UNIVERSIDADE ESTADUAL DE MARINGA

Departamento de Informática

Disciplina: PIHS – 9792 -Turma 01

Programação para Interfaceamento de Hardware e Software

Ambiente de Desenvolvimento: plataforma IA-32

Linguagem: GNU Assembly

Aula 09 – Instruções Gnu Assembly – Parte 01

Prof. Ronaldo A. L. Goncalves

O objetivo desta prática é o de 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 registradores de 16 bits

pushad/popad : empilha/desempilha todos os registradores de 32 bits

sall/salw/salb : SAL (Shift Arithmetic Left): desloca bits a esquerda, sobre registradores de 32, 16 ou 8 bits, retirando-os do extremo esquerdo e inserindo 0's no extremo direito.

sarl/sarw/sarb : SAR (Shift Arithmetic Right): desloca bits a direita, sobre registradores de 32, 16 ou 8 bits, retirando-os do extremo direito e inserindo 0's no extremo esquerdo.

rorl/rorw/rorb : ROR (Rotate Right): desloca bits a direita, sobre registradores de 32, 16 ou 8 bits, retirando-os do extremo direito e inserindo-os de volta no extremo esquerdo.

roll/rolw/rolb : ROL (Rotate Left): desloca bits a esquerda, sobre registradores de 32, 16 ou 8 bits, retirando-os do extremo esquerdo e inserindo-os de volta no extremo direito.

xchgl/xchgw/xchgb : (Exchange) troca conteúdos dos registradores entre si

ATENÇÃO: Os sufixos "l, w ou b" nas instruções indicam que a instrução manipula dados **long**, **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 plataforma.

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

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.

Siga os passos a seguir para testar as instruções que foram elencadas.

1) Crie um arquivo “instrucoes01.s” e coloque o seguinte esqueleto fonte inicial

```
.section .data
saida:      .asciz "Teste %d: O valor do registrador é: %X\n\n"
saida2:     .asciz "Teste %d: Os valores dos registradores sao: %X e %X\n\n"
saida3:     .asciz "Teste %d:\n EAX = %X\n EBX = %X\n ECX = %X\n EDX = %X\n ESI = %X\n EDI = %X\n\n"

.section .text
.globl _start
_start:
```

2) Acrescente o trecho de código a seguir para mover, empilhar e imprimir dados de 32, 16 e 08 bits

```
    # teste 1

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

    # teste 2

    movw    $0xABCD, %bx
    pushl   %ebx
    pushl   $2
    pushl   $saida
    call    printf
```

```
# teste 3

movb    $0xEE, %bh
movb    $0xFF, %bl
pushl   %ebx
pushl   $3
pushl   $saida
call    printf

addl    $36, %esp       # desfazendo todos os últimos 9 pushs
                        # para limpar a pilha
```

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

```
pushl   $0
call    exit
```

Salve o arquivo, monte e link. Execute e analise os resultados. **Observe que** uso do símbolo **$** deve ser usado antes de constantes numéricas e antes dos rótulos/variáveis indicam o endereço que eles apontam na memória. Discuta com o professor o que você não entendeu.

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 parênteses) ou registrador; o operando destino pode ser memória (variável ou registrador entre parênteses) ou registrador. O operando destino não pode ser dado imediato (constante ou endereço de variável → $). Os operandos fonte e destino não podem ser simultaneamente memória.

Resumo: Genericamente, instrução push possui o seguinte formato:

"pushx 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.

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.

Agora, insira no programa antes das instruções de finalização do programa, um a um, os trechos de códigos das etapas a seguir, mantendo os trechos anteriores já inseridos. Para cada trecho inserido, monte, link e execute o programa para observar os resultados do respectivo trecho. Discuta com o professor o que você não entendeu.

4) Insira instruções para testar quais registradores o printf altera e use a pilha para backupeá-los e assim protege-los das alterações.

```
# teste 4:

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   $4
pushl   $saida3
call    printf

# teste 5

pushl   %edi
pushl   %esi
pushl   %edx
pushl   %ecx
pushl   %ebx
pushl   %eax
pushl   $5
pushl   $saida3
call    printf

addl    $40, %esp       # desempilha os últimos 10 pushs para liberar os registradores backupeados
```

```
# teste 6

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   $6
pushl   $saida3
call    printf

addl    $32, %esp       # desempilha os 8 últimos pushs
```

5) Insira operações de rotação sobre registradores 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   $7
pushl   $saida
call    printf

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

addl    $24, %esp       # desempilha os 6 últimos pushs
```

6) Insira operações de rotação sobre registradores 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   $9
pushl   $saida
call    printf

movl    $0x12345678, %eax
rorl    $8, %eax
rorw    $4, %ax
rorb    $4, %al
pushl   %eax
pushl   $10
pushl   $saida
call    printf

addl    $24, %esp       # desempilha os 6 últimos pushs
```

OBS: As instruções rol e ror também podem operar sobre variáveis (memória), como por exemplo, "**rorl $16, n**", sendo n uma variável/rótulo para inteiro. Elas também podem ser usadas para rotacionar a quantidade de bits especificadas no registrador de 8 bits "**%cl"**. Lembre-se; o registrador **%cl** é parte menos significativa do registrador **%ecx**. Exemplos podem ser:

- movb $8, %cl : ou "movl $8, %ecx", pois o valor $8 recai sobre o %cl
- roll %cl, %eax : rotaciona 8 bits a esquerda do registrador %eax
- movl n, %ecx : o valor de n recai sobre 0 %cl
- rorl %cl, %eax : rotaciona n bits a direita do registrador %eax

7) Insira operações de deslocamento sobre registradores de 32/16/8 bits 16/8/4 e 8/4/4 bits a esquerda

```
movl    $0x12345678, %eax
salb    $4, %al
salw    $8, %ax
sall    $16, %eax
pushl   %eax
pushl   $11
```

```
pushl   $saida
call    printf

movl    $0x12345678, %eax
salb    $4, %al
salw    $4, %ax
sall    $8, %eax
pushl   %eax
pushl   $12
pushl   $saida
call    printf

addl    $24, %esp       # desempilha os 6 últimos pushs
```

8) Insira operações de deslocamento sobre registradores 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   $13
pushl   $saida
call    printf

movl    $0x12345678, %eax
sarl    $8, %eax
sarw    $4, %ax
sarb    $4, %al
pushl   %eax
pushl   $14
pushl   $saida
call    printf

addl    $24, %esp       # desempilha os 6 últimos pushs
```

OBS: As instruções sal e sar também podem operar sobre variáveis (memória), como por exemplo, "**sarl $16, n**", sendo n uma variável/rótulo para inteiro. Elas também podem ser usadas para deslocar a quantidade de bits especificadas no registrador de 8 bits "**%cl"**. Lembre-se; o registrador **%cl** é parte menos significativa do registrador **%ecx**. Exemplos podem ser:

- movb $8, %cl : ou “movl $8, %ecx”, pois o valor $8 recai sobre o %cl
- sall %cl, %eax : desloca 8 bits a esquerda do registrador %eax
- movl n, %ecx : o valor de n recai sobre 0 %cl
- sarl %cl, %eax : desloca n bits a direita do registrador %eax

9) Insira operações de troca de conteúdo entre registradores de 32/16/8 bits

```
movl    $0x12341234, %eax
movl    $0xabcdabcd, %ebx
xchgb   %al, %bl
xchgw   %ax, %bx
xchgl   %eax, %ebx
pushl   %ebx
pushl   %eax
pushl   $15
pushl   $saida2
call    printf

addl    $16, %esp       # desempilha os 4 últimos pushs
```

Tarefa de Aula: Leia um dado inteiro de 32 bits. Interprete esse dado como sendo 2 dados de 16 bits concatenados. Some essas duas metades e armazene como um dado de 32 bits. Mostre o resultado na tela. Multiplique esse resultado por uma potência de 2 (2 elevado a n), sendo n lido do usuário. Mostre o resultado. Use o registrador %cl para colocar o n.