Instruções Lógicas
Memória
Pilha

1

Instruções Lógicas

- "Um deslocamento lógico move os bits um determinado número de posições para a direita ou esquerda.
- As posições que não são preenchidas pela operação de deslocamento são preenchidas com um bit zero (0).
- Um deslocamento aritmético faz o mesmo, exceto que o bit de sinal é sempre retida.
- **Essa variação permite que uma operação de deslocamento forneça um mecanismo rápido para multiplicar ou dividir os números do complemento de 2 por 2."**

2

instruções lógicas

SHR *operando, valor*

desloca o *operando* para a direita por bits *de valor* .

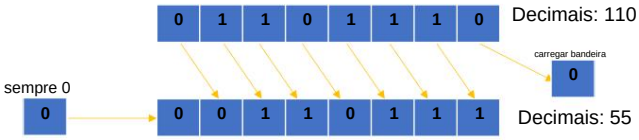
operando pode ser memória ou registrador (8, 16 ou 32 bits)

valor pode ser um valor numérico ou registro CL

coloque 0 no bit esquerdo

divide o *operando* sem sinal por 2valor (divisão muito rápida)

Exemplo
MOV AL,01101110B
SHR AL,1



3

instruções lógicas

Operando SHL , *valor*

shift deixou o *operando* por bits *de valor* .

operando pode ser memória ou registrador (8, 16 ou 32 bits)

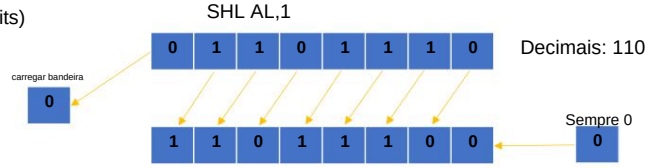
valor pode ser um valor numérico ou registro CL

coloque 0 no bit certo

multiplique o *operando* por 2valor (multiplicação muito rápida)

Decimais: 220
2's complementares:
011011100 (9 bits)

Exemplo
MOV AL,01101110B
SHL AL,1



4

instruções lógicas

Operando SAR , valor (deslocar aritmética para a direita)

desloca o *operando* para a direita por bits *de valor* .

operando pode ser memória ou registrador (8, 16 ou 32 bits)

valor pode ser um valor numérico ou registro CL

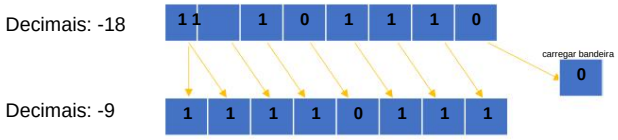
o bit esquerdo permanece o mesmo

divide o *operando* com sinal por 2valor

Nota: o resultado para valores negativos pode ser diferente do resultado usando idiv

-9/4=-2 usando IDIV e -3 usando SAR -> arredondamento para infinito negativo.

MOV AL,11101110B
SARAL,1



5

instruções lógicas

Operando SAL , valor (deslocar aritmética para a esquerda)

shift deixou o *operando* por bits *de valor* .

operando pode ser memória ou registrador (8, 16 ou 32 bits)

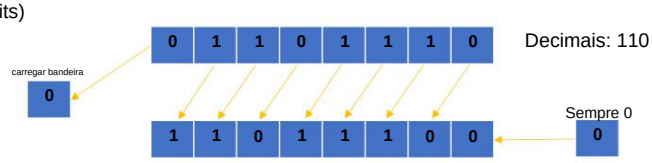
valor pode ser um valor numérico ou registro CL

coloque 0 no bit certo

multiplique o *operando* por 2valor (multiplicação muito rápida)

Exemplo
MOV AL 01101110B
SALAL,1

Decimais: 220
2's complementares:
011011100 (9 bits)



6

instruções lógicas

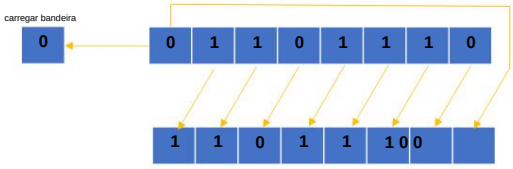
Operando ROL , valor (girar para a esquerda)

girar à esquerda do *operando* por bits *de valor* .

operando pode ser memória ou registrador (8, 16 ou 32 bits)

valor pode ser um valor numérico ou registro CL

Exemplo
MOV AL, 01101110B
ROLAL,1



7

instruções lógicas

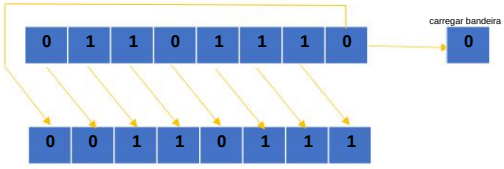
Operando ROR , valor (girar para a direita)

gire o *operando* para a direita por bits *de valor* .

operando pode ser memória ou registrador (8, 16 ou 32 bits)

valor pode ser um valor numérico ou registro CL

Exemplo
MOV AL, 01101110B
RORAL,1



8

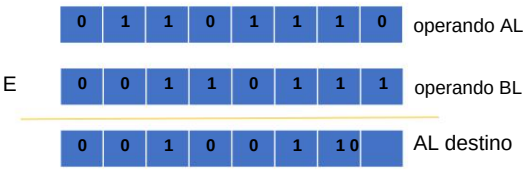
instruções lógicas

E *destino, origem*

executar um pouco a pouco E operação

destino e *fonte* têm as mesmas regras que mov

Exemplo
MOV AL,01101110B
MOV BL,00110111B
E AL,BL



9

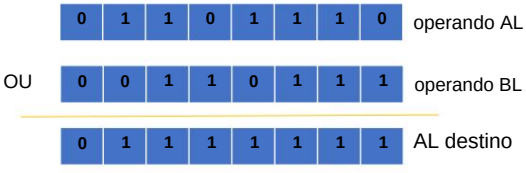
instruções lógicas

OU *destino, fonte*

executar uma operação OU bit a bit

destino e *fonte* têm as mesmas regras que mov

Exemplo
MOV AL,01101110B
MOV BL,00110111B
OU AL,BL



10

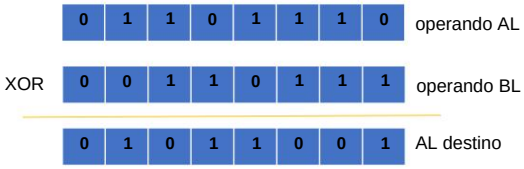
instruções lógicas

XOR *destino, fonte*

executar uma operação XOR bit a bit

destino e *fonte* têm as mesmas regras que mov

Exemplo
MOV AL,01101110B
MOV BL,00110111B
XOR AL, BL



11

Unidade de geração de endereços

- Parte especializada da unidade de execução
- Operação básica: calcula um endereço com base nos sinais de controle da unidade de controle e possivelmente no conteúdo adicional dos registradores
  - por exemplo, ponteiro base + índice = endereço

- Pode ser muito simples, mas também muito complexo • MMU (memory management unit)
  - muitos modos diferentes, proteção de memória, espaço de endereço virtual
  - otimização de cache, previsão de ramificação, carregamento especulativo, etc. • coberto mais tarde!

12

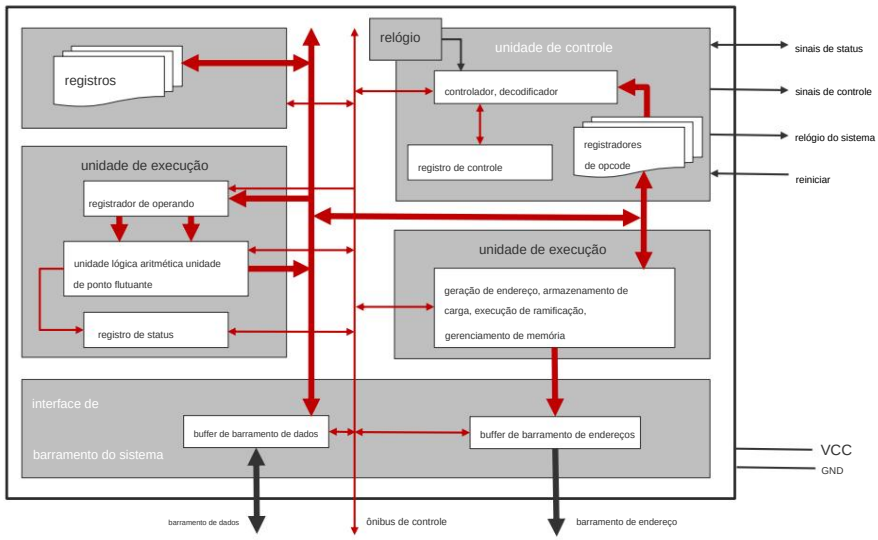
Interface de barramento do sistema

- A interface de barramento do sistema (Bus Interface Unit, BIU) é a conexão do microprocessador ao seu ambiente (todos os outros componentes de um microcomputador)

•Propósito

- Buffering de endereços e dados (operandos e instruções) •
  Adaptação de ciclos de clock, largura do barramento,
  tensões • Tristate: desconectando o processador do barramento externo

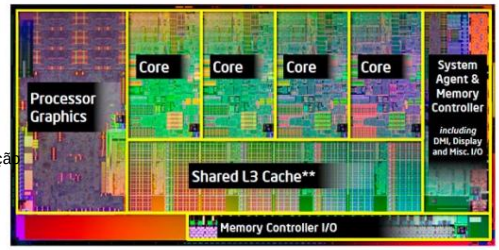
13



14

Componentes adicionais de um microprocessador

- Memória cache (memória rápida para instruções e operandos, abordada posteriormente) • Unidade de processamento vetorial • Processador gráfico • Unidade de processamento de sinal • Redes neurais, suporte AI • Controlador de interrupção



15

instruções de endereço

registrador LEA , variável

Load Effective Address coloca o endereço da *variável* no *registrador*

Exemplo

LEA EDI,lista; coloca no registrador EDI o endereço da variável lista

Após esta linha as seguintes instruções são equivalentes

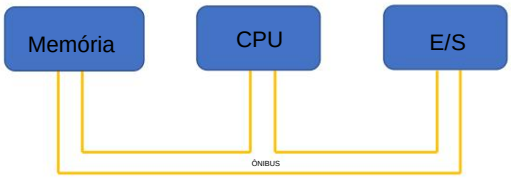
MOV lista[0],AL;

MOV [EDI], AL;

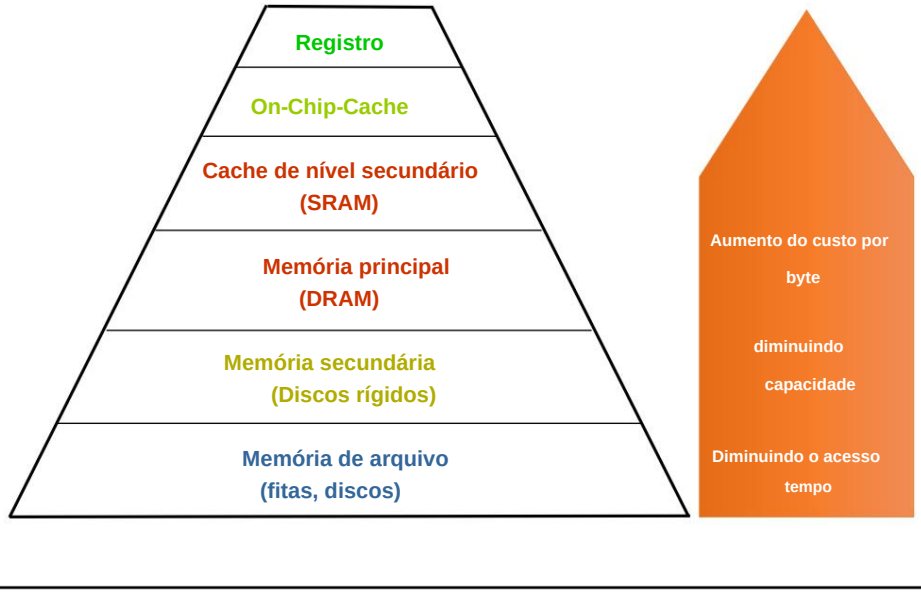
Nota: EDI – Índice de Destino Estendido

16

Esquema básico de computador



17



18

organização da memória

- A memória é organizada em bytes
- Binário é sempre o sistema! • Cada endereço como apenas um byte
- Cada byte como apenas um endereço

- Podemos fazer apenas duas operações na memória
  - Ler
  - Escrever

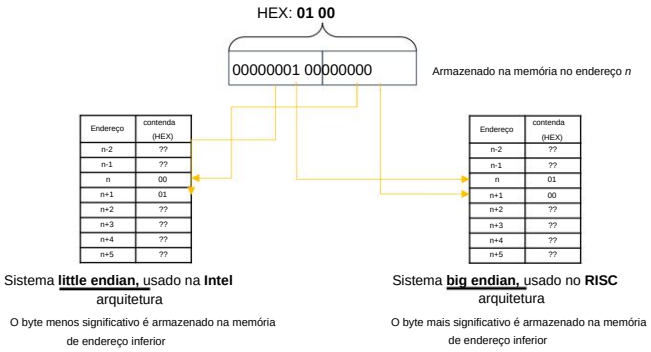
Endereço (binário)	contenda (HEX)
000	00
001	15
010	34
011	87
100	FA
101	FF
110	03
111	1F

19

organização da memória

Como representar dados maiores que 1 byte

Exemplo: inteiro curto (2 bytes), com valor 256d



Sistema **little endian,** usado na **Intel** arquitetura

O byte menos significativo é armazenado na memória de endereço inferior

Sistema **big endian,** usado no **RISC** arquitetura

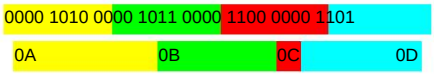
O byte mais significativo é armazenado na memória de endereço inferior

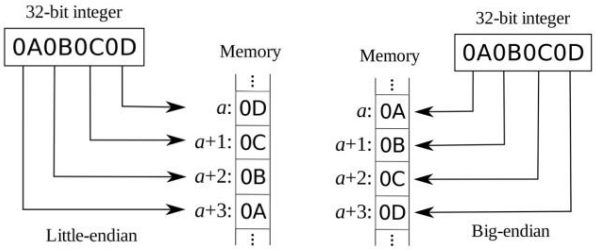
20

organização da memória

exercícios

Represente os 4 bytes a seguir no sistema little- e big-endian no endereço *a*





21

Operações de gravação e leitura de memória

Uma única instrução em uma **CPU de 64 bits** pode escrever ou ler 1, 2, 4 ou 8 bytes

Instruções especiais em uma CPU de 64 bits também podem escrever ou ler 16, 32 ou 64 bytes (128, 256 ou 512 bits). **Detalhes mais tarde.**

O número de bytes a serem escritos ou lidos é definido pelo tamanho do operando.

- tamanho 1 gravação\leitura 1byte
- tamanho 2 gravação\leitura 2bytes
- tamanho 4 gravação\leitura 4bytes
- tamanho 8 gravação\leitura 8bytes

Não é possível em uma única instrução escrever ou ler 3, 5, 6 e 7 bytes

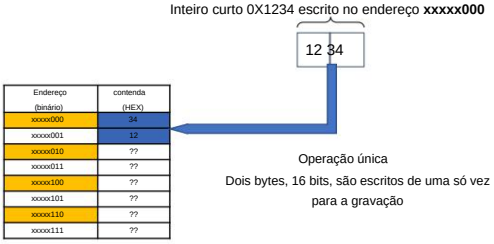
22

Alinhamento de memória

As operações de leitura e escrita são mais rápidas se a memória estiver alinhada com o tamanho do operando.
Alguns sistemas devem usar sempre memória alinhada

Se o **tamanho do operando for 2** e a **memória de endereços for múltipla de 2**

Inteiro curto 0X1234 escrito no endereço **xxxxx000**



Operação única
Dois bytes, 16 bits, são escritos de uma só vez para a gravação

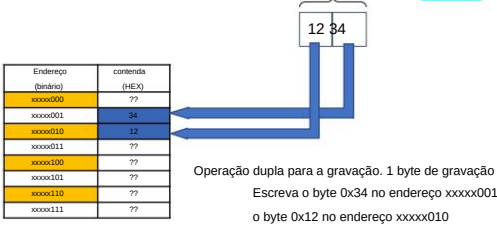
23

Alinhamento de memória

As operações de leitura e escrita são mais lentas se a memória não estiver alinhada com o tamanho do operando.

Se o **tamanho do operando for 2** e **a memória de endereço não for múltipla de 2**

Inteiro curto 0X1234 escrito no endereço xxxxx001



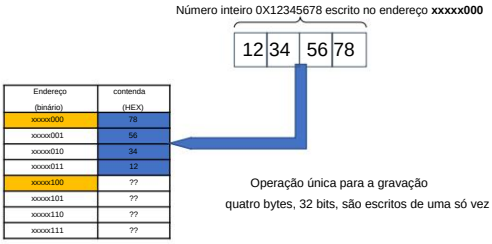
Operação dupla para a gravação. 1 byte de gravação a cada vez
Escreva o byte 0x34 no endereço xxxxx001 Escreva o byte 0x12 no endereço xxxxx010

24

Alinhamento de memória

As operações de leitura e escrita são mais rápidas se a memória estiver alinhada com o tamanho do operando.

Se o **tamanho do operando for 4** e a **memória de endereços for múltipla de 4**



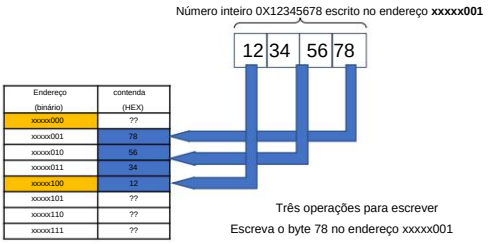
Operação única para a gravação
quatro bytes, 32 bits, são escritos de uma só vez

25

Alinhamento de memória

As operações de leitura e escrita são mais lentas se a memória não estiver alinhada com o tamanho do operando.

Se o tamanho do operando for **4** e **a memória de endereço não for múltipla de 4**

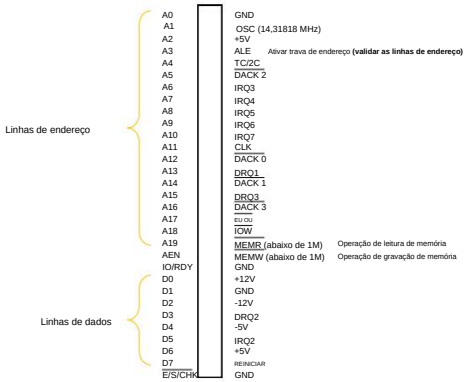


Três operações para escrever
Escreva o byte 78 no endereço xxxxx001
Escreva os bytes 3456 no endereço xxxxx010
Escreva o byte 12 no endereço xxxxx100

26

ISA BUS (8 bits)

barramento muito simples usado desde a CPU 8086



27

Ciclo de gravação

(simplifique)

1. A CPU coloca o endereço nas linhas de endereço
2. A CPU valida o endereço através do sinal ALE 3.

A memória decodifica as linhas de endereço

4. A CPU coloca os dados nas linhas de

dados 5. A CPU ativa o sinal MEMW 6.

A memória salva o conteúdo das linhas de dados na posição do endereço

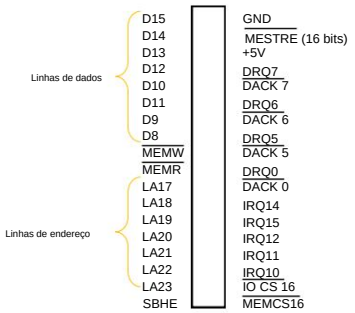
28

Ciclo de leitura
(simplifique)

1. A CPU coloca o endereço nas linhas de
endereço 2. A CPU valida o endereço através do
sinal ALE 3. A memória decodifica as
linhas de endereço 4. A CPU ativa o
sinal MEMR 5. A memória coloca nas linhas de dados os dados do
posição de endereço 6. A CPU lê os dados das linhas de dados

29

Barramento ISA estendido (16 bits)



30

Pilha

Pilha é uma zona de memória especial usada para armazenar dados

Variáveis e parâmetros locais são armazenados na pilha

Cada chamada de função cria um novo espaço para sua própria pilha

Isso possibilita funções recursivas, pois as variáveis locais e os parâmetros para cada chamada de função são exclusivos

31

Pilha

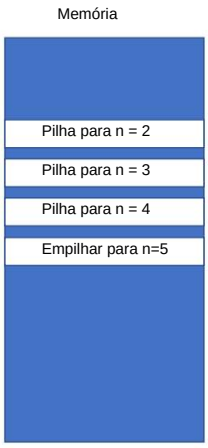
```
unsigned int fatorial(unsigned int n) {

    if (n <= 2)
        return(n);
    return(fatorial(n - 1)*n);

} int main(int argc, char* argv[]) {

    printf("5 fatorial é %d", fatorial(5));
}
```

```
5 factorial is 120
```



32

Pilha

A pilha é organizada como um LIFO (Last In, First Out)

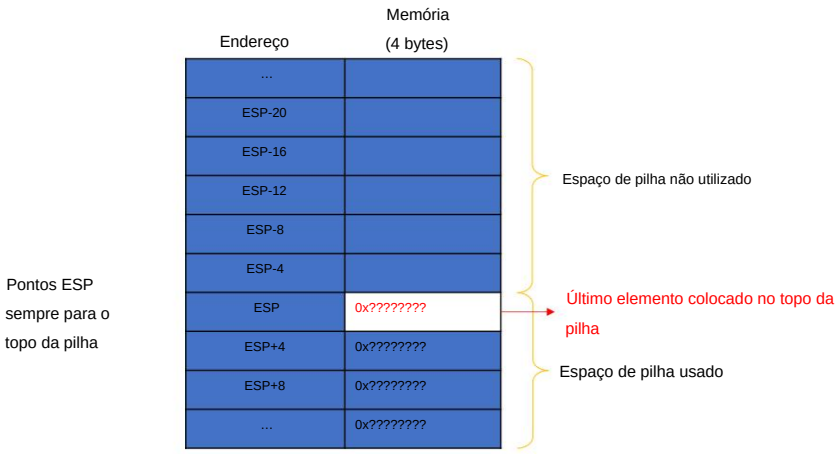
O registrador ESP (Stack Pointer) aponta, a qualquer momento, para o último elemento colocado na pilha (stack top)

A pilha cresce de baixo (endereço alto) para cima (endereço baixo)

A pilha é organizada em 4 bytes por elemento (exceção posterior…)

33

Pilha



34

Instruções de empilhamento

***fonte* PUSH**

Coloque na pilha a fonte

A fonte pode ser um registrador, memória ou valor

Se o tamanho da fonte for de 16 bits (registro ou memória), a pilha usará apenas 2 bytes

Em outras situações, tamanho de 8 bits ou tamanho de 32 bits, a pilha usa 4 bytes

Usando valores como *fonte ,* a pilha usa sempre 4 bytes

35

Instruções de empilhamento

fonte PUSH

Equivalente para tamanho de fonte de 8 ou 32 bits

SUB ESP,4

MOV [ESP], fonte

Equivalente para tamanho de fonte de 16 bits

SUB ESP,2

MOV [ESP],fonte

36

Instruções de empilhamento

MOV EAX,12345678H

PUSH EAX

Empilhar antes de empurrar (ESP=XXXX18)

	xxxx00	
	xxxx04	
	xxxx08	
	xxxx0C	
	xxxx10	
	xxxx14	
ESP	xxxx18	0x????????
	xxxx1C	0x????????
	xxxx20	0x????????
	xxxx24	0x????????

Pilha após push (ESP=XXXX14)

	xxxx00	
	xxxx04	
	xxxx08	
	xxxx0C	
	xxxx10	
ESP	xxxx14	0x12345678
	xxxx18	0x????????
	xxxx1C	0x????????
	xxxx20	0x????????
	xxxx24	0x????????

37

Instruções de empilhamento

MOV AX,1234H

EMPURRAR MACHADO; (tamanho de 16 bits. Use apenas 2 bytes na pilha)

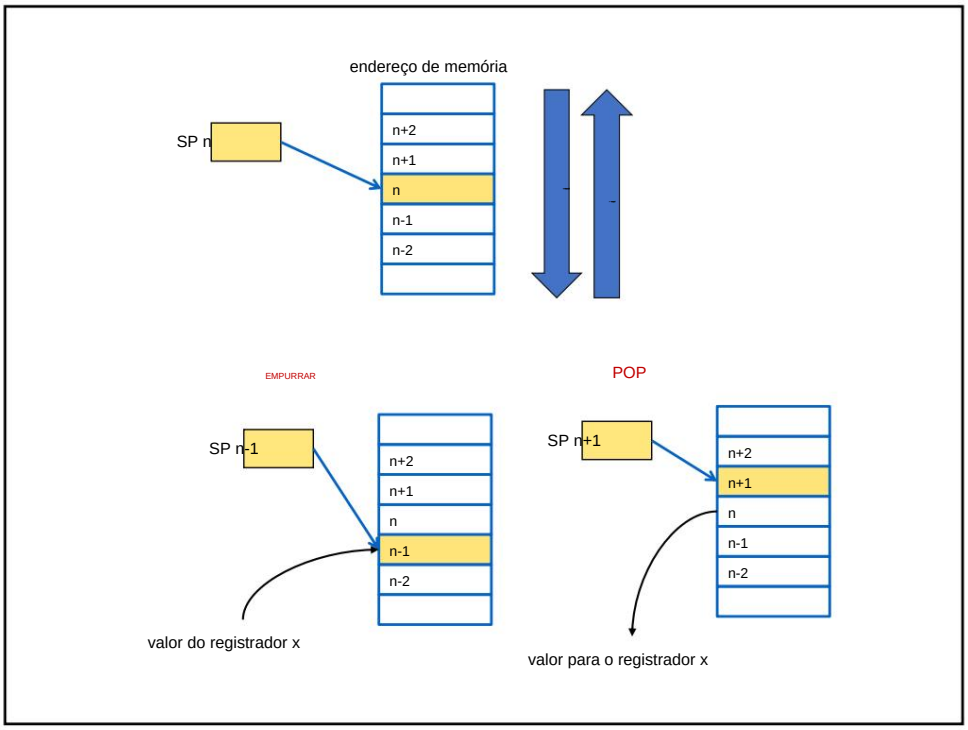
Empilhar antes de empurrar (ESP=XXXX18)

	xxxx00	
	xxxx04	
	xxxx08	
	xxxx0C	
	xxxx10	
	xxxx14	
ESP	xxxx18	0x????????
	xxxx1C	0x????????
	xxxx20	0x????????
	xxxx24	0x????????

Pilha após push (ESP=XXXX16)

	xxxx02		
	xxxx06		
	xxxx0A		
	xxxx0E		
	xxxx12		
ESP	xxxx16	0x1234	2 bytes
	xxxx18	0x????????	
	xxxx1C	0x????????	
	xxxx20	0x????????	
	xxxx24	0x????????	

38



39

Instruções de empilhamento

Destino POP

Pegue da pilha o último elemento e coloque-o no destino

O destino pode ser um registrador ou memória

Se o tamanho do destino for 16 bits, apenas 2 bytes serão retirados da pilha

Em outras situações, tamanho de 8 bits ou tamanho de 32 bits, 4 bytes são retirados da pilha

40

Instruções de empilhamento

Destino POP

Equivalente para tamanho de destino de 8 ou 32 bits

Destino MOV,[ESP]

ADICIONE ESP,4

Equivalente para tamanho de destino de 16 bits

Destino MOV,[ESP]

ADICIONAR ESP,2

41

Instruções de empilhamento

POP EBX

Empilhar antes de estourar (ESP=XXXX14)

	xxxx00	
	xxxx04	
	xxxx08	
	xxxx0C	
	xxxx10	
ESP	xxxx14	0x12345678
	xxxx18	0x????????
	xxxx1C	0x????????
	xxxx20	0x????????
	xxxx24	0x????????

Pilha após pop (ESP=XXXX18)

	xxxx00	
	xxxx04	
	xxxx08	
	xxxx0C	
	xxxx10	
	xxxx14	0x12345678
ESP	xxxx18	0x????????
	xxxx1C	0x????????
	xxxx20	0x????????
	xxxx24	0x????????

42

Instruções de empilhamento

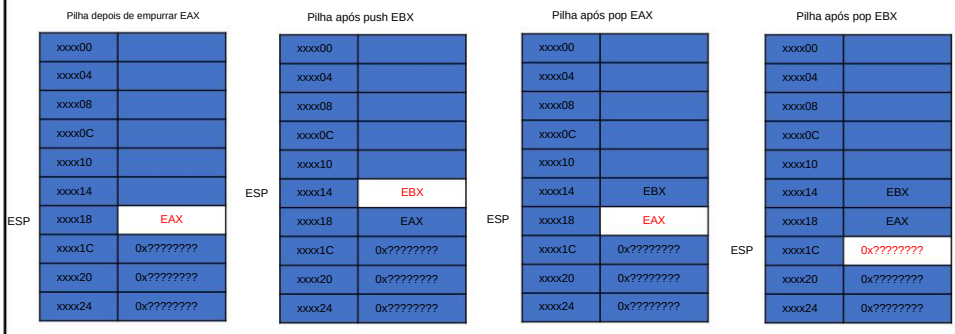
POPBX; (tamanho de 16 bits. Use apenas 2 bytes na pilha)

43

Instruções de empilhamento

PUSH EAX
PUSH EBX
POP EAX
POP EBX

Descubra o que este código faz



44

Acesso aos dados da pilha

A instrução MOV pode ser usada para acessar dados na pilha como qualquer outro dado na memória

Exemplos

MOV EAX,[ESP]; coloque 0x77777777 em EAX

MOV EAX,[ESP+4]; coloque 0x88888888 em EAX

MOV EAX,[ESP-4]; coloque 0x66666666 em EAX

45