**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-3584 Section : BCY-3B 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

First one is: MOV ecx, 858C1200h

Second one is: 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

67 66 89 A4 EA 011

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 = 00000085H

MOV AL,3DH

AND AL, 74H ; b. AL = 00000034H

MOV AL,9BH

OR AL,35H ; c. AL = 000000BFH

MOV AL,72H

XOR AL,0DCH ; d. AL = 000000AEH

(iv) Differentiate between the following Assembly Language instructions:

MOV EAX, OFFSET VAR1

MOV EAX, VAR1

**MOV EAX, OFFSET VAR1:** This stores the reference/address of the memory operand VAR1.

**MOV EAX, VAR1:** This stores the value of the memory operand in the EAX register.

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

1. A stack Is a convenient temporary save area for registers when they are used.
2. When CALL instruction executes the instruction address after the call instruction is stored on the stack to return to the calling function.
3. We can pass arguments to a function using the runtime stack.
4. Stack provided temporary storage for local variables created inside a function.

(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: FCH

(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

ANS) The additional instructions that are generated by the assembler are as follows:

When entering the procedure:

Push ESI

Push ECX

When leaving the procedure:

POP ESI

POP ECX

(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.

(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.

.DATA

|  |  |  |  |
| --- | --- | --- | --- |
|  | AH =1 | AH=2 | AH=3 |
| If V1=V2 then |  |  | √3 |
| If V1<V2 then |  | √2 |  |
| If V1>V2 then |  |  |  |

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: 00001FE2H

|  |
| --- |
| EBP at 1115000A4h |
| 111500000h |
| 5h |
| 6h |

MOV EAX, [EBP+8]

ADD EAX, [EBP+12] ; EAX: 0BH

PUSH OFFSET X1

PUSH OFFSET X2 ; ESP: 00001DAH

|  |
| --- |
| Offset of X2 |
| Offset X1 |
| Value of ebp at 115000A4h |
| 11500000h |
| 5h |
| 6h |

POP ESI

POP EBX

ADD [ESI], EAX ; X2: 32H

ADD [EBX], EAX ; X1: 30H

MOV ESP, EBP

POP EBP

|  |
| --- |
| 11500000H |
| 5H |
| 6H |

RET 8 ; EIP: 11500000H

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 = 40400DH
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

To set the zero flag, one must use arithmetic or logical operation.

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

If the value negated is the smallest value in range it will set the overflow flag.

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 (Invalid. Cannot move 2 values in a register at once)

1. MOV AL, J+2 ;AL = 8
2. MOVSX EAX, L[1] ;EAX = 00006534h
3. MOV EBX, M[2]; EBX = 0003000h
4. INC [ESI] ;ESI = OFFSET J ; (Invalid. INC can only increment value of known file size such as byte, word, dword, etc.)
5. MOV I, L (Invalid. Cannot move memory to memory)
6. MOV EAX, DWORD PTR J (Valid. EAX = 39383736h)
7. MOV L, WORD PTR M (Invalid. Memory to memory move not allowed)
8. MOV ESI, L (Invalid. Cannot move small value in 32 bit register)
9. 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 = 06AH

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 = 0000FFFFH

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

AND eax, 0FFFFF0F0H

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

mov eax, 0

mov ebx, 0

mov ax, 7fffh

mov bx, 8000h

cmp ax, bx

call dumpregs

JL yes

JB yes\_again

;The JB instruction will execute because we are dealing with unsigned comparisons. The cmp instruction generates -1, which sets the SF and OF to 1 which causes the JB instruction to execute.

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

1. Implement the following pseudo-code in assembly language (Intel IA-32) . 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;

}

Ans(a)

.data

op1 SDWORD 10

op2 SDWORD 20

op3 S DWORD 25

x DWORD 0

y DWORD 0

.code

main proc

mov eax, op1 ;op1

mov ebx, op2 ;op2

mov edx, op3 ;op3

top: cmp eax, ebx

JL inc\_cmp

JNL endd

inc\_cmp:

inc eax

cmp edx, ebx

JNE add\_10

add y, 2

push y

pop x

add\_10:

add y, 10

push y

pop x

JMP top

endd:

exit

main endp

      end main

(b)

; All values are

; 32-bit unsigned integers

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

X=10

else

X=20

Ans(b)

.data

val1 DWORD 10

val2 DWORD 20

val3 DWORD 15

x DWORD 0

y DWORD 0

.code

main proc

mov eax, val1

cmp eax, val2

JA L2

mov x, 10

JMP L4

L2:

mov ebx, val3

cmp val2, ebx

JNA L3

mov x, 10

JMP L4

L3:

mov x, 20

L4:

exit

main endp

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 version of this program is required.

.data

x DWORD 14

y DWORD 9

z DWORD 16

.code

main proc

mov eax, x

mov ebx, y

mov edx, z

push eax

push ebx

push edx

call MINIMUM

exit

main endp

MINIMUM PROC

cmp eax, ebx

JAE L2

cmp eax, edx

JAE L2

JMP L4

L2:

cmp ebx, edx

JAE L3

mov eax, ebx

JMP L4

L3:

mov eax, edx ;store result here

L4:

ret

MINIMUM ENDP

end main

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.
3. TableCopy PROC uses ESI EDI ECX

mov esi, Table1

mov edi, Table2

mov ecx, LENGTHOF table1

CLD

REP movsb

ret

TableCopy ENDP

1. Search PROC USES ecx esi eax

MOV ecx, [esp + 16]

MOV esi, [esp + 20]

MOV eax, [esp + 24]

MOV edi, -1

L1:

INC edi

CMP al, [esi + edi]

LOOPNZ L1

JZ L2

INC edi

L2:

ret 12

Search ENDP

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

1. binomial coefficients C(n, k)

BinomCoeff PROC, n:DWORD, k:DWORD

cmp k, 0

JE L4

mov ecx, n

cmp ecx, k

JE L4

dec n

INVOKE BinomCoeff, n, k

dec k

INVOKE BinomCoeff, n, k

ret

L4:

inc eax

ret

BinomCoeff 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)**;

}

main PROC

mov eax, 1

mov edx, 0

invoke power, 5, 2

call writeDec

exit

main endp

Power PROC, x:dword, n:dword

cmp n, 0

JE L4

DEC n

INVOKE power, x, n

mul x

L4:

ret

Power 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 | } |

FibSer PROC

push ebp

mov ebp, esp

push ebx

sub esp, 24

mov DWORD PTR [ebp-20], edi

cmp DWORD PTR [ebp-20], 0

jne L2

mov eax, 0

jmp L3

L2:

cmp DWORD PTR [ebp-20], 1

jne L4

mov eax, 1

jmp L3

L4:

mov eax, DWORD PTR [ebp-20]

sub eax, 1

mov edi, eax

call FibSer

mov ebx, eax

mov eax, DWORD PTR [ebp-20]

sub eax, 2

mov edi, eax

call FibSer

add eax, ebx

L3:

mov ebx, DWORD PTR [ebp-8]

leave

ret

FibSer 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.

ExSort PROC:

MOV ecx, LENGTHOF array - 1

L1:

MOV esi, 0

MOV edx, LENGTHOF array - 1

L2:

CMP edx, 0

JE L5

MOV eax, array[esi]

ADD esi, TYPE array

CMP eax, array[esi]

JB L3

MOV ebx, array[esi]

MOV array[esi], eax

SUB esi, TYPE array

MOV array[esi], ebx

DEC edx

ADD esi, TYPE array

JMP L2

L3:

DEC edx

JMP L2

L5:

LOOP L1

MOV esi, OFFSET array

MOV ecx, LENGTHOF array

MOV ebx, TYPE array

ret

ExSort ENDP

**(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.

selecSort PROC

MOV ecx, LENGTHOF array - 1

L1:

MOV esi, i

MOV edi, i

MOV j, esi

MOV edx, LENGTHOF array

L2:

CMP edx, esi

JE L4

MOV edi, j

MOV eax, array[edi \* TYPE array]

CMP array[esi \* TYPE array], eax

JAE L3

MOV j, esi

MOV edi, esi

L3:

INC esi

JMP L2

L4:

MOV esi, i

PUSH array[edi \* TYPE array]

PUSH array[esi \* TYPE array]

POP array[edi \* TYPE array]

POP array[esi \* TYPE array]

INC i

LOOP L1

MOV esi, OFFSET array

MOV ecx, LENGTHOF array

MOV ebx, TYPE array

ret

selecSort ENDP

**Part III**

Q. No. 4 Assembly Language [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 Intel IA 32 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

.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 version of this program is required.

MOV ecx, eax

SHL eax, 4

MOV ebx, ecx

SHL ebx, 2

ADD eax, ebx

MOV ebx, ecx

SHL ebx, 1

ADD eax, ebx

ADD eax, ecx

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: 00001FE2H

|  |
| --- |
| EBP ar 111500A4H |
| 11500000H |
| 5h |
| 6h |

MOV EAX, [EBP+8]

ADD EAX, [EBP+12] ; EAX: 0000000BH

PUSH OFFSET X1

PUSH OFFSET X2 ; ESP: 00001FDAH

|  |
| --- |
| OFFSET X2 |
| OFFSET X1 |
| EBP at 115000A4H |
| 11500000H |
| 5H |
| 6H |

POP ESI

POP EBX

ADD [ESI], EAX ; X2: 32H

ADD [EBX], EAX ; X1: 30H

|  |
| --- |
| Value at EBP 115000A4H |
| 11500000H |
| 5H |
| 6H |

MOV ESP, EBP

POP EBP

|  |
| --- |
| 11500000H |
| 5H |
| 6H |

RET 8 ; EIP: 11500000H

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".
2. 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.

Search Proto, val:DWORD, index:DWORD

writefound PROTO, val:DWORD, index:DWORD

.data

arrD DWORD 500 DUP(?) ;A large integer array, we assume there is some value here

found BYTE " found at index ", 0

prompt EQU "Enter a value: ", 0

ask BYTE prompt

.code

main PROC

MOV edx, OFFSET ask

CALL WriteString

CALL ReadDec

INVOKE search, eax, 0

exit

main ENDP

search PROC, val:DWORD, index:DWORD

MOV eax, index

CMP eax, LENGTHOF arrd

JE L4

MOV eax, val

MOV esi, index

CMP arrD[esi \* TYPE arrD], eax

JNE L3

INVOKE WriteFound, val, index

L3:

INC index

INVOKE search, val, index

L4:

ret

search ENDP

WriteFound PROC USES eax, val:DWORD, index:DWORD

MOV eax, val

CALL WriteDec

MOV edx, OFFSET found

CALL WriteString

MOV eax, index

CALL WriteDec

CALL CRLF

ret

WriteFound ENDP

end main

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’;

}

}

.data

moon BYTE 20 DUP('0'), 0

.code

star\_array PROC

ENTER 20, 0

LEA esi, [ebp + 20]

MOV edi, OFFSET moon

MOV ecx, 20

L1:

MOV BYTE PTR [esi + ecx - 1], '\*'

MOV BYTE PTR [edi + ecx - 1], 'x'

LOOP L1

LEAVE

ret

star\_array ENDP

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 version of this program is required.

DivTill5 PROTO, n:DWORD, divi:DWORD

.code

main PROC

mov eax, 0D4A4H

mov ecx, 0Ah ;our divisor

INVOKE DivTill5, eax, ecx

exit

main endp

DivTill5 PROC, n:DWORD, divi:DWORD

CMP n, 05h

JBE L4

mov eax, n

mov edx, 0

mov ecx, divi

div ecx

mov n, eax

INVOKE DivTill5, n, divi

L4:

ret

DivTill5 ENDP

end main

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.

.data

ArraySearchValues DWORD 20 DUP(?)

ArrayValues DWORD 500 DUP(?)

.code

search PROC

MOV ecx, LENGTHOF ArraySearchValues

MOV esi, OFFSET ArraySearchValues

L1:

PUSH ecx

MOV eax, [esi]

MOV ecx, LENGTHOF ArrayValues

MOV edi, OFFSET ArrayValues

L2:

REPNE SCASD

JNE L3

CALL WriteFound

JMP L2

L3:

POP ecx

ADD esi, TYPE ArraySearchValues

LOOP L1

ret

search ENDP

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.

For ASCII values

.data

str1 BYTE "FAST NATIONAL UNIVERSITY",0

ascii BYTE LENGTHOF str1 DUP(0)

.code

toASCII PROC

MOV esi, OFFSET str1

MOV edi, OFFSET ascii

MOV ecx, LENGTHOF str1

CLD

REP MOVSB

MOV esi, OFFSET ascii

MOV ecx, LENGTHOF ascii

MOV ebx, TYPE ascii

ret

toASCII ENDP

For “SITYA”

.data

str1 BYTE "FAST NATIONAL UNIVERSITY",0

toFind BYTE "SITYA",0

replace BYTE 15 DUP(0)

.code

searchSITYA PROC

MOV al, 'S'

MOV edi, OFFSET replace

MOV ecx, LENGTHOF replace

CLD

REPNE SCASB

JNZ L4

DEC edi

MOV esi, OFFSET toFind

REPE CMPSB

L4:

ret

searchSITYA ENDP