**FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

KARACHI CAMPUS

Grand Assignment Fall 2022

*Computer Organization & Assembly Language*

Total Points: **155**

Solve on this paper, and attached the program results

Roll No: **21k-4653** Section : **3K** Signature: \_\_\_\_\_\_\_\_\_\_

Part I

Question No. 1: Programming Basics [10\*02 = 20 Points]

Machine Language

(i) The following bytes are found in order somewhere in memory. Assuming they are machine codes, decode the values into meaningful assembly language mnemonics. [Solve this when Machine Language is covered in the class]

B9 00 12,8C 85 DC 01

**Mov ecx, 858C1200h**

**Fadd Qword PTR [ecx]**

(ii) Convert the following independent Assembly Language instructions to Machine Language code – give your answers in hexadecimal:

MOV [SI+490], SP

**89 A4 EA 01**

**1010 0100**

ADD AL, [BX + SI]

**67 02 00**

JNZ NEXT ; NEXT is a label at offset 0008H and

**0F 85 02 00 00 00**

PUSH AX

**66 50**

MOV AX, VAR + 6 ; OFFSET of VAR is 0002H

**66 A1 08 00 00 00**

SUB CX, VAR2 ; OFFSET of VAR2 is 0008H

**66 2B 0D 00 00 00 08**

INC DX

**66 42**

(iii) In the following instruction sequence, show the resulting value of AL where indicated, in hexadecimal:

MOV AL,7AH

NOT AL ; a. AL = **85h**

MOV AL,3DH

AND AL,74H ; b. AL = **34h**

MOV AL,9BH

OR AL,35H ; c. AL = **0BFH**

MOV AL,72H

XOR AL,0DCH ; d. AL = **0AEH**

(iv) Differentiate between the following Assembly Language instructions:

MOV EAX, OFFSET VAR1 ; **EAX will hold the address of VAR1, i.e. its offset.**

MOV EAX, VAR1 ; **EAX will hold the *value* of VAR1**

(v) List *four* important uses of the runtime stacks in programs.

* **Preserve register/memory values throughout procedures.**
* **Create and store local variables.**
* **Store the return address for procedures.**
* **Pass parameters to procedures.**

(vi) Suppose EAX=1234H, EBX=5678H, ECX=9ABCH, and ESP=100H, Give the contents of EAX, EBX, ECX, and ESP after the execution of the following instructions:

PUSH EAX

PUSH EBX

XCHG EAX, ECX

POP ECX

PUSH EAX

POP EBX

1. EAX : **9ABCH** b) EBX : **9ABCH** ECX : **5678H** c) ESP : **0FCH**

(vii) What additional instructions are generated by the assembler as a result of assembling the following procedure?

MYSUM PROC USES ESI ECX

MOV ECX, 10

L1:

ADD EAX, [ESI]

SUB ESI, 4

LOOP L1

ret

MYSUM ENDP

MYSUM PROC

PUSH ESI

PUSH ECX

MOV ECX, 10

L1:

ADD EAX, [ESI]

SUB ESI, 4

DEC ECX

JNZ L1

POP ECX

POP ESI

ret

MYSUM ENDP

(viii) Generate a Map file for an assembly language program that has a code size of 100h bytes, data size of 50h bytes and a stack of 200h bytes. Using this map file, give the contents of CS, DS, and SS registers if this program is loaded at address of 508A0h.

|  |  |  |
| --- | --- | --- |
| Segment Type | Address | Lengths |
| CODE | 0x0000 | 100h |
| DATA | 0x0100 | 50h |
| STACK | 0x0150 | 200h |

Cs register: 0x508A0 + 0x0000 = 0x508A0

Ds register: 0x508A0 + 0x0100 = 0x509A0

Ss register: 0x508A0 + 0x0150 = 0x509F0

(ix) The shown program sets AH to a value depending on the comparison result of unsigned integers V1 and V2. For each condition in the table below, use “√” sign to indicate which value AH will have after the program is executed. If there are more than one possibility, use “?” sign to indicate which value of AH is possible.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AH =1 | AH=2 | AH=3 |
| If V1=V2 then |  |  | Checkmark with solid fill |
| If V1<V2 then |  | Checkmark with solid fill |  |
| If V1>V2 then | Checkmark with solid fill |  |  |

.DATA

V1 DB(?)

V2 DB(?)

.CODE

Start:

•

•

MOV AL, V1

CMP AL, V2

JZ Label1

JS Label1

MOV AH, 1

JMP Continue

Label1:

JE Label2

MOV AH, 2

JMP Continue

Label2:

MOV AH, 3

Continue: . . .

1. Give the contents of the following registers, along with the run-time stack, when the following instructions are executed. Initially, consider ESP = 00001FF8h.

Note: SOLVE THIS PART HERE. No Marks will be awarded without proper working using the stack diagrams.

X1 DWORD 25H

X2 DWORD 27H

MAIN PROC

PUSH 6H

PUSH 5H

CALL P1

11500000H MOV RESULT, EAX ; ESP: **\_\_00001FF8h\_**

MAIN ENDP

P1 PROC

115000A4H PUSH EBP

MOV EBP, ESP ; EBP: **\_\_\_\_1FE2h\_\_\_**

|  |
| --- |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

MOV EAX, [EBP+8]

ADD EAX, [EBP+12] ; EAX: **\_\_\_\_0Bh\_\_\_\_\_**

PUSH OFFSET X1

PUSH OFFSET X2 ; ESP: **\_\_\_1FDAh\_\_\_\_**

|  |
| --- |
| ? (OFFSET X2) |
| ? (OFFSET X1) |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

POP ESI

POP EBX

ADD [ESI], EAX ; X2: **\_\_\_\_32h\_\_\_\_\_**

ADD [EBX], EAX ; X1: **\_\_\_\_30h\_\_\_\_\_**

|  |
| --- |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

MOV ESP, EBP

POP EBP

|  |
| --- |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

RET 8 ; EIP: **\_\_11500000h\_**

***; Stack empty***

P1 ENDP

**Part II**

Q. No 2 Answer all the questions in this section. [2x22=44]

.DATA

BARRAY BYTE 10H, 20H, 30H, 6 DUP (0AH)

ALIGN 4

WARRAY WORD 5 DUP(1000H)

PRESSKEY EQU <"PRESS ANY KEY TO CONTINUE ...",0>

DARRAY DWORD 5 DUP(56789ABH),7 DUP(12345678H)

PROMPT BYTE PRESSKEY

What will be the value of EAX, and AL after executing each of the following instructions? Assume that the address of barray is 404000h.

1. MOV EAX, TYPE WARRAY ; EAX = **2h**
2. MOV EAX, LENGTHOF BARRAY ; EAX = **9h**
3. MOV EAX, SIZEOF DARRAY ; EAX = **30h**
4. MOV EAX, OFFSET WARRAY ; EAX = **40400Ch**
5. MOV EAX, DWORD PTR BARRAY ; EAX = **0A302010h**
6. MOV AL, BYTE PTR DARRAY ; AL = **0ABh**
7. Would the following instruction set the zero flag? Explain.

MOV AX, 0000h ; clear the AX register

No, the instruction MOV AX, 0000h will not set the zero flag.

The zero flag (ZF) is a bit in the FLAGS register that is set to 1 when the result of an arithmetic or logical operation is zero, and set to 0 otherwise. This instruction simply moves the value of 0000h (0 in decimal) into the AX register, without performing any arithmetic or logical operation. Therefore, the zero flag will not be affected by this instruction.

MOV AX, 0000h

MOV BX, 0000h

; Set the zero flag by performing an AND operation on two operands that are both zero

AND AX, BX

; Set the zero flag by performing a subtraction operation

SUB AX, AX

1. Is it possible for a NEG instruction to set the Overflow flag?

**Yes, if the value that is negated is the smallest negative value in the range. For example, the following instructions will set the overflow flag**

Mov al, 10000000b

Neg al

Consider a program that has the following data segment:

I EQU 2Eh, 2h

J BYTE '6789'

K EQU 140

L WORD 3412h, 8765h

M DWORD 4, 3, 5, 6, 7

Indicate whether the following instructions are valid or not. If valid, give the result of the operation in hexadecimal. If invalid, give the reason.

1. MOV AL, I+1 **; A 32-bit value can not be moved to an 8-bit register**
2. MOV AL, J+2 **; AL = 38h or ‘8’**
3. MOVSX EAX, L[1] **; EAX = 00006534h**
4. MOV EBX, M[2] **; EBX = 0003000h**
5. INC [ESI] ;ESI = OFFSET J , **Invalid instruction – the operand to INC must have some size, for example: INC BYTE PTR [ESI]**
6. MOV I, L ; **Invalid mem to mem move**
7. MOV EAX, DWORD PTR J **; EAX = 39383736h**
8. MOV L, WORD PTR M **; Invalid mem to mem move**
9. MOV ESI, L **; A 16-bit value can not be moved to a 32-bit register**
10. Consider the following code:

mov ax, 0h

mov cx, 0Ah

doLoop:

dec ax

loop doLoop

What is the value of the *ax* register after the completion of the doLoop?

Ax = **FFF6h**

1. When an interrupt occurs, arrange the following operations in their order of occurrence?

a) interrupt service routine executed

b) the registers are restored by popping their values off of the stack

c) the processor identifies the source of the interrupt

d) the program counter and other registers' values are pushed onto the stack

e) the address of the interrupt service routine is placed in the program counter

1. **C** 2. **D** 3. **E** 4. **A** 5. **B** [02]
2. In the following code sequence, show the value of AL after each shift or rotate instruction has executed:

mov al,0D4h

shr al,1 ; a. AL = **6Ah**

mov al,0D4h

sar al,1 ; b. AL = **0EAh**

Suppose that you have the following initial register content: AX=F2E9H, BX=0002H CX=08A0H and DX=F1E0H

1. Show the contents of AX and the flags (CF,OF,SF and ZF) after executing:

ADD AX, BX ; a.CF = **0** b.OF = **0** c.SF = **1** d.ZF = **0** AX = **0F2EBh**

1. Show the contents of CX and the flags (CF,OF,SF and ZF) after executing:

SUB CX, DX ; a.CF = **1** b.OF = **0** c.SF = **0** d.ZF= **0**  **CX = 16C0h**

1. Show the contents of BX and the flags (CF,OF,SF and ZF) after executing:

NEG BX ; a.CF = **1** b. OF = **0** c.SF= **1** d.ZF= **0** **BX = 0FFFEh**

1. After the execution of the following sequence of instructions, what is the value of EAX?   
   MOV AH, 9Fh   
   MOV AL, FFh   
   XOR AH,AH   
   OR AH,AL

EAX = 0x00FF

1. Write a single instruction to mask out 1st and 3rd nibble of EAX

AND EAX, 0xF0F0h

1. Compares the integers 7FFFh and 8000h and show how the JB (unsigned) and JL (signed) instructions would generate different results.

**After the comparison, the relevant flag values are:**

**CF = 0, ZF = 0, SF = 0, OF = 1**

**The carry flag is clear, so JB would not execute.**

**The sign flag is clear, but the overflow flag is set, so JL would execute.**

Question No.3 : Assembly Language Programming [7x5=35 Points]

1. Implement the following pseudo-code in assembly language (Intel IA-32 and MIPS code) . Also, give the corresponding data definition directives:

(a)

; All values are

; 32-bit signed integers

while (OP1 < OP2)

{

OP1++;

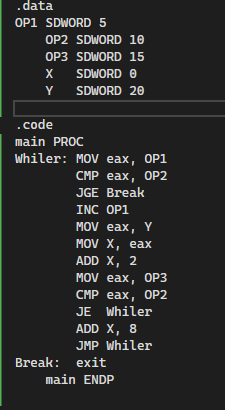
if (OP3 == OP2)

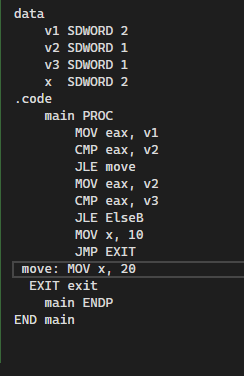
X = Y + 2;

else

X = Y + 10;

}





(b)

; All values are

; 32-bit unsigned integers

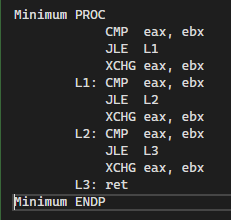
if(VAL1>VAL2) AND (VAL2>VAL3) then

X=10

else

X=20

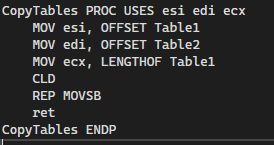
1. Write an assembly language procedure MINIMUM that is called from the MAIN procedure to find the minimum MIN among X, Y and Z. The arguments are passed by value to the procedure MINIMUM using registers. The result is also returned in a register. Also, write the corresponding data definition directives. The Intel IA 32 and MIPS version of this program is required.

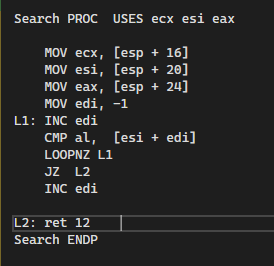


1. Suppose that there are two tables defined in the data segment, DS=2FF0H, namely Table1 and Table2. Table1 is at offset 1000H and Table2 is at offset 2000H. Both tables have a size of 100 bytes.

***Solve here***

1. Write a code segment to copy the content of Table1 to Table2.
2. Write a subroutine to search for a constant number that can be represented in a byte, in a table, and returns the index of the table where the number is found in the DI register. Assume that the constant number to be searched is pushed first in the stack, followed by the table address, and finally the size of the table. Then, write a code segment to search for the number 5 in Table1 and the number 10 in Table2, using the subroutine, and store the corresponding indices in registers AX and BX respectively.



**(b) **

(iv) Write an Assembly Language program to compute (a) the binomial coefficients C(n, k) and Power (X, N) using the recursive definition:

1. binomial coefficients C(n, k)

BinomialCoefficients PROC n:DWORD, k:DWORD

CMP k, 0

JE Base

MOV ecx, n

CMP ecx, k

JE Base

DEC n

INVOKE BinomialCoefficients, n, k ; C(n - 1, k)

DEC k

INVOKE BinomialCoefficients, n, k ; C(n - 1, k - 1)

ret

Base: INC eax

ret

BinomialCoefficients ENDP

main PROC

MOV eax, 0

INVOKE BinomialCoefficients, 5, 2

CALL WriteDec

exit

main ENDP

1. Power (X, N)

int Power(int X, int N) {

        if( N == 0 ) return 1;

        else return **Power( X, N-1)** \* X;

}

void main(void) {

        cout <<**Power(5,2)**;

}

Power PROC x:DWORD, n:DWORD

CMP n, 0

JE Base

DEC n

INVOKE Power, x, n

MUL X

Base: ret

Power ENDP

main PROC

MOV eax, 1

MOV edx, 0

INVOKE Power, 5, 2

CALL WriteDec

exit

main ENDP

1. Write an Assembly Language program to find the nth term Fibonacci Sequence:

|  |  |  |
| --- | --- | --- |
| 01 | int fibonacci(int n) | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | if(n==0) return0; | |
| 04 | else |

|  |  |
| --- | --- |
| 05 | if(n==1) return1; |
| 06 | elsereturnfibonacci(n - 1) + fibonacci(n - 2); | |

|  |  |  |
| --- | --- | --- |
| 07 | } | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | int main() | |
| 10 | { |

|  |  |
| --- | --- |
| 11 | int input; |
| 12 | cin >> input; | |

|  |  |  |
| --- | --- | --- |
| 13 | cout << fibonacci(input) << endl; | |
| 14 | } |

Fibonacci PROC n:DWORD

CMP n, 0

JE Base

CMP n, 1

JE Base

DEC n

INVOKE Fibonacci, n

DEC n

INVOKE Fibonacci, n

ret

Base: ADD eax, n

ret

Fibonacci ENDP

main PROC

CALL ReadInt

MOV ebx, eax ; ebx = Input

MOV eax, 0

INVOKE Fibonacci, ebx

CALL WriteDec

CALL CRLF

exit

main ENDP

(vi) **EXCHANGE SORT**

The exchange sort is similar to its cousin, the bubble sort, in that it compares elements of the array and swaps those that are not in their proper positions.  (Some people refer to the "exchange sort" as a "bubble sort".)  The difference between these two sorts is the manner in which they compare the elements. The exchange sort compares the first element with each following element of the array, making any necessary swaps.

for (i = 0; i < n-1; i++)

for (j = 0; j < n-i-1; j++)

if (a[j] > a[j+1])

{

t = a[j];

a[j] = a[j+1];

a[j+1] = t;

}

Write an assembly Language program to sort the elements using exchange sort.

Text

Description automatically generated

**(vii) SELECTION SORT**

Selection sort carries out a sequence of passes over the table. At the first pass an entry is selected on some criteria and placed in the correct position in the table. The possible criteria for selecting an element are to pick the smallest or pick the largest. If the smallest is chosen then, for sorting in ascending order, the correct position to put it is at the beginning of the table. Now that the correct entry is in the first place in the table the process is repeated on the remaining entries. Once this has been repeated *n*-1 times the *n*-1 smallest entries are in the first *n*-1 places which leaves the largest element in the last place. Thus only *n*-1 passes are required. The algorithm can be described as follows:

for (i = 0; i < n-1; i++)

{

// find smallest entry in ith to n-1 th place

// p is subscript of smallest entry yet found

p = i;

for (j = i+1; j < n; j++)

if (a[j]<a[p])

p = j;

// exchange pth element with ith element

t = a[p];

a[p] = a[i];

a[i] = t;

}

For intimation, you can visit the below link:

Write an assembly Language program to sort all the elements using Selection sort.

mov esi, offset data ; Get the address of the data array

mov ecx, LENGTHOF data ; Get the number of elements in the array

outerLoop:

mov ebx, 0 ; Initialize the minimum index to 0

mov eax, [esi + ebx \* 4] ; Get the first element in the array

innerLoop:

cmp ebx, ecx ; Check if we've reached the end of the array

je outerLoopDone ; If so, we're done with the outer loop

; Find the minimum element in the array

mov edx, [esi + ebx \* 4] ; Get the current element

cmp edx, eax ; Compare it to the current minimum

jl updateMinimum ; If it's smaller, update the minimum

inc ebx ; Move to the next element in the array

jmp innerLoop ; Continue the inner loop

updateMinimum:

mov eax, edx ; Update the minimum element

inc ebx ; Move to the next element in the array

jmp innerLoop ; Continue the inner loop

outerLoopDone:

; Swap the minimum element with the first element in the array

xchg eax, [esi] ; Swap the elements

add esi, 4 ; Move to the next element in the array

dec ecx ; Decrement the loop counter

jmp outerLoop ; Continue the outer loop

**Part III**

Q. No. 4 Assembly Language + MIPS [9x5= 45 Points]

(i) Suppose the following data is received from a wireless sensor node operating in a smart building and is stored in EAX register, as shown in Figure 1. You are required to write an assembly language program in (a) Intel IA 32 and (b) in MIPS assembly with the corresponding data definition directives that would extract the data items and store them at memory locations Sequence\_Number, Revision\_Count, Status, and Sensor\_Data.

1. Bits 0 to 11 reflect an integer Sequence\_Number of the packet being sent.
2. Bits 12 – 14 show an integer Revision\_Count of the packet.
3. Bit 15 is the Status of the sensor flag (0 – Forwarded Data and 1 – Sensed Data)
4. Bits 16 – 31 contain the Sensor\_Data.

|  |  |  |  |
| --- | --- | --- | --- |
| 16 bits | 1 bit | 3 bits | 12 bits |
| Sensor\_Data | Status | Revision\_  Count | Sequence\_Number |
|  |  |  |  |

Figure: 1 .data

Sequence\_Number WORD 0

Revision\_Count BYTE 0

Status BYTE 0

Sensor\_Data WORD 0

.code

main PROC

MOV Sequence\_Number, ax

AND Sequence\_Number, 0000111111111111b

SHR eax, 12

MOV Revision\_Count, al

AND Revision\_Count, 00000111b

SHR eax, 3

MOV Status, al

AND Status, 00000001b

SHR eax, 1

MOV Sensor\_Data, ax

exit

main endp

END main

1. Using shift and add instructions multiply a decimal number X10 by 2310. Assume that the result does not exceed the range of a16-bit register. The Intel IA 32 and MIPS version of this program is required.

mul23 PROC USES ebx ecx

MOV cx, ax

SHL ax, 4

MOV bx, cx

SHL bx, 2

ADD ax, bx

MOV bx, cx

SHL bx, 1

ADD ax, bx

ADD ax, cx

ret

mul23 ENDP

1. Give the contents of the following registers, along with the run-time stack, when the following instructions are executed. Initially, consider ESP = 00001FF8h.

Note: SOLVE THIS PART HERE. No Marks will be awarded without proper working using the stack diagrams.

X1 DWORD 25H

X2 DWORD 27H

MAIN PROC

PUSH 6H

PUSH 5H

CALL P1

11500000H MOV RESULT, EAX ; ESP: **\_\_00001FF8h\_**

MAIN ENDP

P1 PROC

115000A4H PUSH EBP

MOV EBP, ESP ; EBP: **\_\_\_\_1FE2h\_\_\_**

|  |
| --- |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

MOV EAX, [EBP+8]

ADD EAX, [EBP+12] ; EAX: **\_\_\_\_0Bh\_\_\_\_\_**

PUSH OFFSET X1

PUSH OFFSET X2 ; ESP: **\_\_\_1FDAh\_\_\_\_**

|  |
| --- |
| ? (OFFSET X2) |
| ? (OFFSET X1) |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

POP ESI

POP EBX

ADD [ESI], EAX ; X2: **\_\_\_\_32h\_\_\_\_\_**

ADD [EBX], EAX ; X1: **\_\_\_\_30h\_\_\_\_\_**

|  |
| --- |
| ? (Value of EBP at 115000A4h) |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

MOV ESP, EBP

POP EBP

|  |
| --- |
| 11500000h (Return address of P1) |
| 5h |
| 6h |

RET 8 ; EIP: **\_\_11500000h\_**

***; Stack empty***

P1 ENDP

1. Write an assembly language program to copy the characters of a string to a target string. The characters are stored in such a way that only a single instance of any character in the string is stored. Initialize a source string to: "This is the source string".

.data

source BYTE "This is the source string", 0

target BYTE LENGTHOF source DUP(0)

flags BYTE 32 DUP(0) ; Boolean array of bits to mark which characters have been copied

.code

copy PROC USES eax ecx esi edi

CLD ; Clear the direction flag for MOVSB

MOV eax, 0 ; Clear eax (its lowest bit is used for indexing)

Top: CMP BYTE PTR [esi], 0

JE OutB ; Jump out of the loop if the current character is the null terminator

MOV al, [esi] ; Move the current character into al for division

MOV ch, 8

DIV ch ; Divide ax by 8, this places the byte-index of our boolean array into al, and

; the bit-index of the sub-array into ah

MOV cl, ah ; Move ah into cl, so it can be used in shifting to create a bit mask

MOV ch, 1 ; Move 1 into ch, to create a bit mask

SHL ch, cl ; Shift ch left by cl, bl now holds the bit mask

MOV ah, 0 ; Clear ah, so the flags array is indexed only by al

TEST flags[eax], ch ; Index into the flags and test the bit corresponding to the current

; character

JNZ NoCopy ; If the zero flag is clear, the current character has already been copied

; once, do not copy it again

OR flags[eax], ch ; Set the bit in the flags corresponding to the current character

MOVSB ; Copy the character and increment both source and target pointers

JMP Top

NoCopy: INC esi ; Increment the source pointer even if no copying occurred

JMP Top

OutB: ret

copy ENDP

main PROC

MOV esi, OFFSET source

MOV edi, OFFSET target

CALL copy

exit

main ENDP

END main

1. Write a recursive procedure to find a value in a large integer array. Ask the user to enter an integer value in the main program. You should pass user supplied value as parameter to the recursive function using the INVOKE directive. Also, draw labeled diagrams to show stack values at each iteration of this recursive function.

Text

Description automatically generated

1. Write an assembly language code to implement the following high-level language code showing the use of LEA instruction and OFFSET assembler directive.

char moon [20];

void star\_array () {

char cell[20];

for (int i=19; i>=0; i--) {

cell[i] = ‘\*’;

moon[i] = ‘x’;

}

}

Text

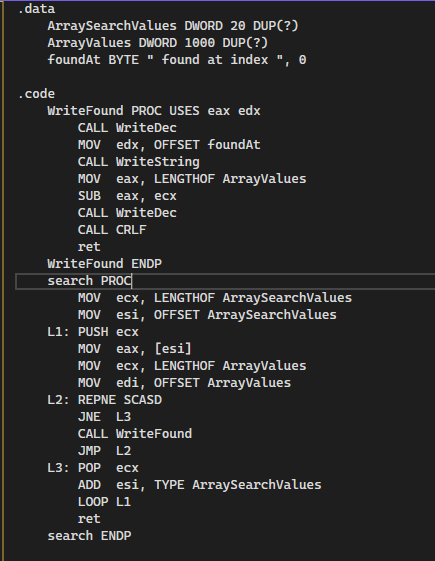
Description automatically generated

1. Write a recursive procedure in x86 assembly language that divides a number by another number and stops when dividend is less than or equal to 5h. Consider dividend = D4A4h and divisor = Ah. The Intel IA 32 and MIPS version of this program is required. In MIPS Assembly you have a choice to use the simple loop-based implementation.

Text

Description automatically generated

1. Using string primitives, write an assembly language program that searches 20 elements of array ArraySearchValues in 1000 un sorted elements of another array ArrayValues.



1. Using string primitives, write a program that converts the string “FAST NATIONAL UNIVERSITY” to its respective ASCII values into a new array. Also, write a procedure to search a particular string SITYA defined in the data directives.

Text

Description automatically generated