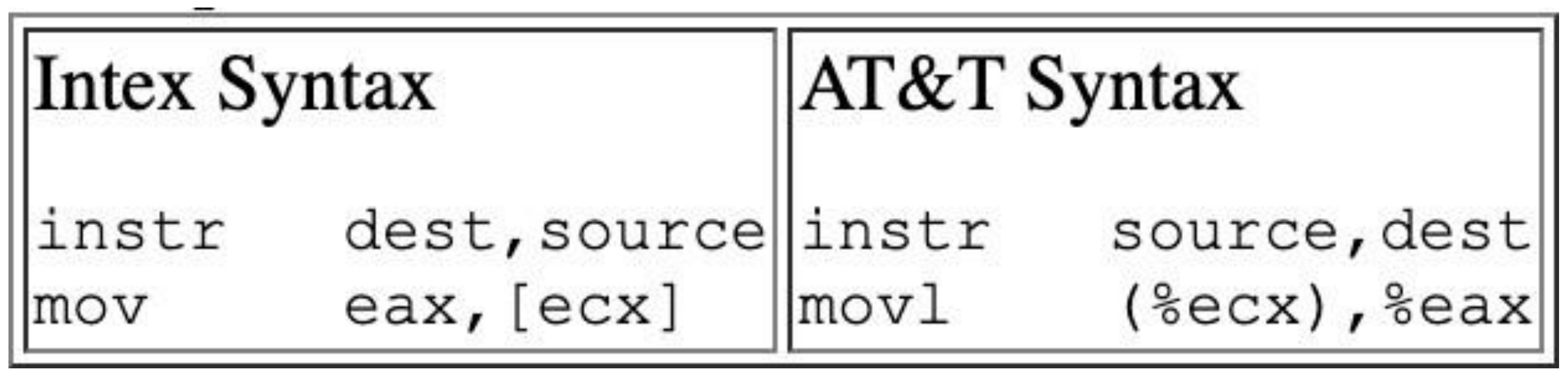
Стек
push, pop

AT&T vs Intel

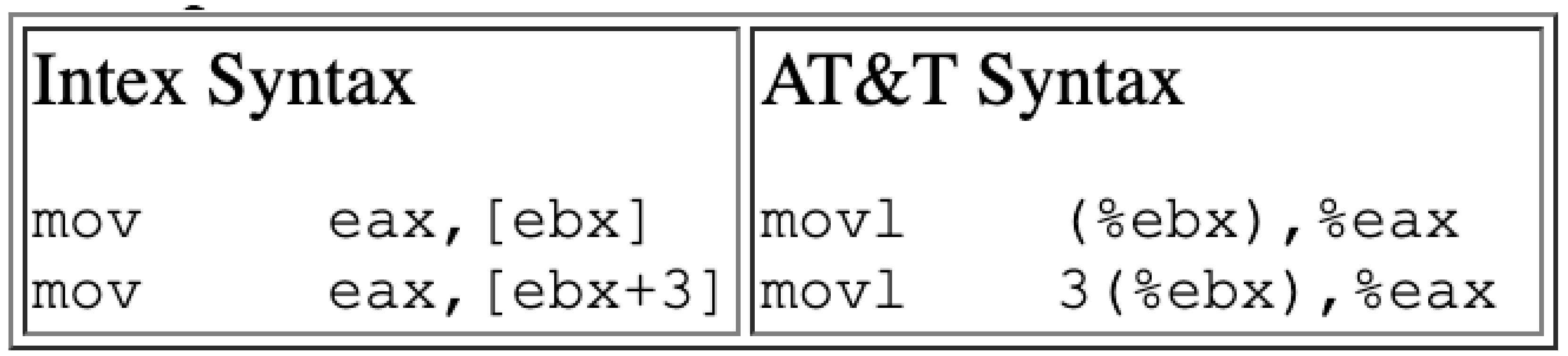
- objdump -M intel …
- gcc -masm=intel …
- gdb:
  - set disassembly-flavor intel
- Сравнение синтаксиса



Intex Syntax	AT&T Syntax
`mov      eax,1`	`movl     $1,%eax`
`mov      ebx,0ffh`	`movl     $0xff,%ebx`
`int      80h`	`int      $0x80`



Intex Syntax	AT&T Syntax
`instr    dest,source`	`instr    source,dest`
`mov      eax,[ecx]`	`movl     (%ecx),%eax`



Intex Syntax	AT&T Syntax
`mov      eax,[ebx]`	`movl     (%ebx),%eax`
`mov      eax,[ebx+3]`	`movl     3(%ebx),%eax`

Где почитать и попробовать

- godbolt.org
- objdump

Как писать на асме?

- Программа на асме стартер пак:
  - Объявление глобальной переменной _**start**
  - Определение метки _**start** (с неё начинается исполнение программы)
  - Завершение программы с помощью сискола exit.

Регистры

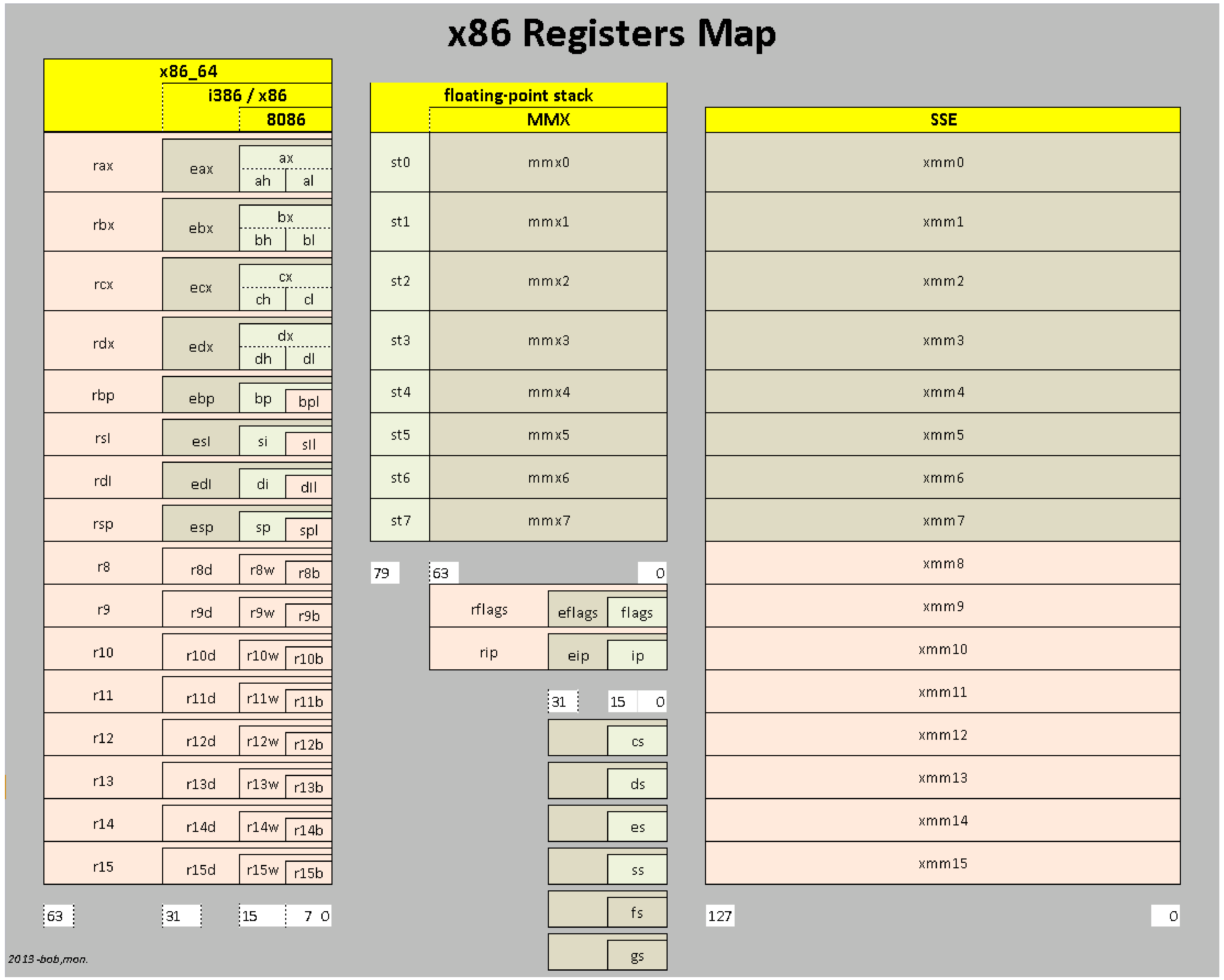
add *reg*, *reg*

add *reg*, *mem*

add *reg*, *imm*

add *mem*, *reg*

add *mem*, *imm*



Секции

- .text - код
- .bss - глобальные и статические переменные, которые проинициализированы нулём или никак
- .data - всё остальное
- .rodata - то же самое, но неизменяемое

Сравнения

```
    cmp al, n2        ; сраниваем значения регистра AL и переменной n2
    jne not_equal     ; если значения не равны, переходим к метке not_equal
    mov eax, 0
    jmp equal
not_equal:
    mov eax, 1
equal:
```

- cmp

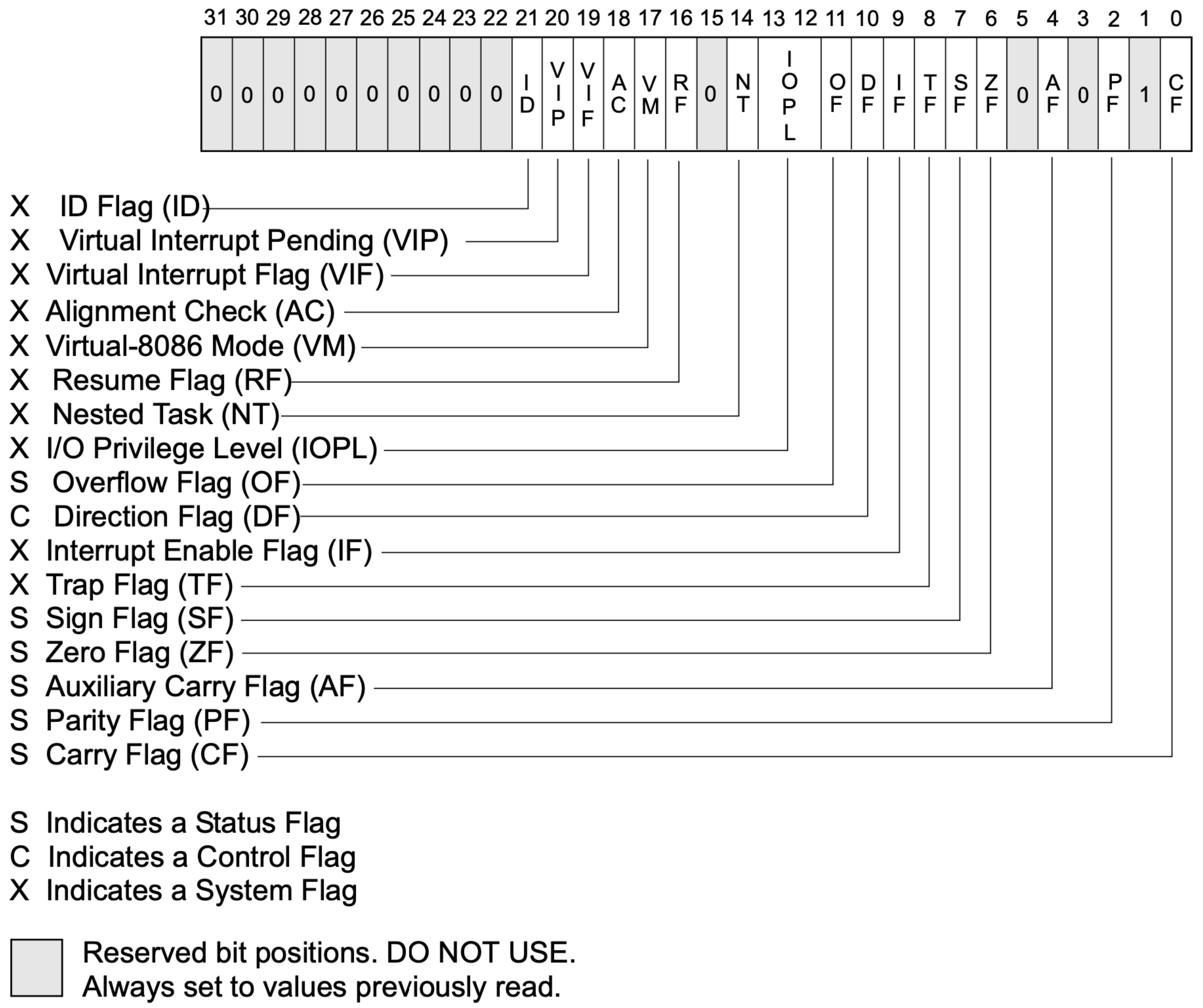
Регистр флагов EFLAGS

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.
Parity flag — Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.
Adjust 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.

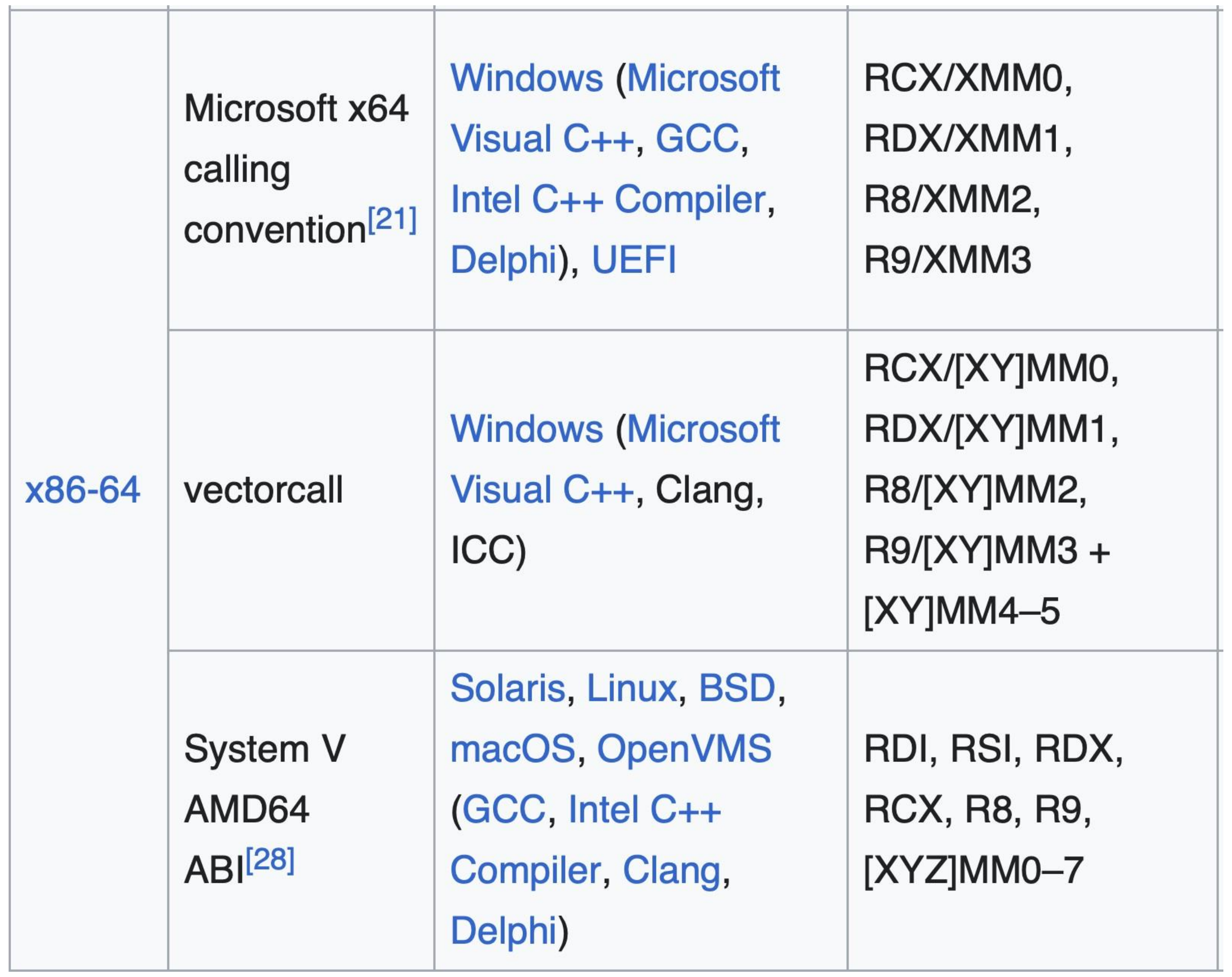
Zero flag — Set if the result is zero; cleared otherwise.
Sign flag — Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)

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.

- FLAGS-register

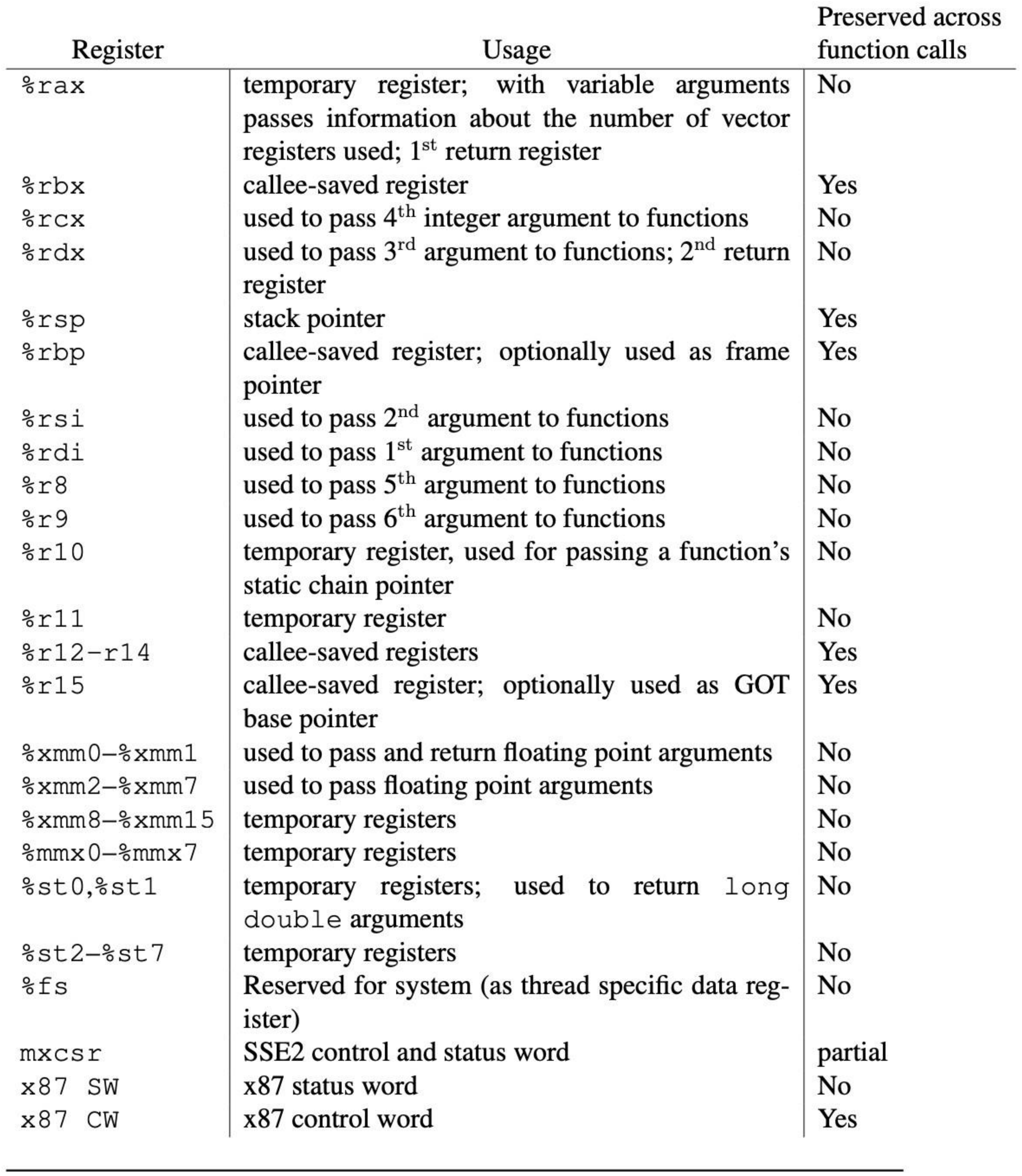

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE.
Always set to values previously read.

Calling conventions



x86-64	Microsoft x64 calling convention[21]	Windows (Microsoft Visual C++, GCC, Intel C++ Compiler, Delphi), UEFI	RCX/XMM0, RDX/XMM1, R8/XMM2, R9/XMM3
	vectorcall	Windows (Microsoft Visual C++, Clang, ICC)	RCX/[XY]MM0, RDX/[XY]MM1, R8/[XY]MM2, R9/[XY]MM3 + [XY]MM4–5
	System V AMD64 ABI[28]	Solaris, Linux, BSD, macOS, OpenVMS (GCC, Intel C++ Compiler, Clang, Delphi)	RDI, RSI, RDX, RCX, R8, R9, [XYZ]MM0–7

- Разные виды calling conventions
- SystemV ABI



Register	Usage	Preserved across function calls
`%rax`	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
`%rbx`	callee-saved register	Yes
`%rcx`	used to pass 4th integer argument to functions	No
`%rdx`	used to pass 3rd argument to functions; 2nd return register	No
`%rsp`	stack pointer	Yes
`%rbp`	callee-saved register; optionally used as frame pointer	Yes
`%rsi`	used to pass 2nd argument to functions	No
`%rdi`	used to pass 1st argument to functions	No
`%r8`	used to pass 5th argument to functions	No
`%r9`	used to pass 6th argument to functions	No
`%r10`	temporary register, used for passing a function's static chain pointer	No
`%r11`	temporary register	No
`%r12–r14`	callee-saved registers	Yes
`%r15`	callee-saved register; optionally used as GOT base pointer	Yes
`%xmm0–%xmm1`	used to pass and return floating point arguments	No
`%xmm2–%xmm7`	used to pass floating point arguments	No
`%xmm8–%xmm15`	temporary registers	No
`%mmx0–%mmx7`	temporary registers	No
`%st0,%st1`	temporary registers; used to return `long double` arguments	No
`%st2–%st7`	temporary registers	No
`%fs`	Reserved for system (as thread specific data register)	No
`mxcsr`	SSE2 control and status word	partial
`x87 SW`	x87 status word	No
`x87 CW`	x87 control word	Yes

gdb features

- i(nfo) r(egister) (rax)
- p $eflags
- x /nfu addr (z.B. x/3uh 0x54320)
- ~/.gdbinit настройки
- set disassembly-flavor intel

- Форматы чтения

Что посмотреть?

- Intel Architecture Manual vol.1
- Intel Architecture Manual
- nasm tutorial