IA32: Controlo de Fluxo e Variáveis

Guião V - Resolução

Questão 1 (*Comparações*)

a)

```
char ctest(int a, int b) {
       char t1 = a < b;
       return t1;
}
```

```
movl    12(%ebp), %eax
cmpl    %eax, 8(%ebp)
setl    %al
movsbl  %al,%eax
```

b)

```
char ctest(unsigned a, int b){
       char t1 = a < b;
       return t1;
}
```

```
movl    12(%ebp), %eax
cmpl    %eax, 8(%ebp)
setb    %al
movsbl  %al,%eax
```

c)

```
char ctest(int a) {
       char t1 = a > 0;
       return t1;
}
```

```
movl    8(%ebp), %eax
testl   %eax, %eax
setg    %al
movsbl  %al,%eax
```

d)

```
char ctest(int a, int b)
{
       char t1 = a != b;
       char t2 = a >= b;
       char t3 = t1 + t2;
       return t3;
}
```

```
movl    12(%ebp), %eax
movl    8(%ebp), %ecx
cmpl    %eax, %ecx
setne   %dl
cmpl    %eax, %ecx
setge   %al
addb    %al, %dl
movsbl  %dl,%eax
```

Questão 2 (*Controlo de fluxo*)**:**

a) A instrução **`jbe`** tem como endereço-alvo um valor relativo ao PC/IP (depois de ele ter sido incrementado para apontar para a próxima instrução), i.e.,***0x8048d1c + 2 + 0xffffffda = 0x8048cf8***, como se pode ver no código "desmontado" (notar a extensão do bit de sinal em 0xda).

```
8048d1c: 76 da          jbe 0x8048cf8
```

b) O endereço-alvo da instrução **`jmp`** é ***0x8048d44*** que foi calculado adicionando o valor 0x54 ao endereço da próxima instrução Assim, temos para endereço da instrução **`mov`** a posição:

```
0x8048d44 – 0x54 ->  0x804c8cf0
```

```
0x8048cee: eb 54            jmp 8048d44
0x8048cf0: c7 45 f8 10 00   mov $0x10,0xfffffff8(%ebp)
```

c) O endereço-alvo está à distância ***0x000000cb*** relativo ao endereço da instrução **`nop`**), pelo que o endereço na instrução **`jmp`** é calculado adicionando aqueles dois valores.

```
8048902: e9 cb 00 00 00     jmp 0x80489d2
```

Questão 3 (*vetores de tipo simples*)**:**

```
$ gcc -S -O0 vectorInt.c
```

```
      .file "vectorInt.c"
      .text
.globl ini
      .type ini,@function
ini:
      pushl %ebp
      movl  %esp, %ebp
      movl  $0, i
.L2:
      cmpl  $99, i
      jle   .L5
      jmp   .L1
.L5:
      movl  i, %edx
      movl  i, %eax
      movl  %eax, vector(,%edx,4)
      incl  i
      jmp   .L2
.L1:
      leave
      ret
```

```
.Lfe1:
      .size ini,.Lfe1-ini
      .section    .rodata
.LC0:
      .string     "Sum=%d\n"
      .text
.globl main
      .type main,@function
main:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      call  ini
      movl  $0, sum
      movl  $0, i
.L7:
      cmpl  $99, i
      jle   .L10
      jmp   .L8
.L10:
      movl  i, %eax
      movl  vector(,%eax,4), %eax
      addl  %eax, sum
      incl  i
      jmp   .L7
.L8:
      subl  $8, %esp
      pushl sum
      pushl $.LC0
      call  printf
      addl  $16, %esp
      leave
      ret
.Lfe2:
      .size main,.Lfe2-main
      .comm vector,400,32
      .comm sum,4,4
      .comm i,4,4
```

a) Identifique e explique as instruções responsáveis pelo **ciclo `for (...)`**

```
    movl   $0, i       ; i=0
.L2:
    cmpl   $99, i      ; i-99 -> afeta as flags
    jle    .L5         ; se (i<=99) salta para .L5 (corpo do ciclo)
    jmp    .L1         ; senão (i>=100) salta para .L1 (fim do ciclo)
.L5:
    ...                ; corpo do ciclo
    incl   i           ; ++i
    jmp    .L2         ; regressa a .L2 (controlo do ciclo)
.L1:
```

b) Identifique e explique as instruções responsáveis pelo cálculo do endereço de **`vector[i]`**

```
    movl   i, %edx                  ; edx=i
    ...
    movl   %eax, vector(,%edx,4)    ; end_destino=edx*4+vector=i*4+vector
```

Questão 4 (*vetores de estruturas*)**:**

```
$ gcc -S -O0 vectorEstrutura.c
```

```
      .file "vectorEstrutura.c"
      .text
.globl ini
      .type ini,@function
ini:
      pushl %ebp
      movl  %esp, %ebp
      movl  $1, i
.L2:
      cmpl  $99, i
      jle   .L5
      jmp   .L1
.L5:
      movl  i, %eax
      movb  $0, vector(,%eax,8)
      movl  i, %edx
      movl  i, %eax
      movl  %eax, vector+4(,%edx,8)
      incl  i
      jmp   .L2
.L1:
      leave
      ret
```

```
.Lfe1:
      .size ini,.Lfe1-ini
      .section   .rodata
.LC0:
      .string    "Sum=%d\n"
      .text
.globl main
      .type main,@function
main:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      call  ini
      movl  $0, sum
      movl  $0, i
.L7:
      cmpl  $99, i
      jle   .L10
      jmp   .L8
.L10:
      movl  i, %eax
      movl  vector+4(,%eax,8), %eax
      addl  %eax, sum
      incl  i
      jmp   .L7
.L8:
      subl  $8, %esp
      pushl sum
      pushl $.LC0
      call  printf
      addl  $16, %esp
      leave
      ret
.Lfe2:
      .size main,.Lfe2-main
      .comm vector,800,32
      .comm sum,4,4
      .comm i,4,4
```

a) Identifique e explique as instruções responsáveis pelo cálculo do endereço de **vector[i].a**. Compare com a resposta à questão 3.b).

```
movl   i, %edx                  ; edx=i
movl   i, %eax
movl   %eax, vector+4(,%edx,8)  ; endereço=edx*8+vector+4=i*8+vector+4
```

Cada elemento da estrutura ocupa **8** bytes (4 chars + 1 int), logo o endereço de **vector[i]** é **vector+8*i** e o endereço de **vector[i].a** é **vector+8*i+4**, porque o campo **a** está afastado **4** bytes em relação ao início do elemento **i** do **vector**.

Em relação à questão 3.b as diferenças são:
i) O tamanho de cada elemento do *array* é 8 em vez de 4;
ii) Exige-se um deslocamento extra de 4 bytes para obter o endereço do campo **a**.

b) Modifique no código em C o tamanho do campo **s** da estrutura para **8** caracteres. Identifique e explique as instruções responsáveis pelo cálculo do endereço de **vector[i].a**. Compare com a resposta à alínea anterior.

```
typedef struct {
  char s[8];
  int a;
} tipoEstrutura;
```

```
$ gcc -S -O0 vectorEstrutura_4B.c
```

```
      .file "vectorEstrutura_4B.c"
      .text
.globl ini
      .type ini,@function
ini:
      pushl %ebp
      movl  %esp, %ebp
      movl  $1, i
.L2:
      cmpl  $99, i
      jle   .L5
      jmp   .L1
.L5:
      movl  i, %edx
      movl  %edx, %eax
      sall  $1, %eax
      addl  %edx, %eax
      sall  $2, %eax
      movb  $0, vector(%eax)
      movl  i, %edx
      movl  %edx, %eax
      sall  $1, %eax
      addl  %edx, %eax
      leal  0(,%eax,4), %edx
      movl  i, %eax
      movl  %eax, vector+8(%edx)
      incl  i
      jmp   .L2
.L1:
      leave
      ret
```

```
.Lfe1:
      .size ini,.Lfe1-ini
      .section   .rodata
.LC0:
      .string    "Sum=%d\n"
      .text
.globl main
      .type main,@function
main:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      call  ini
      movl  $0, sum
      movl  $0, i
.L7:
      cmpl  $99, i
      jle   .L10
      jmp   .L8
.L10:
      movl  i, %edx
      movl  %edx, %eax
      sall  $1, %eax
      addl  %edx, %eax
      sall  $2, %eax
      movl  vector+8(%eax), %eax
      addl  %eax, sum
      incl  i
      jmp   .L7
.L8:
      subl  $8, %esp
      pushl sum
      pushl $.LC0
      call  printf
      addl  $16, %esp
      leave
      ret
.Lfe2:
      .size main,.Lfe2-main
      .comm vector,1200,32
      .comm sum,4,4
      .comm i,4,4
```

ini()

```
movl   i, %edx                ; edx=i
movl   %edx, %eax             ; eax=i
sall   $1, %eax               ; eax=i*2
addl   %edx, %eax             ; eax=i*2+i=i*3
leal   0(,%eax,4), %edx       ; edx=(i*3)*4+0=i*12
movl   i, %eax
movl   %eax, vector+8(%edx)   ; i*12+vector+8
```

main()

```
movl   i, %edx                 ; edx=i
movl   %edx, %eax              ; eax=i
sall   $1, %eax                ; eax=i*2
addl   %edx, %eax              ; eax=i*3
sall   $2, %eax                ; eax=(i*3)*4=i*12
movl   vector+8(%eax), %eax    ; i*12+vector+8
```

Cada elemento da estrutura ocupa **12** bytes (8 chars + 1 int), logo o endereço de **vector[i]** é **vector+12*i** e o endereço de **vector[i].a** é **vector+12*i+8**, porque o campo **a** está afastado **8** bytes em relação ao início do elemento **i** do **vector**.

Comparando com 4.a), o cálculo do endereço é bastante mais complexo e utiliza mais 4 instruções, que são necessárias para fazer a multiplicação por 12. Esta multiplicação por 12, como não é potência de 2, não pode ser feita com o fator de escala dos modos de endereçamento do IA32.