COAL ASSIGNMENT

20k1898

Muhammad Bilal Khan 20k1898

Q1a) The only parameters in the provided data are 4 and 10, which are present at places 0xFFFF00E8 and 0xFFFF00E4 respectively.

b) At address 0xFFFF0100, only one local variable, namely 8, is present

c) According to the specified program set, the ESP is updated following a push instruction by decrementing by 4 bytes, which is equal to the size of the local variable.

d) In the program above, EBP has the value 0xFFFF0100. This is the base of the stack frame's address.

e) The value of [EBP-12] is 0xFFFF00F4. This is the location where the value of EAX is stored.

f) After "Lea ESP [EBP-12]", the value of ESP will be 0xFFFF00F4. This instruction moves the stack pointer to point to the location where the value of EAX is stored, which is 12 bytes below the current value of EBP.

g) The value of EBP after the procedure returns will be the same as it was before the procedure was called. The procedure uses the stack frame of the calling function, and the value of EBP is not modified within the procedure.

Q2 Stack memory map

Parameter 2	ESP + 0x0C
Parameter 1	ESP + 0x08
Local variable 4	ESP + 0x04
Local variable 3	ESP + 0x00
Local variable 2	ESP – 0x04
Local variable 1	ESP - 0x08
Return Address	ESP – 0x0C
Stack	
Space	

Parameter 1: located at [ESP+4]

Parameter 2: located at [ESP+8]

Local variable 1: located at [ESP-4]

Local variable 2: located at [ESP-8]

Local variable 3: located at [ESP-12]

Local variable 4: located at [ESP-16]

To access stack parameters and local variables

Parameter 1: MOV EAX, [ESP+4]

Muhammad Bilal Khan 20k1898

Parameter 2: MOV EAX, [ESP+8]

Local variable 1: MOV EAX, [ESP-4]

Local variable 2: MOV EAX, [ESP-8]

Local variable 3: MOV EAX, [ESP-12]

Local variable 4: MOV EAX, [ESP-16]

Q3 INCLUDE Irvine32.inc

```
.data
  array: dw 1, 2, 3, 4, 5
  array_size: equ 5
  sum: dd 0

.code
  MAIN PROC
  mov eax, 0
  mov ebx, array_size
  mov ecx, 0
  lea edx, [sum]
  lea esi, [array]



  call SUM


  mov eax, 4
  mov ebx, 1
  mov ecx, sum
  mov edx, 4
```

Muhammad Bilal Khan 20k1898

```
  mov eax, 1
  xor ebx, ebx


SUM:
  push ebp
  mov ebp, esp


  mov esi, [ebp+16]          ; address of array
  mov edi, [ebp+12]          ; offset
  mov eax, [ebp+8]           ; index
  mov ebx, [ebp+4]           ; size
  mov edx, [ebp+20]          ; address of sum variable

  cmp ebx, 0
  je done

  add eax, edi               ; calculate index+offset
  movzx edx, word [esi+eax*2] ; get array element at index+offset
  add [edx], eax             ; add element value to sum
  dec ebx                    ; decrement size
  call SUM                   ; recursive call
  add esp, 4

done:
   mov esp, ebp
```

```
  pop ebp
  ret
exit
main ENDP
END MAIN
Q4 main PROC
INVOKE differentinputs, input1, input2, input3
INVOKE differentinputs, input4, input5, input6
INVOKE differentinputs, input7, input8, input9
INVOKE differentinputs, input10, input11, input12
INVOKE differentinputs, input13, input14, input15
    Invoke ExitProcess,0
Main endp
END MAIN
Differentinputs PROC firstinput: DOWRD, secondinput: DWORD, thirdinput: DWORD :
Push ebx
Push edx
Mov eax,firstinput
Mov ebx,secondinput
Mov edx,thirdinput
Cmp eax,ebx
Je notdifferent
Cmp ebx,edx
Je notdifferent
Mov eax,1
Jmp ending
```

Muhammad Bilal Khan 20k1898

```
Notdifferent:
Mov eax,0
Ending:
Pop edx
Pop ebx
Rec
Differentinputs endp


.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:DWORD
INCLUDE irvine32.inc
.data
Input1 DWORD 13
Input2 DWORD 14
Input3 DWORD 15
Input4 DWORD 5
Input5 DWORD 5
Input6 DWORD 6
Input7 DWORD 999
Input8 DWORD 16134
Input9 DWORD 999
Input10 DWORD 1324
Input11 DWORD 11
Input12 DWORD 898
Input13 DWORD 134134
```

Muhammad Bilal Khan 20k1898

Input14 DWORD 17

Input14 DWORD 17

.code

Q5 INCLUDE Irvine 32.inc

CountNearMatches PROTO, Pointer Arr1:PTR SDWORD, Pointer Arr2:PTR SDWORD, Arr Size:DWORD, d:DWORD

data

First Arr SDWORD 1,2,3,4,5 First Arr1 SDWORD 6,7,8,9,10 Second Arr SDWORD 11,12,13,14,15

Second Arri SDWORD 16,17,18,19,20 count DWORD ?,0

difference1 DWORD 11

difference2 DWORD 0

.code

main PROC

INVOKE CountNearMatches, ADDR First Arr, ADDR First Arr1, LENGTHOFFirst Arr, difference1

call Writeint

call Crif

INVOKE CountNearMatches, ADDR Second_Arr, ADDR Second_Arr1,

LENGTHOFSecond_Arr, differencez

call Writeint:Library Function to display a message in Irvine 32 call Crif ¡Library Function

exit

main ENDP

mov edi,Pointer Arr2

index register.

mov ecx,Arr_Size Lable1:

mov ebx,0

mov ebx, [esi]

register.

register.

mov edx,0

mov edx,[edi]

IF ebx > edx

mov eax,ebx

Q6 INCLUDE Irvine32.inc

```
Extended_Sub PROC
   push ebp
   mov ebp, esp

   mov eax, [ebp+8]
   mov ebx, [ebp+12]
   mov ecx, [ebp+16]

   xor edx, edx
   subloop:
      mov al, [eax+ecx-1]
      sbb al, [ebx+ecx-1]
      mov [eax+ecx-1], al
      dec ecx
      jnz subloop
      pop ebp
   ret
Extended_Sub ENDP

main PROC
```

```
    int1 BYTE 10101010b, 01010101b, 11110000b, 00001111b, 10000000b, 00000001b, 11001100b, 00110011b, 01010101b, 10101010b
    int2 BYTE 01010101b, 10101010b, 00001111b, 11110000b, 00000001b, 10000000b, 00110011b, 11001100b, 10101010b, 01010101b

    mov ecx, LENGTHOF int1
    call Extended_Sub

    mov edx, OFFSET int1
    call WriteHexDump
    call Crlf

    exit
main ENDP
```

Q7

```
INCLUDE Irvine32.inc

Extended_Add PROC
    push ebp
    mov ebp, esp

    mov eax, [ebp+8]
    mov ebx, [ebp+12]
    mov ecx, [ebp+16]

    xor edx, edx
    addloop:
        mov al, [eax+ecx-1]
```

```
        adc al, [ebx+ecx-1]
        mov [eax+ecx-1], al
        dec ecx
        jnz addloop

    pop ebp
    ret
Extended_Add ENDP

main PROC
    int1 BYTE 10101010b, 01010101b, 11110000b, 00001111b, 10000000b, 00000001b, 11001100b, 00110011b, 01010101b, 10101010b
    int2 BYTE 01010101b, 10101010b, 00001111b, 11110000b, 00000001b, 10000000b, 00110011b, 11001100b, 10101010b, 01010101b

       mov ecx, LENGTHOF int1
    call Extended_Add

    mov edx, OFFSET int1
    call WriteHexDump
    call Crlf
        exit
main ENDP
```

Q8 INCLUDE Irvine32.inc

```
GCD PROC
    push ebp
    mov ebp, esp
    mov eax, [ebp+8]   ; a
```

```
    mov ebx, [ebp+12]  ; b

    cmp eax, 0
    jne a_not_zero
    mov eax, ebx
    jmp end_gcd

a_not_zero:
    cmp ebx, 0
    jne b_not_zero
    jmp end_gcd

b_not_zero:
    ; Base case
    cmp eax, ebx
    je end_gcd

    ; Recursive case
    cmp eax, ebx
    ja subtract_a
    sub ebx, eax
    jmp GCD

subtract_a:
    sub eax, ebx
    jmp GCD
```

```
end_gcd:
   pop ebp
   ret
GCD ENDP

main PROC
   pairs DD 5, 20, 24, 18, 432, 226
      mov ecx, LENGTHOF pairs
   shr ecx, 1
   mov esi, OFFSET pairs
   gcdloop:
      mov eax, [esi]      ; a
      mov ebx, [esi+4]    ; b
      call GCD
      call WriteInt
      call Crlf
      add esi, 8
      loop gcdloop
      exit
main ENDP
```

Q9

```
INCLUDE Irvine32.inc
Procedure prototype CountMatches PROTO, pArr1: PTR SDWORD,
pArr2: PTR SDWORD, length: DWORD
dala
str1 BYTE "The number of matching elements is: ",0
```

Muhammad Bilal Khan 20k1898

array1 SDWORD-3, +3, -5, +7, -3, -2

array2 SDWORD +4, +3,-5, -3, -1, +8

.code

main PROC

calls the procedures

call Cirscr; clears the screen

find the number of matching elements in arrays arr1, an2

INVOKE CountMatches, ADDR arr1, ADDR arr2, LENGTHOF ar1 mov edx, OFFSET stri

call WriteString: writes str1

call WriteDec; writes EAX

call Crif

exit

main ENDP

CountMatches PROC USES esi edi ecx, pArrt: PTR SDWORD, points the 15 array pArr2: PTR SDWORD,; points the 2nd array length: DWORD; the length of two arrays finds the number of matching array elements Receives: pointers to two arrays and their length

Returns: EAX = number of matching array elements

Relums: EAX= number of matching away elements mov eax,0; initialize EAX=0

mov esi parr1; ESI is the pointer to 1" array

mov edi pan2: EDI is the pointer to 2nd array

mov ecxlength

L1:

cmp [es][ed]

jne L2

inc eax; increments EAX on a match

12:

inc esi; ESI is incremented

inc edi: EDI is incremented

loop L1

ret; returns EAX

CountMatches ENDP

END main

Q10 i. 77 F7

ii. 51

iii. 8B 04 DI A6 2A

iv. 23 04 DI ED BP

v. 0B 1C DI F8 20 BX

vi. E2 E7

vii. 8B 34 SI 90 04 SP

viii. 8E 1C DI 76 04 ES

ix. B8 FE FE

x. C1 E8 03

xi. 58

xii. 80 03 F8

xiii. 2D XX XX (where XX XX is the two's complement of VAR)

xiv. 8C D8

Q11 i. MOV AX, 20B9h

ii. MOV [EDI-2], ebp

iii. LOOP -119

iv. MOV [EBP-23DC], GS

v. MOV AX, 2530h

vi. MOV DS, AX

vii. MOV [EDI-4], edi

viii. CALL -30CE

Muhammad Bilal Khan 20k1898

ix. CMP CH, BL

x. MUL EDI

Q12a) The MIPS instruction 'bne' compares the contents of two registers and, if they are not equal, redirects execution to the instruction at the specified label. The execution process involves fetching the instruction, decoding it, reading the register contents, performing the comparison, calculating the branch address, and updating the program counter. If the register contents are equal, the instruction is considered false, and execution proceeds to the next instruction sequentially.

b) In MIPS, the 'addi' instruction adds a signed immediate value to a register, stores the result in another register, and increments the program counter. The execution process includes fetching the instruction, decoding it, reading the register contents, adding the immediate value to the register contents, storing the result in the designated register, and incrementing the program counter. The 'addi' instruction is typically designed to execute within a single clock cycle and is commonly utilized for small incremental or decremental operations on registers.

13a) In a RISC processor, the "Store R6, 1000h(R8)" instruction is executed through a series of steps: instruction fetch, instruction decode, register fetch, calculation of memory address, data write to memory, and update of the program counter. This particular sequence is characteristic of RISC processors, which prioritize high performance by employing a streamlined and regular instruction pipeline.

13b) Executing the "Subtract R6, R4, R7" instruction in a RISC processor involves the following steps: instruction fetch, instruction decode, register fetch, execution of the subtraction operation, and update of the program counter. This sequence of actions is in line with the typical functioning of RISC processors, which are designed to achieve optimal performance through a simplified and consistent instruction pipeline.