CS 271 Computer Architecture and Assembly Language

Self-Check for Lecture#14

Solutions

Here is a partial data segment:

```
MAX = 50
.data
...
list        DWORD       MAX DUP(0)
a           DWORD       25
b           DWORD       15
  ...
```

1. Given: the address of `list` is 0x0300.
   a. What is the (hexadecimal) address of a? ___0x03C8___
      list takes 200 (decimal) bytes of memory = 0xC8 bytes. So the address of a is the address of list + the size of list = 0x0300 + 0xC8 = 0x03C8

   b. What is the (hexadecimal) address of the 33rd element of list? ___0x0380___
      (Hint: in C or Java, the 33rd element is list[32])
      32 elements of list take 32 x 4 = 128 (decimal) bytes of memory = 0x80 bytes. So the address of the 33rd element is the address of list + the number of bytes taken by the first 32 elements
        = 0x0300 + 0x80 = 0x0380

Here is a partial "listing file" that uses the data segment above:

```
00000000       main        PROC
00000000                   push   a
00000005                   push   b
0000000A                   push   OFFSET list
0000000F                   call   someProc
00000014       next        ...

                           exit   ;exit to operating system
0000006C       main        ENDP

0000006C       someProc    PROC
0000006C                   push   ebp
0000006F                   mov    ebp, esp
00000072       etc         ...

0000008B   C3              ret    ;return to calling procedure
0000008C       someProc    ENDP
```

2. Initially, `esp` contains 0A04, and `ebp` contains 0BB9. `main` has called `someProc`, and the first two statements of `someProc` have been executed.

   a. ebp contains 0x09F0
   b. Show the contents of the system stack →
   c. Write a statement to move the value of actual parameter a into the eax register. (Global name a is not permitted.)

      mov eax, [ebp + 16]

   d. Write the statements to move the value of the bth element of list into the ebx register. (Consider b=0 to be the 1st element of list) (Global names b and list are not permitted.)

Address	Contents	Meaning
09E4		
09E8		
09EC		
09F0	0x0BB9	old ebp
09F4	0x0014	return @
09F8	0x0300	@ list
09FC	15	value of b
0A00	25	value of a
0A04	?	unknown

```
mov     esi, [ebp + 8]          ; move the OFFSET of list into ESI
mov     eax, 4                  ; there are 4 bytes per DWORD
mov     ebx, [ebp + 12]         ; move the value of b into EBX
mul     ebx                     ; Multiply EAX by b to (almost) get the offset to the bth element
sub     eax, 4                  ; EAX now holds the offset from ESI to the bth element of list
mov     ebx, [esi+eax]          ; move the element into EBX
```

3. Given the following partial data segment:

```
.data
loVal           DWORD       ?
hiVal           DWORD       ?
randVal         DWORD       ?


.code
main      PROC
          call   Randomize           ; from the Irvine library

;   Code to get loVal and hiVal from the user goes here.

          push   loVal
          push   hiVal
          push   OFFSET randVal
          call   nextRand

;   More main procedure code

          exit
main      ENDP
```

Write the `nextRand` procedure so that it satisfies the following header documentation. You may use appropriate Irvine library procedures. Note that used registers must be saved and restored.

```
; Procedure nextRand
; Procedure to get the nest random number in the range specified by the user.
; Receives parameters on the system stack (in the order pushed):
;        Lowest acceptable value (loVal)
;        Highest acceptable value (hiVal)
;        Address of return value
; Preconditions:  loVal < hiVal
; Registers used:  none

nextRand    PROC
      pushad                  ;save registers
      mov   ebp,esp           ;set stack frame pointer
      mov   eax,[ebp+40]      ;hiVal in eax
      sub   eax,[ebp+44]      ;subtract loVal
      inc   eax               ;  and add 1 to get the number of integers in range
      call  RandomRange       ;eax gets value in [0 .. range-1]
      add   eax,[ebp+44]      ;eax has value in [loVal .. hiVal]
      mov   edi,[ebp+36]      ;edi gets destination memory address
      mov   [edi],eax         ;send result to memory
      popad                   ;restore registers
      ret   12                ;return and clear activation record
nextRand    ENDP
```