FAST- National University of Computer & Emerging Sciences, Karachi.
FAST School of Computing,
Final Examinations, Fall 2022.
26th December, 2022, 12:00 pm – 3:00 pm

Course Code: EE 2003	**Course Name:** Computer Organization and Assembly Language
Instructor: Dr. Nouman M Durrani, Syed Asim Mehmood, Kashan Hussain, Atiya Jokhio, Rabia Ahmed, Aashir Mahboob, Aamir Ali, Zakir Hussain, Rukhsar Ali	

Instructions:

- The Student must fill in the Below Student Roll No. and other information before attempting the paper.
- Attempt **Part-I** on the **question paper**.
- There are **02 Questions** on **04 pages** in **Part-I**.
- Where asked for values, the student must provide value in **hex-decimal** notation, answer will be considered wrong otherwise.
- Values are provided with their radixes, consider decimal otherwise.

Estimated Time: 70 minutes **Max. Points**: 25

Student Roll No: _____ Section: _____ Student Signature: _____

Part – I

Q. No. 1 Encircle only the best option for the following set of questions. **[CLO: 2] [5 X 1 = 5 Points]**

(i) Support the best statement about LOOPZ:
   a. LOOPZ instruction executes a block of code repeatedly by checking whether ECX is greater than zero and that Zero Flag is CLEAR.
   b. LOOPZ instruction executes a block of code repeatedly by checking whether ECX is greater than zero and that Zero Flag is SET.
   c. LOOPZ executes a block of code repeatedly by checking only whether ECX is equal to zero
   d. LOOPZ instruction executes a block of code repeatedly by checking only whether Zero Flag is SET

(ii) Identify the WRONG statement when discussing Stack:
   a. Every PUSH operation decrements the ESP.
   b. The Value of ESP is incremented when a POP operation is performed.
   c. The variables declared in procedures are stored onto the stack.
   d. The variables declared in the .data segment are stored onto the stack

(iii) RET Instruction is used for returning from a subroutine. What is true about RET 8:
   a. It Pops top of the stack into the instruction pointer EIP and 8 bytes are added to the stack pointer (ESP).
   b. It Pops top of the stack into the instruction pointer EIP and 8 bytes are subtracted from the stack pointer (ESP).
   c. It Pushes EIP onto the stack and 8 bytes are subtracted from the stack pointer (ESP).
   d. It adds 8 bytes to the stack pointer (ESP) and Pops top of the stack into the instruction pointer EIP

(iv) Given that 88C3h is a machine language instruction, support the best statement among the following:
   a. There is one operand in memory without displacement.
   b. There is one operand in memory with 16-bits (or more) displacement.

c. There is one operand in memory with 8-bits displacement.
d. There is no operand in memory.

(v) The situation in MIPS architecture, where the operands are not available is called _____.

a. Data hazard b. Control hazard
c. Stall d. Structural hazard

Q. No. 2 Briefly answer all the questions in the provided space: **[CLO: 2]**
[20 x 1 = 20 Points]

(i) Given the following array ARR1, write an x86 assembly instruction to read the third element of ARR1 into a 16-bit register.

```
ARR1   DWORD 02H, 01H, 03H, 3 DUP (010H), 100H;
```

MOV AX, ARR1 [2 * TYPE ARR1]

(ii) Elaborate through an example, how does SCASW differ from MOVSW?

SCASW → Compare [EDI] with Ax
MOVSW → Copies data from [ESI] to [EDI]

(iii) Consider the following code snippet. What is the value of AX after execution of the following instructions?

```
MOV   AX, 90F8h
MOV   CX, 08h
L1:   SHRD AX, CX, 1
JC    L2
LOOP  L1
L2:   RET
```

AX = A90F h

(iv) The purpose of CLD and STD instructions to use with the string primitive instruction?

CLD → Forward direction
STD → Reverse Direction

(v) The DATE variable of a file directory entry uses bits 0 to 3 for the DAY, bits 4 to 7 for the MONTH, and bits 8 to 15 for the YEAR. Write instructions to copy the MONTH to a word variable wMONTH.

```
MOV AX, DATE
SHL AX, 8
SHR AX , 12
MOV wMONTH, AX
```

(vi) What will be the final value of AX after the following code is executed?

```
MOV   BL, 12
MOV   AX, 98
DIV   BL
```

AX = 0208 h

(vii) Consider a two-byte x86 machine code as B8 C5. How would you determine the size of register(s) used in this instruction?

As W = 0, size of register used in this is 8-bit

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

JB assume both operands are unsigned and works only on Carry Flag (CF). JL JB assume both operands are signed and works only on Carry Flag (CF), and Overflow Flag (OF).

(ix) Suppose AX = 5678h, BX=0ABCDh, CX=3412h, and DX=0EFCDh , give the contents of AX, BX, CX, and DX after the execution of following instructions have been completed:

```
SHL     AX, 2
PUSH    AX
ROL     CX, 2
PUSH    CX
RCL     BX, 1
PUSH    BX
XCHG    DH, DL
PUSH    DX
POP     BX      ; BX = CDEF h
POP     CX      ; CX = 579A h
POP     DX      ; DX = D048 h
POP     AX      ; AX = 59E0 h
```

(x) Draw a labeled stack after a call instruction with two parameters (assume 32-bit values). Assume EAX=00120020H, EBX=000BAC00H, at start of main proc ESP = 00AB45D1 H

```
MAIN PROC
PUSH EAX
 PUSH EBX
 CALL STACKVALUES
MAIN ENDP
STACKVALUES PROC
PUSH EBP
MOV EBP, ESP
MOV EAX,[EBP + 12]
SUB EAX,[EBP + 8]
POP EBP
RET
STACKVALUES ENDP
```

Address	Value
00AB45C5 H	EBP
00AB45C9 H	RET VALUE
00AB45CD H	**000BAC00 H**
00AB45D1 H	**00120020 H**

(xi) Discuss the difference between the two basic modes of operation of x86 processors: the Real addressing mode and the Protected addressing mode.

Real Mode	Protected Mode
• In this mode the processor 8088 / 8086. • This mode has only 1 MB memory addressing mode. • This mode handles only one task at a time. • In this mode translation is not required. • In this mode processor or computer directly communicate with ports and devices. • This mode is not supporting memory management mode.	• In this processor sworks in full capacity. • This mode has more than 1 MB to few GB memory addressing capacity. • This mode handles multiple tasks at a time. • In this mode memory address translation is required. • In this processor communicates with ports and devices through OS • This mode supports memory management.

(xii) When do you typically use the CBW and CWD instructions?

Before performing IDIV instruction for sign extension

(xiii) Where indicated, write down the values of Carry, Sign, Zero and Overflow flags after each instruction has executed:

```
MOV AX, 7FFCH
ADD AL, 1FH          ; CF = 1   SF = 0 ZF = 0 OF = 0
ADD AH, 1            ; CF = 0 SF = 1 ZF = 0 OF = 1
```

(xiv) Differentiate between the following Assembly Language instructions:

```
MOV EAX, OFFSET X  ; _Address of X is moved to EAX
MOV EAX, X         ; _ Value of X is moved to EAX
```

(xv) When does RECURSION is preferred over LOOPING? Give some real world example.

Recursion has more expressive power than iterative looping constructs. Real world example is factorials.

(xvi) Elaborate the difference between MOVZX and MOVSX instructions through an example.

MOVZX → is used for ZERO extension

MOVSX → is used for sign extension

(xvii) Consider the following declaration of a binary search tree data structure.

```
struct TREE {
int data;
struct TREE* left;
struct TREE* right;
};
```

Describe the layout of this structure on an x86 32-bit architecture. If "TREEOFFSET" is the address of the beginning of the structure, then **TREEOFFSET** is the offset of the data field, **TREEOFFSET+4** is the offset of the left field, and **TREEOFFSET+8** is the offset of right field.

(xviii) Write an assembly language program that adds two 64-bit numbers. Assume a 32-bit architecture.

```
.data
X QWORD ?
Y QWORD ?
Z QWORD ?
.CODE
MOV EAX, DWORD PTR X
MOV EBX, DWORD PTR Y
ADD EAX, EBX
MOV DWORD PTR Z, EAX
MOV EAX, DWORD PTR [X+4]
```

MOV EBX, DWORD PTR [Y+4]

ADC EAX, EBX

MOV DWORD PTR [Z+4], EAX

(xix) Write a valid x86 assembly INVOKE instruction for the following sample function:

int Factor(int* arr[], int num, int* x)

INVOKE Factor, ADDR arr, num, ADDR x

(xx) Write some key differences between CISC and RISC architectures.

CISC stands for complex instruction set computer. The CISC approach attempts to minimize the number of instructions per program.

RISC stands for Reduced Instruction set computer. The RISC reduces the cycles per instruction at cost of number of instructions per program

FAST- National University of Computer & Emerging Sciences, Karachi.
FAST School of Computing,
Final Examinations, Fall 2022.
26th December, 2022, 12:00 pm – 3:00 pm

Course Code: EE 2003	Course Name: Computer Organization and Assembly Language
Instructors: Dr. Nouman M Durrani, Syed Asim Mehmood, Kashan Hussain, Atiya Jokhio, Rabia Ahmed, Aashir Mahboob, Aamir Ali, Zakir Hussain, Rukhsar Ali	
Student's Roll No: [redacted]	Section: [redacted]

Instructions:

- Attempt all questions. Part II consists of 5 questions (Q3-Q7) on 3 pages.
- Return the question paper along with your answer sheet.
- Read each question completely before answering it.
- All the answers must be solved according to the SEQUENCE given in the question paper, otherwise points will be deducted.
- Where asked for values, only provide the **hex-decimal** values.
- Problems needing iterations should be coded using iterative instructions. No points will be awarded otherwise.

Time Allowed: 105 minutes. **Maximum Points:** 45 points

==========

PART II

Machine Language [10-15 Minutes]: [CLO: 3] [4 + 3 = 7 Points]

Q. No. 3 (a) Convert the following independent Assembly Language instructions to Machine Language code – give your answers in hexadecimal (binary answers will not be graded):

(i) `PUSH BX`

(ii) `MOV AX, CX`

(iii) `MOV [BX][DI]+4567h, DX`

(iv) `Loop L1 ; Displacement = 40h`

(b) Convert the following hexadecimal machine codes to assembly language mnemonics. State what each of the byte fields mean:

(i) `B8 12 20 h`

(ii) `52 h`

(iii) `03 04 h`

Procedures [20-25 Minutes]: [Attempt any Two parts from here] [CLO: 4] [2 x 4 = 08 Points]

Q. No. 4 (a) Write an assembly language procedure MAXIMUM that is called from the MAIN procedure to find the maximum MAX among 10 elements of an array ARRAY1. The arguments are passed by reference to the procedure MAXIMUM using registers. The result is also returned in a register. Also, write the corresponding data definition directives.

(b) Write an Assembly Language Program having a procedure CAPITALCASE to find all capital letters in a string and store only Capital Case Letters in a DWORD array CCASE. In this problem, you are supposed to use recursion performed through stack frames discussed in the class.

For example, if the INPUT string passed is: FAST National University, Karachi

The Expected Output will be: FAST N U K

(c) Write a procedure that receives a string through the stack (i.e., the string pointer is passed to the procedure) and removes all leading and duplicate blank characters in the string. For example, if the input string passed is (␣ indicates a blank character).

␣ ␣ ␣ ␣␣Reading␣ ␣ ␣Assembly␣␣ ␣ ␣ ␣Language.

it will be modified by removing all leading and duplicate blanks as

Reading␣Aeembly␣Language.

Conditional Processing and Integer Arithmetic [20-30 Minutes]: **[CLO: 2]** **[2 x 4 = 8 Points]**

[Attempt any Two parts from here]

Q. No. 5 (a) Data transmission systems and file subsystems often use a form of error detection that relies on calculating the parity (even or odd) of blocks of data. Suppose you are given an integer that requires 32 bits to store. You are asked to find whether its binary representation has an odd or even number of 1's. Write a program to read an integer (should accept both positive and negative numbers) from the user and outputs whether it contains an odd or even number 1's. Your program should also print the number of 1's in the binary representation.

(b) Write an assembly language program with a loop and indexed addressing that calculates the sum of all the gaps between successive array elements. The array elements are DWORDS, sequenced in non-decreasing order. For example, the array {0, 2, 5, 9, 10} has differences of 2, 3, 4, and 1 respectively, whose sum equals 10.

(c) The following data is received from a real-time measuring device and is stored at memory location **Data1**. You are required to write an assembly language program segment with the corresponding data definition directives that would extract the data items and store them in **WORD** type variables **Day, Month, Year, Temperature** and **Pressure.** The Year is relative to 1980.

5 bits	4 bits	7 bits	7 bits	9 bits
Day	Month	Year	Temperature	Pressure

Strings and Arrays [20-25 Minutes]: [Attempt any three parts from here] **[CLO: 2] [3 x 4 = 12 Points]**

Q. No. 6 (a) Using string primitives, write an assembly language program that searches 10 elements of an array ArraySearchValues in 500 unsorted elements of another array ArrayValues.

(b) Suppose EBX points to the following array ewords:

```
ewords byte "each other eager eagerly v2car eagerness foxy eagle-eyed earache", 0
```

Write x86 code snippet to process this array and print the starting offset of each word that is starting with letter other than e.

(c) Write an assembly language program to read a string of characters from the user and prints/store the vowel count. For each vowel, the count includes both uppercase and lowercase letters. For example, the input string

```
"Advanced Programming in UNIX Environment"
```

produces the following output:

Vowel Count

```
a or A 3
e or E 3
i or I 4
0 or O 2
u or U 1
```

(d) Using string primitive instructions, replace each element of a given array by its mathematical square. Assume any valid type for array1.

01	02	03	04	05	06	07	08	09	10

MIPS Assembly and Pipelining [10-15 Minutes]: **[CLO: 1]** **[4 + 6 = 10 Points]**

Q. No. 7 (a) What is pipelining? Discuss the five pipeline stages for the following Instruction:

ADD R_1, R_2, R_3

(b) Consider the following pipeline diagram showing the execution of a series of instructions:

LW R_1, 4(R_9)										
SW R_1, 8(R_9)										
SUB R_4, R_1, R_5										
AND R_4, R_1, R_7										
OR R_8, R_1, R_9										

Answer the following:

i. In a processor implementation, a data hazard can slow down the pipeline. What is a data hazard? Give a short example using MIPS code that illustrates the problem and give a brief explanation of the problem.
ii. What type of hazard may occur in the above pipeline stage diagram? Explain why?
iii. Rewrite the above pipeline diagram with the best possible solution to eliminate the occurrence of hazards.

Instruction	Opcode	Instruction	Opcode	Instruction	Opcode	Instruction	Opcode
mov reg8, imm	1011 0rrr [imm8]	pop reg16	0101 1rrr	and reg16, reg16	0010 00x1 [11-reg-r/m]	Loop	1110 0010 [disp8]
mov reg16, reg16	1000 1001 [11-reg-r/m]	pop mem16	1000 1111 [mod-000-r/m]	and reg8, mem8	0010 0010 [mod-reg-r/m]	Loope Loopz	1110 0001 [disp8]
mov reg16, imm	1011 1rrr [imm16]	push reg16	0101 0rrr	and reg16, mem16	0010 0011 [mod-reg-r/m]	Loopne Loopnz	1110 0000 [disp8]
mov reg8, mem	1000 1010 [mod-reg-r/m]	push reg16	1111 1111 [11-110-r/m]	and mem8, reg8	0010 0000 [mod-reg-r/m]	or mem16, reg16	0000 1001 [mod-reg-r/m]
mov mem, reg16	1000 1001 [mod-reg-r/m]	push mem16	1111 1111 [mod-110-r/m]	and mem16, reg16	0010 0001 [mod-reg-r/m]	or reg16, reg16	0000 10x1 [11-reg-r/m]
mov reg16, mem	1000 1011 [mod-reg-r/m]	and reg8, reg8	0010 00x0 [11-reg-r/m]	and reg16, imm16	1000 00s1 [11-100-r/m] [imm]	or reg16, imm16	1000 00s0 [11-001-r/m] [imm]
add reg, R/m	0000 00dw [mod-reg-r/m]	add reg, imm	10000 00sw [mod-000-reg]	sub reg, R/m	0001 01dw [mod-reg-r/m]	inc mem	1111 111w [mod-000-r/m]

Mod=11			Effective Address Calculation			
R/M	W=0	W=1	R/M	Mod= 00	Mod= 01	Mod= 10
000	AL	AX	000	[Bx] + [SI]	[Bx] + [SI] + D_8	[Bx] + [SI] + D_{16}
001	CL	CX	001	[BX] + [DI]	[BX] + [DI] + D_8	[BX] + [DI] + D_{16}
010	DL	DX	010	[BP] + [SI]	[BP] + [SI] + D_8	[BP] + [SI] + D_{16}
011	BL	BX	011	[BP] + [DI]	[BP] + [DI] + D_8	[BP] + [DI] + D_{16}
100	AH	SP	100	[SI]	[SI] + D_8	[SI] + D_{16}
101	CH	BP	101	[DI]	[DI] + D_8	[DI] + D_{16}
110	DH	SI	110	Direct Address	[BP] + D_8	[BP] + D_{16}
111	BH	DI	111	[BX]	[BX] + D_8	[BX] + D_{16}

STAY BRIGHT

Q #03
A:
(i) PUSH BX:
53h
(ii) MOV AX, CX:
89 C8h
(iii) MOV [BX][DI]+4567h, DX:
89 91 67 45h
(iv) Loop L1:
E2 40h
B:
(i) B8 12 20 h:
MOV AX, 2012h.
(ii) 52 h:
PUSH DX.
(iii) 03 04 h:
ADD AX, [ESI]

Q #04:
A:

```
MAXIMUM PROC
      MOV AX, [BX]
      MOV MAX, AX
      ADD BX,
      L1: CMP CX, o
            JE END
            MOV AX, [BX]
            CMP AX, MAX
            JG UPDATE
            ADD BX, 2
            LOOP L1

    UPDATE: MOV MAX, AX
                ADD BX, 2
                LOOP L1

      END:
MAXIMUM ENDP

; Data definition directives
ARRAY1 DW 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
MAX DB ?
; Call the MAXIMUM procedure to find the maximum element in ARRAY1
MOV BX, OFFSET ARRAY1
MOV CX, 10
CALL MAXIMUM
; The maximum element is returned in the MAX variable
```

OR

```
ARRAY1     dw     1, 2, 3, 4, 5, 6, 7, 8, 9, 10
MAX        dw     ?
MAXIMUM    proc
            mov     cx, 10
            mov     bx, offset ARRAY1
            mov     ax, [bx]
            mov     MAX, ax
```

```
L1:         add     bx, 2
            cmp     ax, [bx]
            jg      L2
            mov     ax, [bx]
            mov     MAX, ax
            jmp     L1
L2:         mov     ax, MAX
            ret
MAXIMUM    endp

; MAIN procedure
MAIN        proc
            call    MAXIMUM
            mov     ah, 09h
            mov     dx, offset MAX
            int     21h
            ret
MAIN        endp
B:
CAPITALCASE PROC
of the string
      CMP CX, 0
      JE END

      MOV AL, [BX]
      CMP AL, 'A'
      JB NOT_CAPITAL
      CMP AL, 'Z'
      JA NOT_CAPITAL

      MOV [DX], AX
      ADD DX, 2

      NOT_CAPITAL: ADD BX, 1
                   DEC CX
                   PUSH CX
                   PUSH BX
                   PUSH DX
                   CALL CAPITALCASE
                   POP DX
                   POP BX
                   POP CX

      END:
CAPITALCASE ENDP

; Main program
MAIN PROC
      INPUT DB "FAST National University, Karachi", 0
      CCASE DW ?, ?, ?, ?, ?, ?, ?

      MOV BX, OFFSET INPUT
      MOV CX, LENGTHOF INPUT
      MOV DX, OFFSET CCASE
      CALL CAPITALCASE
      END MAIN
```

C:

```
REMOVE_BLANKS PROC
    LOCAL CURR_CHAR:BYTE
    LOCAL PREV_CHAR:BYTE
    LOCAL NEW_LEN:WORD

    MOV BX, [SP]
    MOV SI, BX
    MOV DI, BX

    MOV CURR_CHAR, ' '
    MOV PREV_CHAR, ' '
    MOV NEW_LEN, 0

    L1: CMP BYTE PTR [SI], 0
        JE END
        CMP CURR_CHAR, ' '
        JE SKIP
        CMP PREV_CHAR, ' '
        JNE NO_DUP

        SKIP: MOV PREV_CHAR, CURR_CHAR
              ADD SI, 1
              LOOP L1

        NO_DUP: MOV [DI], CURR_CHAR
                ADD DI, 1
                ADD NEW_LEN, 1
                MOV PREV_CHAR, CURR_CHAR
                ADD SI, 1
                LOOP L1
    END: MOV [SP], DI
         MOV AX, NEW_LEN
REMOVE_BLANKS ENDP
```

Q #05:

A:

```
.MODEL SMALL
.STACK 100H

.DATA
    PROMPT1 DB "Enter an integer: ", 0
    PROMPT2 DB "Number of 1's: ", 0
    PROMPT3 DB "Odd number of 1's", 0
    PROMPT4 DB "Even number of 1's", 0
    INPUT DB ?

.CODE

MAIN PROC
    LOCAL NUM:DWORD
    LOCAL COUNT:DWORD
    LOCAL NEG:DWORD
    LEA DX, PROMPT1
    MOV AH, 09H
    INT 21H
    MOV AH, 0AH
```

```
        LEA DX, INPUT
        INT 21H

        MOV NUM, 0
        MOV COUNT, 0
        MOV NEG, 0
        XOR BL, BL
        MOV BL, INPUT[0]
        CMP BL, '-'
        JNE POSITIVE
        MOV NEG, 1
        XOR BL, BL
        INC BYTE PTR INPUT
        DEC INPUT[0]
        POSITIVE:
            MOV BL, INPUT[0]
            XOR AX, AX
            XOR BH, BH
            XOR AH, AH
            XOR DX, DX
            MOV CL, BL
            LEA SI, INPUT
            MOV AL, '0'
            SUB AL, '0'
            MOV BH, 10
            MUL BH
            ADD NUM, AX
            INC SI
            DEC CL
            LOOP POSITIVE

        CMP NEG, 1
        JNE COUNT1S

B:
.MODEL SMALL
.STACK 100H

.DATA
        ARRAY DW 0, 2, 5, 9, 10
        LENGTH DW 5
        SUM DW ?

.CODE

MAIN PROC
        LOCAL I:DWORD

        MOV SUM, 0

        MOV CX, LENGTH
        MOV DI, OFFSET ARRAY
        MOV SI, DI
        ADD SI, 2
        L1: CMP CX, 0
            JE END
            MOV AX, [SI]
```

```
            SUB AX, [DI]
            ADD SUM, AX
            ADD DI, 2
            ADD SI, 2
            LOOP L1
        END:
MAIN ENDP
C:
.DATA
        Data1 DB ?
        Day DW ?
        Month DW ?
        Year DW ?
        Temperature DW ?
        Pressure DW ?
.CODE

EXTRACT_DATA PROC
        MOV AX, Data1
        AND AX, 1111100000B ; Mask the lower 5 bits
        SHR AX, 5
        MOV Day, AX
        MOV AX, Data1
        AND AX, 11110000B
        SHR AX, 4
        MOV Month, AX

        MOV AX, Data1
        AND AX, 01111111B
        MOV Year, AX
        MOV AX, Data1
        AND AX, 0000000001111111B
        SHR AX, 7
        MOV Temperature, AX
        MOV AX, Data1
        AND AX, 00000000000000011111111B
        SHR AX, 8
        MOV Pressure, AX
EXTRACT_DATA ENDP

Q #06:
A:
SEARCH_ARRAY proc
        push ebp
        mov ebp, esp
        push ebx
        push ecx
        push edx
        push esi
        push edi

        mov ebx, [ebp+8]
        mov ecx, [ebp+12]
        mov edx, [ebp+16]
        mov esi, [ebp+20]
        mov edi, 0
        L1:
```

```
            cmp ecx, 0
            je L2
            mov edi, 0
            mov eax, [ebx]
            mov ebx, ebx + 4
            L3:
                cmp esi, 0
                je L4
                mov edx, [edx]
                cmp edx, eax
                je L5
                mov edx, edx + 4
               dec esi
                jmp L3
            L5:
                inc edi
                jmp L3
            L4:
                add esi, edi
                dec ecx
                jmp L1
      L2:
            mov eax, edi
            pop edi
            pop esi
            pop edx
            pop ecx
            pop ebx
            pop ebp
            ret
SEARCH_ARRAY endp

SEARCH_VALUES_LEN = 10
VALUES_LEN = 500
SEARCH_VALUES BYTE SEARCH_VALUES_LEN DUP(?)
VALUES BYTE VALUES_LEN DUP(?)
B:
mov esi, offset ewords
mov ecx, 0
loop_start:
      mov al, [esi + ecx]
      cmp al, 'e'
      jne print_offset
      inc ecx
    jmp loop_start
print_offset:
      ; print the offset of the current word
      ; (offset is stored in ecx)
      ; ...
      ; increment ecx to move to the next word
      ; ...
      jmp loop_start ; jump back to the start of the loop
```

-> This code snippet will loop through each character in the ewords array, starting with the first character at offset 0. If the character is not 'e', it will print the offset and move on to the next word. If the character is 'e', it will increment the counter and move on to the next character. This process will repeat until the end of the array is reached.

C:

```
input     db       256, 0
count     dw       ?, ?, ?, ?, ?
COUNTVOWELS    proc
                  xor      eax, eax
                  xor      ecx, ecx
                  mov      edx, offset input
L1:            mov      al, [edx]
                  cmp      al, 'a'
                  je       L2
                  cmp      al, 'A'
                  je       L2
                  cmp      al, 'e'
                  je       L2
                  cmp      al, 'E'
                  je       L2
                  cmp      al, 'i'
                  je       L2
                  cmp      al, 'I'
                  je       L2
                  cmp      al, 'o'
                  je       L2
                  cmp      al, 'O'
                  je       L2
                  cmp      al, 'u'
                  je       L2
                  cmp      al, 'U'
                   je       L2
                   jmp      L3
L2:            inc      word ptr count[eax*2]
L3:            inc      edx
                  cmp      byte ptr [edx], 0
                  jne      L1
                  ret
COUNTVOWELS    endp
D:
mov edx, 0
mov esi, offset array1
mov ecx, lengthof array1

L1: mov al, [esi]
      mul al
      mov [esi], al
      inc edx
      inc esi
      dec ecx
      jne L1
```

Q #07:

A:

Pipelining is a technique used in computer architecture to improve the performance of a processor by overlapping the execution of multiple instructions. In a pipelined processor, each instruction is divided into a series of stages, and the stages for multiple instructions are executed concurrently. This allows the processor to perform more than one instruction at a time, increasing the overall throughput of the processor.

The five pipeline stages for the instruction "ADD R1, R2, R3" are:

Instruction fetch: In this stage, the instruction is fetched from memory and loaded into the instruction register.

Instruction decode: In this stage, the instruction is decoded to determine its operation and operands.

Operand fetch: In this stage, the operands (R1, R2, and R3 in this case) are fetched from the register file.

Execution: In this stage, the ADD operation is performed on the operands to produce the result.

Writeback: In this stage, the result is written back to the register file, updating the value of R1.

Each of these stages takes a fixed amount of time to complete, and the pipeline stages for different instructions may be executed concurrently as long as there are no dependencies between the instructions. For example, if the instruction "ADD R1, R2, R3" is followed by the instruction "ADD R4, R5, R1", the operand fetch stage for the second instruction can begin while the execution stage for the first instruction is still in progress, as the result of the first instruction is not needed until the writeback stage. This overlap of execution can significantly increase the performance of the processor.
B:
i. A data hazard is a situation in which an instruction in a pipelined processor depends on the result of a previous instruction that has not yet completed execution. This can slow down the pipeline, as the dependent instruction must wait for the previous instruction to complete before it can execute.

Here is an example of a data hazard using MIPS code:

```
add $t1, $t2, $t3     # Instruction 1
sub $t4, $t1, $t5     # Instruction 2
```

In this example, instruction 2 depends on the result of instruction 1, as it uses the value of $t1 as an operand. If instruction 1 has not yet completed execution when instruction 2 reaches the execution stage of the pipeline, the pipeline must be stalled (paused) until instruction 1 has completed. This can reduce the overall performance of the processor.

ii. In the pipeline stage diagram given, a data hazard may occur between the "SW" instruction and the "SUB" instruction. This is because the "SUB" instruction uses the value of R1 as an operand, and the value of R1 is not updated until the "SW" instruction completes execution and writes the result back to the register file.

iii. One possible solution to eliminate the occurrence of hazards in the above pipeline stage diagram is to rearrange the instructions as follows:

```
SUB R4, R1, R5
LW R1, 4(R9)
AND R4, R1, R7
SW R1, 8(R9)
OR R8, R1, R9
```

By moving the "LW" and "SW" instructions down the pipeline, the "SUB" and "AND" instructions can be completed before the value of R1 is updated, eliminating the hazard. This allows the instructions to be executed more efficiently and improves the overall performance of the processor.