Praticas 05

O objetivo do programa eh testar as instruções básicas a seguir:

movl/movw/movb	: move dados de 32/16/8 bits
pushl/pushw	: empilha dados de 32/16 bits
popl/popw	: desempilha dados de 32/16 bits
pusha/popa	: empilha/desempilha todos os regs de 16 bits
pushad/popad	: empilha/desempilha todos os regs de 32 bits
sall/salw/salb	: desloca bits a esquerda, sobre dados de 32/16 bits, retirando-os do extremo esquerdo e inserindo 0's no extremo direito
sarl/sarw/sarb	: desloca bits a direita, sobre dados de 32/16 bits, retirando-os do extremo direito e inserindo 0's no extremo esquerdo
rorl/rorw/rorb	: rotaciona bits a direita, sobre dados de 32/16 bits, retirando-os do extremo direito e inserindo-os de volta no extremo esquerdo
roll/rolw/rolb	: rotaciona bits a esquerda, sobre dados de 32/16 bits, retirando-os do extremo esquerdo e inserindo-os de volta no extremo direito
xchgl/xchgw/xchgb	: troca conteúdos dos registradores entre si

ATENÇÃO: Os sufixos "l, w ou b" nas instruções indicam que a intrução manipula dados longos, word ou byte, isto é, de 32, 16 ou 8 bits, respectivamente. A ausência desse sufixo também é aceita e indica o maior caso existente na arquitetura.

Os registradores manipulados pelas instruções devem ser compativeis com esse sufixo. Assim, as instruções "l" devem utilizar registradores de 32 bits, por exemplo, "%eax", instruções "w" devem utilizar registradores de 16 bits, por exemplo, "ax", e instruções "b" devem utilizar registradores de 8 bits, por exemplo, "ah" ou "al".

Saiba que para garantir a compatibilidade dos códigos mais antigos nas arquiteturas mais recentes, os registradores recentes são projetados como sendo compostos pelos registradores mais antigos. Assim, os registradores de 32 bits são formados de 2 registradores de 16 bits, que por sua vez são formados por 2 registradores de 8 bits. Por exemplo, o registrador de 32 bits %eax é formado por um registrador de 16 bits na sua metade da esquerda, inacessível diretamente, e por um registrador de 16 bits na sua metade da direita, o %ax. O %ax por sua vez é formado por um registrador de 8 bits na sua metade da esquerda, o %ah, e por um registrador de 8 bits na sua metade da direita, o %al. Da mesma forma ocorre com os registradores %ebx, %ecx, %edx e outros.

Para gerar o executável, gere primeiro o objeto executando o seguinte comando:

```
as praticas_05a.s -o praticas_05a.o
```

e depois link dinamicamente com o seguinte comando:

```
ld praticas_05a.o -l c -dynamic-linker /lib/ld-linux.so.2 -o praticas_05a
```

==========

Codificação: Monte e teste o código, passo a passo, concatenando os trechos a seguir

==========

```
.section .data

saida:
      .asciz      "Teste %d: O valor do registrador eh: %X\n\n"

saida2:
      .asciz      "Teste %d: Os valores dos regs sao: %X e %X\n\n"

saida3:
      .asciz      "Teste %d: EAX = %X ; EBX = ; %X ECX = %X ; EDX = %X ;
ESI = %X ; EDI = %X\n\n"

.section .text

.globl     _start

_start:

1) movendo, empilhando e imprimindo 32/16/08 bits

      movl  $0x12345678, %eax
      pushl %eax
      pushl $1
      pushl $saida
      call  printf

      movw  $0x1234, %ax
      pushw %ax
      movw  $0xABCD, %ax
      pushw %ax
      pushl $1
      pushl $saida
      call  printf

      movb  $0xAA, %ah
      movb  $0xBB, %al
      pushw %ax
      movb  $0xCC, %ah
      movb  $0xDD, %al
      pushw %ax
      pushl $1
      pushl $saida
      call  printf
```

Coloque as instruções de finalização de programa para poder testar o programa até aqui. São elas:

```
      pushl $0
      call exit
```

Agora, insira no programa antes das instruções de finalização do programa, um a um, os trechos de códigos a seguir, de 2 a 12, mantendo os trechos anteriores já inseridos. Para cada trecho inserido, monte, link e execute o programa para observar os resultados. Depois insira o próximo.

Resumo: Genericamente, instrução mov possui o seguinte formato:

"***movx*** *fonte, destino*" # destino ← fonte

onde x = l, w ou b, dependendo dos operandos serem de 32, 16 ou 8 bits; o operando *fonte* pode ser dado imediato (constante ou endereço de variável → $), memória (variável ou registrador entre parenteses) ou registrador; o operando *destino* pode ser memória (variável ou registrador entre parênteses) ou registrador; os operandos *fonte* e *destino* não podem ser simultaneamente memória.

Resumo: Genericamente, instrução push possui o seguinte formato:

"**push*x*** *fonte*" # pilha (%esp) ← fonte

onde x = l ou w (não sendo permitido b), dependendo do operando ser de 32 ou 16 bits; o operando *fonte* pode ser dado imediato (constante ou endereço de variável → $), variável ou registrador.

2) rotacionando regs de 32/16/8 bits 16/8/4 e 8/4/4 bits a esquerda

```
movl  $0x12345678, %eax
roll  $16, %eax
rolw  $8, %ax
rolb  $4, %al
pushl %eax
pushl $2
pushl $saida
call  printf

movl  $0x12345678, %eax
roll  $8, %eax
rolw  $4, %ax
rolb  $4, %al
pushl %eax
pushl $2
pushl $saida
call  printf
```

3) rotacionando regs de 32/16/8 bits 16/8/4 e 8/4/4 bits a direita

```
movl  $0x12345678, %eax
rorl  $16, %eax
rorw  $8, %ax
rorb  $4, %al
pushl %eax
pushl $3
pushl $saida
call  printf

movl  $0x12345678, %eax
rorl  $8, %eax
```

```
      rorw  $4, %ax
      rorb  $4, %al
      pushl %eax
      pushl $3
      pushl $saida
      call  printf
```

4) deslocando regs de 8/16/32 bits 4/8/16 e 4/4/8 bits a esquerda

```
      movl  $0x12345678, %eax
      salb  $4, %al
      salw  $8, %ax
      sall  $16, %eax
      pushl %eax
      pushl $4
      pushl $saida
      call  printf

      movl  $0x12345678, %eax
      salb  $4, %al
      salw  $4, %ax
      sall  $8, %eax
      pushl %eax
      pushl $4
      pushl $saida
      call  printf
```

5) deslocando regs de 32/16/8 bits 16/8/4 e 8/4/4 bits a direita

```
      movl  $0x12345678, %eax
      sarl  $16, %eax
      sarw  $8, %ax
      sarb  $4, %al
      pushl %eax
      pushl $5
      pushl $saida
      call  printf

      movl  $0x12345678, %eax
      sarl  $8, %eax
      sarw  $4, %ax
      sarb  $4, %al
      pushl %eax
      pushl $5
      pushl $saida
      call  printf
```

6) trocando registradores de 32/16/8 bits entre si

```
      movl  $0x12341234, %eax
      movl  $0xabcdabcd, %ebx
      xchgb %al, %bl
      xchgw %ax, %bx
      xchgl %eax, %ebx
      pushl %ebx
      pushl %eax
      pushl $6
      pushl $saida2
```

```
call  printf
```

7) mostrando quais registradores o printf altera. Usando a pilha para backupea-los.

```
movl  $0xAAAAAAAA, %eax
movl  $0xBBBBBBBB, %ebx
movl  $0xCCCCCCCC, %ecx
movl  $0xDDDDDDDD, %edx
movl  $0xEEEEEEEE, %esi
movl  $0xFFFFFFFF, %edi

pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
pushl %eax
pushl $7
pushl $saida3
call  printf

pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
pushl %eax
pushl $7
pushl $saida3
call  printf

addl  $40,  %esp       # desfazendo todos os últimos 11 pushs
                       # para permitir a recuperação dos
                       # registradores empilhados

popl  %eax
popl  %ebx
popl  %ecx
popl  %edx
popl  %esi
popl  %edi

pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
pushl %eax
pushl $7
pushl $saida3
call  printf
```

Resumo: Genericamente, instrução pop possui o seguinte formato:

“***popx*** *reg*” # reg ← pilha (%esp)

onde x = l ou w (não sendo permitido b), dependendo do registrador ser de 32 ou 16 bits. Em situações específicas pode usar as instruções ***pusha*** e ***popa*** para “empilhar todos” e “desempilhar todos” os registradores de 16 bits; use ***pushad*** e ***popad*** para registradores de 32 bits.

DESAFIO: leia 2 números pelo teclado e coloque-os em duas variáveis X e Y; usando deslocamento de bits multiplique o primeiro número por 8, alterando a variável X; quebre o segundo número em dois números de 16bits e some as duas metades gerando um novo número, alterando a variável Y; troque os valores entre X e Y e mostre na tela o resultado obtido.