

# Fundamentos de Engenharia Reversa

## Registradores

Os processadores possuem uma área física em seus *chips* para armazenamento de dados (de fato, somente números, já que é só isso que existe neste mundo!) chamadas de **registradores**, justamente porque registram (salvam) um número por um tempo, mesmo este sendo não determinado.

Os registradores possuem normalmente o tamanho da palavra do processador, logo, se este é um processador de 32-bits, seus registradores possuem este tamanho também.

## Registradores de Uso Geral

Um registrador de uso geral, também chamado de GPR (*General Purpose Register*) serve para armazenar temporariamente qualquer tipo de dado, para qualquer função.

Existem 8 registradores de uso geral na arquitetura Intel x86. Apesar de poderem ser utilizados para qualquer coisa, como seu nome sugere, a seguinte convenção é normalmente respeitada:

Registrador	Significado	Uso sugerido
EAX	Accumulator	Usado em operações aritméticas
EBX	Base	Ponteiro para dados
ECX	Counter	Contador em repetições
EDX	Data	Usado em operações de E/S
ESI	Source Index	Ponteiro para uma <i>string</i> de origem
EDI	Destination Index	Ponteiro para uma <i>string</i> de destino
ESP	Stack Pointer	Ponteiro para o topo da pilha
EBP	Base Pointer	Ponteiro para a base do <i>stack frame</i>

Para fixar o assunto, é importante trabalhar um pouco. Vamos escrever o seguinte programa em Assembly no Linux ou macOS:

```
section .text
    mov eax, 0x30
    or  eax, 0x18
```

Salve-o como `ou.s` e para compilar, vamos instalar o Netwide Assembler (NASM), que para o nosso caso é útil por ser multiplataforma:

```
$ sudo apt install nasm
$ nasm -felf ou.s
```

Confira como ficou o código **compilado** no arquivo objeto com a ferramenta **objdump**, do pacote binutils:


```
$ objdump -dM intel ou.o

ou.o:          file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 30 00 00 00      mov     eax,0x30
   5:   83 c8 18           or      eax,0x18
```

Perceba os *opcodes* e argumentos idênticos aos exemplificados na introdução deste capítulo.

 O NASM compilou corretamente a linguagem Assembly para código de máquina. O objdump mostrou tanto o código de máquina quanto o equivalente em Assembly, processo conhecido como **desmontagem** ou **disassembly**.

Agora cabe à você fazer mais alguns testes com outros registradores de uso geral. Um detalhe importante é que os primeiros quatro registradores de uso geral podem ter sua parte baixa manipulada diretamente. Por exemplo, é possível trabalhar com o registrador de 16 bits AX, a parte baixa de EAX. Já o AX pode ter tanto sua parte alta (AH) quanto sua parte baixa (AL) manipulada diretamente também, onde ambas possuem 8 bits de tamanho. O mesmo vale para EBX, ECX e EDX.

Para entender, analise as seguintes instruções:

```
mov eax, 0xaabbccdd
mov ax, 0xeeff
```

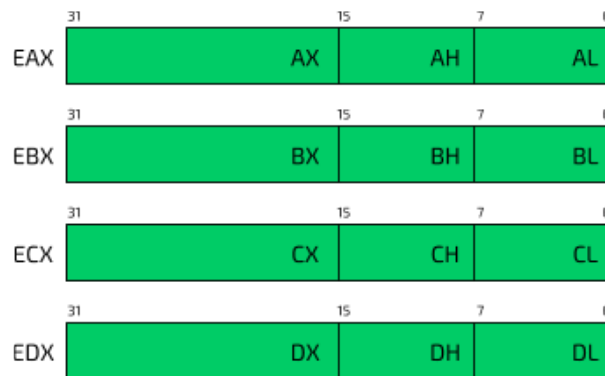
Após a execução das instruções acima, EAX conterá o valor **0xaabb~~ee~~ff**, já que somente sua parte **baixa** foi modificada pela segunda instrução. Agora analise o seguinte trecho:

```
mov eax, 0xaabbccdd
mov ax, 0xeeff
```

```
mov ah, 0xcc
mov al, 0xdd
```

Após a execução das quatro instruções acima, EAX volta para o valor **0xaabbccdd**.

A imagem a seguir ilustra os ditos vários registradores dentro dos quatro primeiros registradores de uso geral.



Sub-divisões de EAX, EBX, ECX e EDX

Os registradores EBP, ESI, EDI e ESP também podem ser utilizados como registradores de 16-bits BP, SI, DI e SP, respectivamente. Note porém que estes últimos não são sub-divididos em partes alta (*high*) e baixa (*low*).

## Ponteiro de Instrução

Existe um registrador chamado de EIP (*Extended Instruction Pointer*), também de PC (*Program Counter*) em algumas literaturas que aponta para a próxima instrução a ser executada. Não é possível copiar um valor literal para este registrador. Portanto, uma instrução `mov eip, 0x401000` **não** é válida.

Outra propriedade importante deste registrador é que ele é incrementado com o número de *bytes* da última instrução executada. Para fixar, analise o exemplo do *disassembly* a seguir, gerado com a ferramenta **objdump**:

```
08049000 <_start>:
8049000:  b8 01 00 00 00      mov     eax,0x1
8049005:  bb 00 00 00 00      mov     ebx,0x0
804900a:  cd 80               int     0x80
804900c:  b8 02 00 00 00      mov     eax,0x2
```

Quando a primeira instrução do trecho acima estiver prestes à ser executada, o registrador EIP conterá o valor 0x8049000. Após a execução desta primeira instrução MOV, o EIP será incrementado em 5 unidades, já que tal instrução possui 5 *bytes*. Por isso o **objdump** já mostra o endereço correto da instrução seguinte. Perceba, no entanto, que a instrução no endereço 804900a possui apenas 2 *bytes*, o que vai fazer com o que o registrador EIP seja incrementado em 2 unidades para apontar para a próxima instrução MOV, no endereço 804900c.

## Registradores de Segmento


Estes registradores armazenam o que chamamos de seletores, ponteiros que identificam segmentos na memória, essenciais para operação em modo real. Em modo protegido, que é o modo de operação do processador que os sistemas operacionais modernos utilizam, a função de cada registrador de segmento fica a cargo do SO. Abaixo a lista dos registradores de segmento e sua função em modo protegido:

Registrador	Significado
CS	Code Segment
DS	Data Segment
SS	Stack Segment
ES	Extra Data Segment
FS	Data Segment (no Windows em x86, aponta para TEB ( <i>Thread Environment Block</i> ) da <i>thread</i> atual do processo em execução)
GS	Data Segment

Não entraremos em mais detalhes sobre estes registradores por fugirem do escopo deste livro.

## Registrador de Flags

O registrador de *flags* EFLAGS é um registrador de 32-bits usado para *flags* de estado, de controle e de sistema.

 *Flag* é um termo genérico para um dado, normalmente "verdadeiro ou falso". Dizemos que uma *flag* está *setada* quando seu valor é verdadeiro, ou seja, é igual a 1.

Existem 10 *flags* de sistema, uma de controle e 6 de estado. As *flags* de estado são utilizadas pelo processador para reportar o estado da última operação (pense numa comparação, por exemplo - você pede para o processador comparar dois valores e a resposta vem através de uma *flag* de estado). As mais comuns são:

Bit	Nome	Sigla	Descrição
0	Carry	CF	Setada quando o resultado estourou o limite do tamanho do dado. É o famoso "va

			um" na matemática para números sem sinal ( <i>unsigned</i> ).
6	Zero	ZF	Setada quando o resultado de uma operação é zero. Do contrário, é zerada. Muito usada em comparações.
7	Sign	SF	Setada de acordo com o MSB ( <i>Most Significant Bit</i> ) do resultado, que é justamente o <i>bit</i> que define se um inteiro com sinal é positivo (0) ou negativo (1), conforme visto na seção Números negativos.
11	Overflow	OF	Estouro para números com sinal.

Além das outras *flags*, há ainda os registradores da FPU (*Float Point Unit*), de debug, de controle, XMM (parte da extensão SSE), MMX, 3DNow!, MSR (*Model-Specific Registers*), e possivelmente outros que não abordaremos neste livro em prol da brevidade.

Agora que já reunimos bastante informação sobre os registradores, é hora de treinarmos um pouco com as instruções básicas do Assembly.