Department of Computer Science
Faculty of Computing
UNIVERSITI TEKNOLOGI MALAYSIA

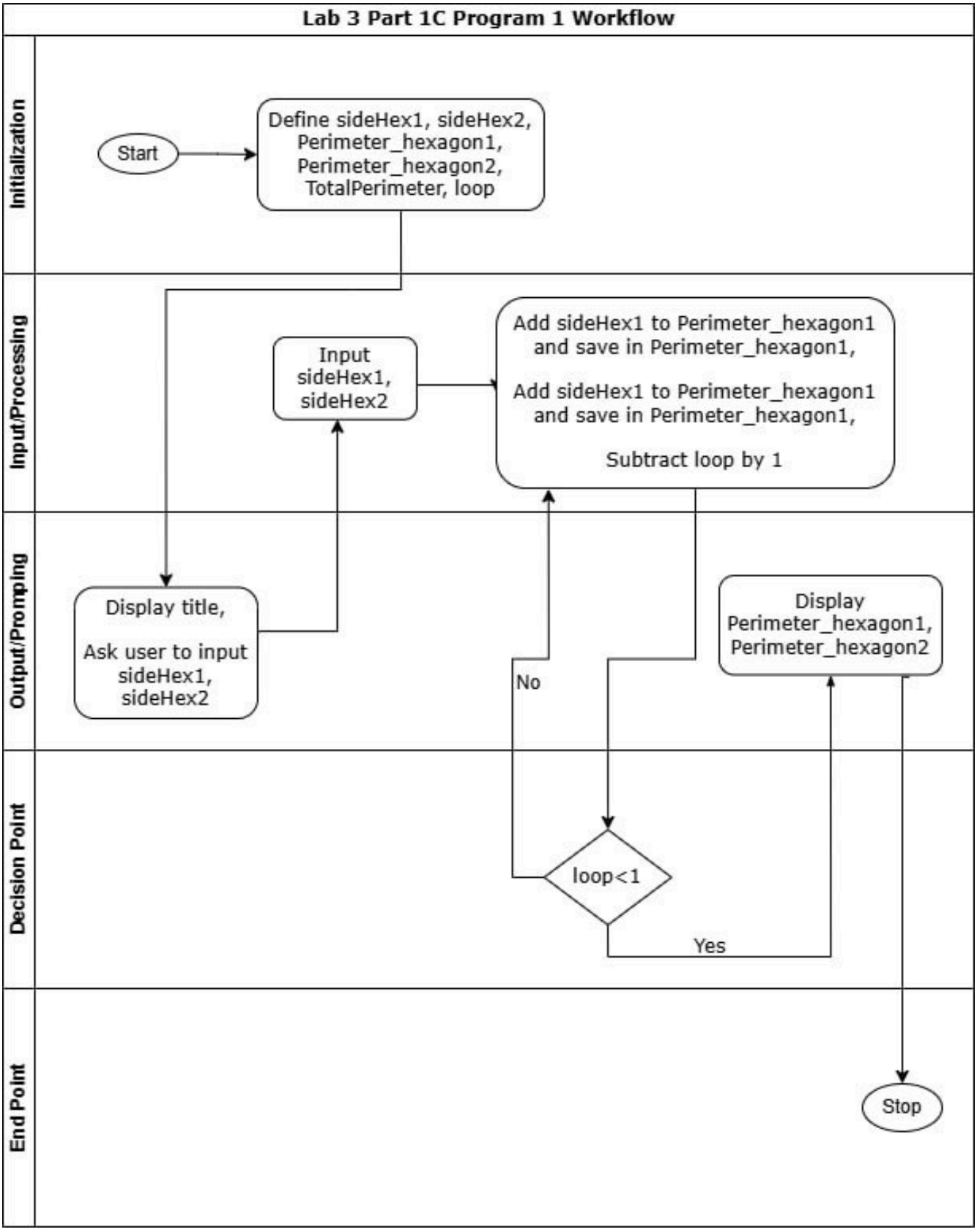
SUBJECT NAME:	**COMPUTER ORGANIZATION AND ARCHITECTURE**
SUBJECT CODE:	**SECR 1033**
SEMESTER:	**2 - 2023/24**
LAB TITLE:	**Programming 3: Interactive Usage of Link Libraries and *Comparison & Conditional Jumps*** ***PART 1: Interactive Usage of Link Libraries*** **PART 2: *Comparison & Conditional Jumps***
INSTRUCTION:	**Complete the lab activities in Part 1C and Part 2C and divide loads among group member equally**

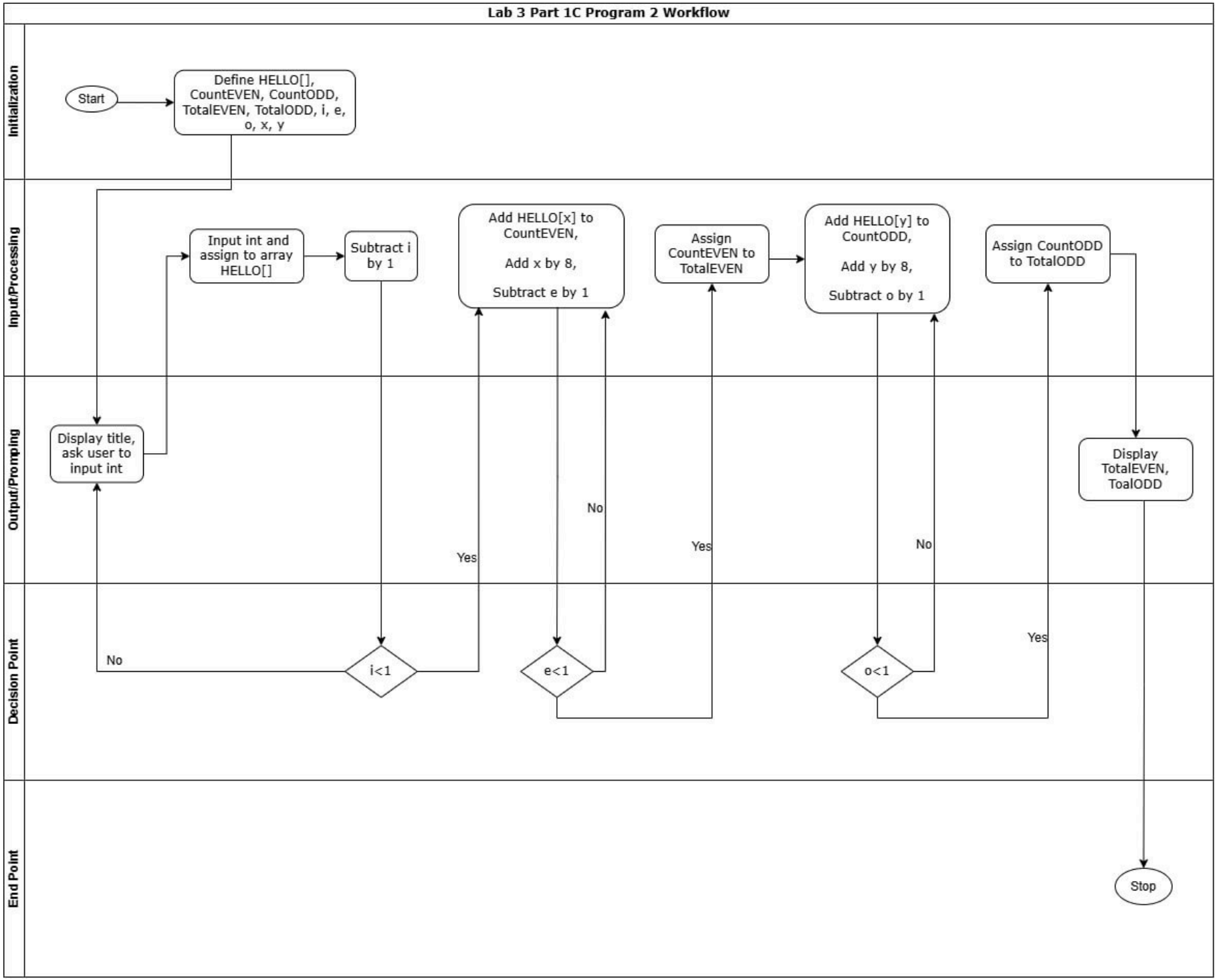
STUDENT INFO :	**Name :**	CHUA JIA LIN
	Metric No :	A23CS0069
	Section:	02
	Email :	chuajialin@graduate.utm.my
STUDENT INFO :	**Name :**	JOANNE CHING YIN XUAN
	Metric No :	A23CS0227
	Section :	02
	Email :	joanneyin@graduate.utm.my

COMMENTS:

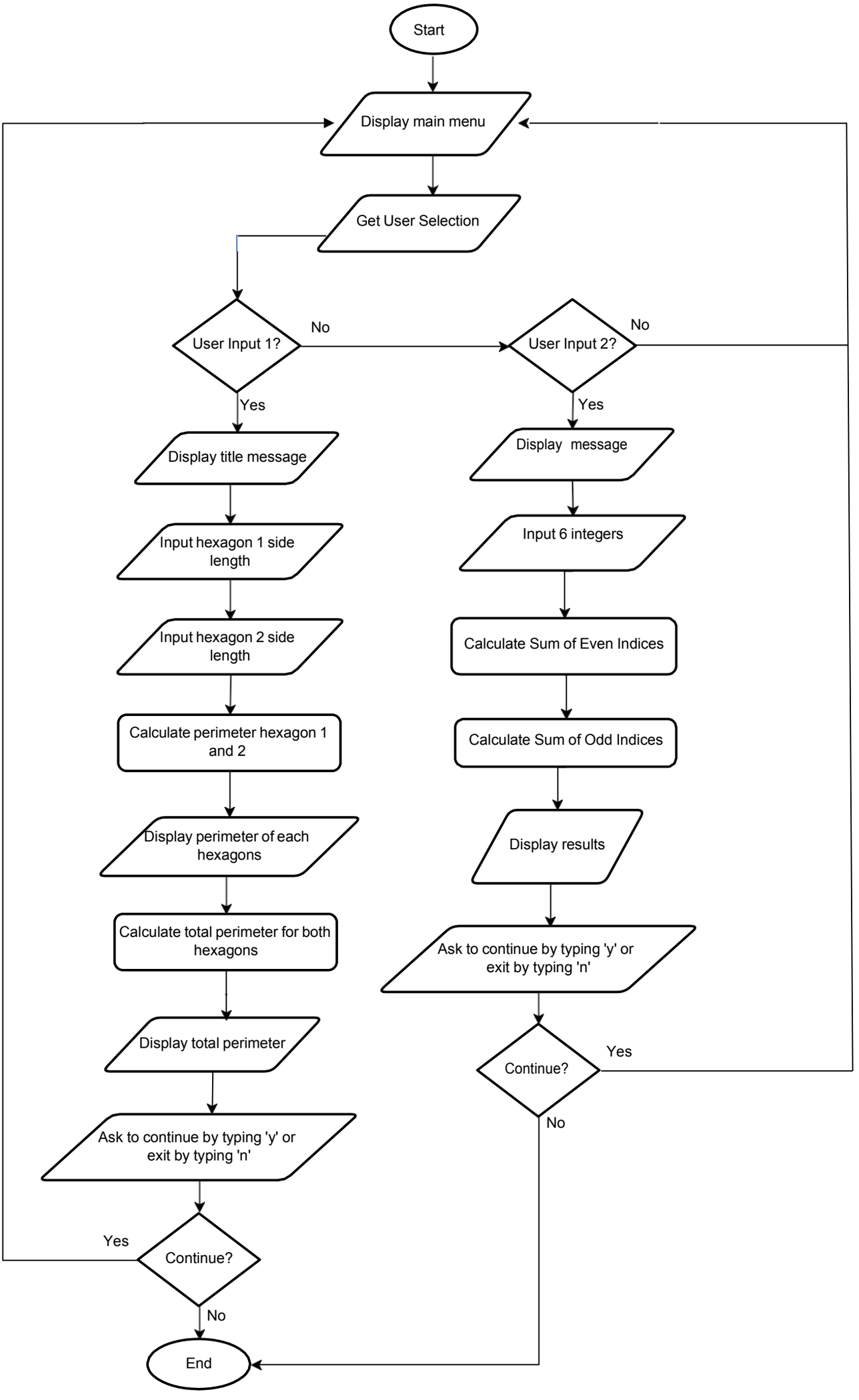
Presentation link : https://youtu.be/JErtYMMjBwU

Prepared by: HKM, MMA, MFR (revised 2024)


Lab 3 Part 1C Program 1 Workflow
Initialization
Input/Processing
Output/Prompting
Decision Point
End Point
Start
Define sideHex1, sideHex2, Perimeter_hexagon1, Perimeter_hexagon2, TotalPerimeter, loop
Input sideHex1, sideHex2
Add sideHex1 to Perimeter_hexagon1 and save in Perimeter_hexagon1,
Add sideHex1 to Perimeter_hexagon1 and save in Perimeter_hexagon1,
Subtract loop by 1
Display title,
Ask user to input sideHex1, sideHex2
Display Perimeter_hexagon1, Perimeter_hexagon2
No
loop<1
Yes
Stop


Lab 3 Part 1C Program 2 Workflow
Initialization
Input/Processing
Output/Prompting
Decision Point
End Point
Start
Define HELLO[], CountEVEN, CountODD, TotalEVEN, TotalODD, i, e, o, x, y
Display title, ask user to input int
Input int and assign to array HELLO[]
Subtract i by 1
i<1
No
Yes
Add HELLO[x] to CountEVEN,
Add x by 8,
Subtract e by 1
e<1
No
Yes
Assign CountEVEN to TotalEVEN
Add HELLO[y] to CountODD,
Add y by 8,
Subtract o by 1
o<1
No
Yes
Assign CountODD to TotalODD
Display TotalEVEN, ToalODD
Stop

Lab 3 Part 2C flowchart



Library Procedures – Overview

Here are some of the procedures available to you. You can find more from the Internet and reference books (Kip Irvine, Assembly Language programming books are a good place to start).

- Clrscr - Clears the console and locates the cursor at the upper left corner.
- Crlf - Writes an end of line sequence to standard output.
- Delay - Pauses the program execution for a specified *n* millisecond interval.
- DumpMem - Writes a block of memory to standard output in hexadecimal.
- DumpRegs - Displays the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP registers in hexadecimal. Also displays the Carry, Sign, Zero, and Overflow flags.
- GetCommandtail - Copies the program's command-line arguments (called the *command tail*) into an array of bytes.
- GetMseconds - Returns the number of milliseconds that have elapsed since midnight.
- Gotoxy - Locates cursor at row and column on the console.
- Random32 - Generates a 32-bit pseudorandom integer in the range 0 to FFFFFFFFh.
- Randomize - Seeds the random number generator.
- RandomRange - Generates a pseudorandom integer within a specified range.
- ReadChar - Reads a single character from standard input.
- ReadHex - Reads a 32-bit hexadecimal integer from standard input, terminated by the Enter key.
- ReadInt - Reads a 32-bit signed decimal integer from standard input, terminated by the Enter key.
- ReadString - Reads a string from standard input, terminated by the Enter key.
- SetTextColor - Sets the foreground and background colors of all subsequent text output to the console.
- WaitMsg - Displays message, waits for Enter key to be pressed.
- WriteBin - Writes an unsigned 32-bit integer to standard output in ASCII binary format.
- WriteChar - Writes a single character to standard output.
- WriteDec - Writes an unsigned 32-bit integer to standard output in decimal format.
- WriteHex - Writes an unsigned 32-bit integer to standard output in hexadecimal format.
- WriteInt - Writes a signed 32-bit integer to standard output in decimal format.
- WriteString - Writes a null-terminated string to standard output.

Reference:

https://csc.csudh.edu/mmccullough/asm/help/

Irvine Library

Part 1B – Let's do a little programming by example

You are given a few examples here. Try them out.

Example 1

Clear the screen, delay the program for **500 milliseconds**, and dump the registers and flags.

```
.code
        call Clrscr mov
        eax,500 call Delay
        call DumpRegs
```

Example 2

Display a null-terminated **string** and move the cursor to the beginning of the next screen line. Attach the output screen capture for this example.

```
.data
str1 BYTE "Assembly language is easy!",0
.code
        mov edx,OFFSET str1 call
        WriteString
        call Crlf
```

Example 3

Display an unsigned integer in binary, decimal, and hexadecimal, each on a separate line. Attach the output screen capture for this example.

```
.data IntVal = 35
.code
        mov eax,IntVal
        call WriteBin ; display binary call
        Crlf
        call WriteDec ; display decimal call
        Crlf
        call WriteHex; display hexadecimal call
        Crlf
```

Example 4

Input a string from the user (*ReadString*). EDX points to the string. Attach the output screen capture for this example. *(**Tips: It is always a good practice to have a string to ask for input)*

```
.data
str2 BYTE "Give me your name: ",0 buffer2
BYTE 21 DUP(0) ; input buffer
.code
        mov edx,OFFSET buffer2 ; point to the buffer mov
        ecx,SIZEOF buffer2 ; specify max characters call
        ReadString ; input the string
```

Example 5

```
mov edx, OFFSET buffer2 ; point to the buffer call
WriteString
call crlf
```

Input a decimal number from the user (*ReadDec*).. The procedure reads a 32- bit unsigned decimal integer from the keyboard and returns the value in EAX. **Output a number** to screen (*WriteDec*). The procedure writes a 32-bit unsigned integer to the console window in decimal format with no leading zeros. Pass the integer in EAX. Attach the output screen capture for this example. *(**Tips: It is always a good practice to have a string to ask for input)*

```
.data
str1 BYTE "Enter a decimal: ",0 val1
dword ?

.code
        mov edx, offset str1 call
        writestring
```

Example 6

Generate and display ten pseudorandom signed integers in the range 0 – 99. Pass each integer to WriteInt in EAX and display it on a separate line. Attach the output screen capture for this example.

```
.code
        mov ecx,10     ; loop counter

L1:     mov  eax,100  ; ceiling value
        call RandomRange   ; generate random int call
        WriteInt           ; display signed int call Crlf
                   ; goto next display line
        loop L1        ; repeat loop
```

Example 6

Reads a 32-bit unsigned decimal integer from standard input, stopping when the Enter key is pressed. All valid digits occurring before a non-numeric character are converted to the integer value. Leading spaces are ignored.

```
.data
decNum    DWORD ?
promptBad BYTE "Invalid input, please enter again",0

.code
read:  call ReadDec
       jnc  goodInput

       mov  edx,OFFSET promptBad
       call WriteString
       jmp  read        ;go input again

goodInput:
           mov  decNum,eax  ;store good value
```

Part 1C – Let's do a little programming on your own

Program 1 : LOOP and ADD

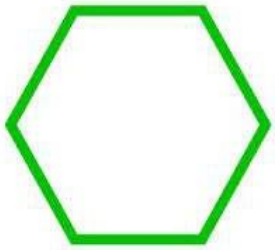


Figure 1 : A hexagon

Figure 1 is illustrates a hexagon figure with same length of side. To calculate the perimeter of the hexagon, the following formula is given.

Perimeter_hexagon1 = side1 + side2 + side3 + side4 + side5 + side6
Perimeter_hexagon2 = side1 + side2 + side3 + side4 + side5 + side6
TotalPerimeter = Perimeter_hexagon1 + Perimeter_hexagon2

Write a complete program using assembly language to calculate the perimeter of TWO different hexagons with different sizes.

In the program, you should do these steps:

i. Get two values from keyboard (32-bit unsigned integer) and save into the variable name ***sideHex1*** for the first hexagon and ***sideHex2*** for the second hexagon.
ii. Calculate both of the perimeters (Example: Perimeter_hexagon1=18 → 3+3+3+3+3+3) by using LOOP and ADD instruction. Save the first result in ***Perimeter_hexagon1*** and the second result in ***Perimeter_hexagon2*** (as 32-bit unsigned integer).
iii. Then, add the two perimeters and save in ***TotalPerimeter*** variable.
iv. Display the output as shown in Figure 2.

```
Calculate Perimeter 2-Hexagon (LOOP and ADD instructions) :

Input Hexagon 1 (side length) : 10
Input Hexagon 2 (side length) : 20

Result of Perimeter Hexagon 1 and 2:
60
120
```

Figure 2: Sample Output

Program 2

Write a program that will **interactively** ask the **user to input the values of 6 integers** in DWORD and you have to put the values into an array name HELLO.

- Example of HELLO array after the user input the values:

1st Value	2nd Value	3rd Value	4th Value	5th Value	6th Value
HELLO[0]	HELLO[4]	HELLO[8]	HELLO[12]	HELLO[16]	HELLO[20]
32	*65*	*77*	*89*	*14*	*54*

- Your CountEVEN will count the value of HELLO[0], HELLO[8] and HELLO[16] and store it in variable name TotalEVEN
- Your CountODD will count the value of HELLO[4], HELLO[12] and HELLO[20] store it in variable name TotalODD
- Lastly, display the value of TotalEVEN and TotalODD
- Display the output as shown in Figure 3.

- **You must use LOOP instruction to do the addition process.**

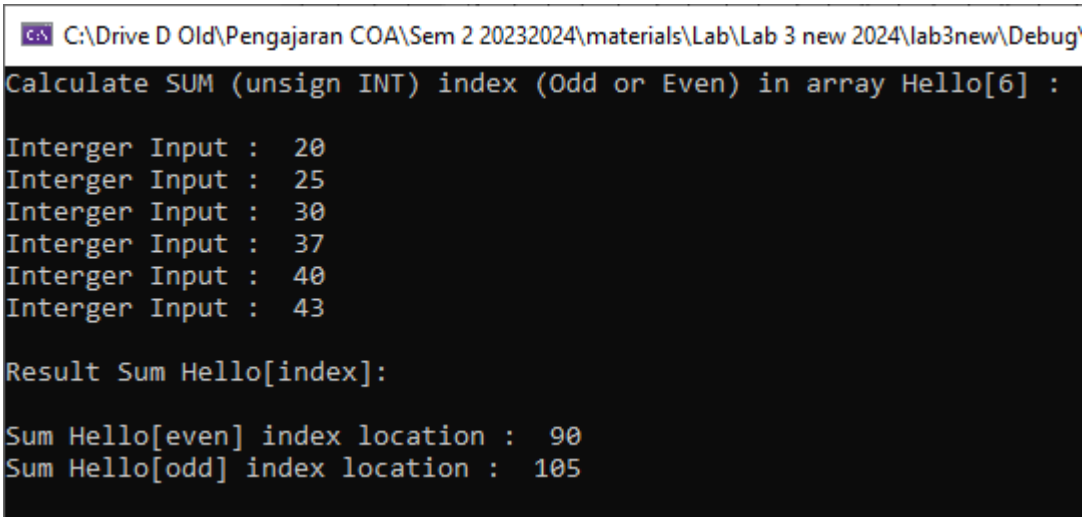


Figure 3: Sample Output

PART 2: ***Comparison & Conditional Jumps***

Part 2A – Programming review

A) BOOLEAN and COMPARISON INSTRUCTIONS

Logical Instructions

The processor instruction set provides the instructions AND, OR, XOR, TEST and NOT Boolean logic, which tests, sets and clears the bits according to the need of the program. These instructions set the CF, OF, PF, SF and ZF flags.

Conditional Instructions

Sometimes a program needs to do different things depending on the result of an operation. As shown in Figure 2.1, if the conditions are met then process A. Otherwise, proceed with process B. This is conditional branching. This is different from unconditional branching (the JMP instruction) previously studied.

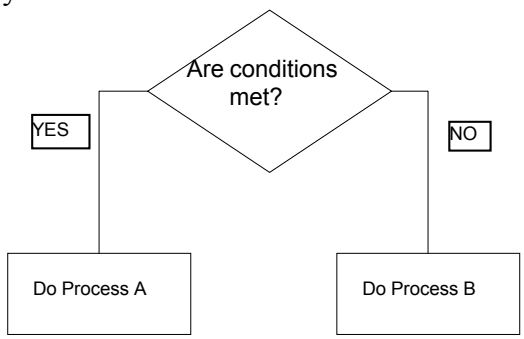


Figure 2.1

Compare (CMP) instruction

First, let us look at the compare (CMP) instruction. This instruction is used in to test branching conditions.

- The CMP instruction compares two operands.
- This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not.
- It does not disturb the destination or source operands (i.e. these does not change)
- The instruction format:
  - The destination operand can be either register or memory.
  - The source operand can be register, memory or immediate value.

```
CMP Destination, Source
```

```
CMP BX, 00 ; Compare the value in BX
with zeroJE TARGET  ; Jump to TARGET if
```

Sneak-peek: JE is Jump if Equal

B) CONDITIONAL JUMPS

Conditional Branching or Conditional Jump

This is performed by a set of jump instructions depending upon the condition. The conditional instructions transfer the control by breaking the sequential flow and they do it by changing the offset value in IP (Instruction Pointer). Written in the form J<condition>. Example: JE, JNZ, JL, JG

- There are different groups of conditional jump instructions:
  - Jumps based on specific flag values
  - Jumps based on equality between operands or the value of (E)CX
  - Jumps based on comparisons of unsigned operands
  - Jumps based on comparisons of signed operands
- The instruction format:

```
J<condition> TARGET
```

```
Examples:
      JE TARGET
      JNZ TARGET
      JL TARGET
```

Table 1: Jumps based on specific flag values

Instruction	Description
JZ	Jump if zero; ZF = 1
JNZ	Jump if not zero; ZF = 0
JC	Jump if carry; CF = 1
JNC	Jump if not carry; CF = 0
JO	Jump if overflow; OF = 1
JNO	Jump if not overflow; OF = 0
JS	Jump if signed; SF = 1
JNS	Jump if not signed; SF = 0
JP	Jump if parity (even); PF = 1
JNP	Jump if not parity (odd); PF = 0

Table 2: Jumps based on equality between operands or the value of (E)CX

Instruction	Description	Instruction	Description
JE	Jump if equal	JCXZ	Jump if CX= 0
JNE	Jump if not equal	JECXZ	Jump if ECX ≠ 0

Table 3: Jumps based on comparisons of unsigned operands

Instruction	Description	Instruction	Description
JA	Jump if above	JNBE	Jump if not below or equal
JAE	Jump if above or equal	JNB	Jump if not below
JB	Jump if below	JNAE	Jump if not above or equal
JBE	Jump if below or equal	JNA	Jump if not above

Note: These are only meaningful when comparing unsigned values

Table 4: Jumps based on comparisons of signed operands

Instruction	Description	Instruction	Description
JG	Jump if above	JNLE	Jump if not less or equal
JGE	Jump if above or equal	JNL	Jump if not less
JL	Jump if less	JNGE	Jump if not greater or equal
JLE	Jump if less or equal	JNG	Jump if not greater

Note: These are only meaningful when comparing signed values

Part 2B – Let's do a little programming by example

You are given a few examples here. Try them out.

Example 1

Increment AX by 1 until reaches the value of 10. This is essentially doing a loop using a CMP command.

```
;using CMP MOV
   EAX,0
L1:
   INC AX
   CMP AX, 10
JL L1
   MOV TOTAL, AX
```

```
;using LOOP
   MOV EBX,0
   MOV ECX,10
L2:
   INC BX
   LOOP L2
   MOV TOTALS, BX
```

Note: the result of both TOTAL and TOTALS are the same.

Example 2

Some conditional jumps examples.

```
   MOV AX,4
   CMP AX,4     ; compare AX with 4
   JE L1  ; if AX = 4 then jump to L1
   MOV BX,0AAAAH ; do this if AX ≠ 4

   JMP HERE     ; use this to guide the program sequence
L1:
   MOV BX, 0BBBBH ; do this if AX = 4

HERE:
   CALL DUMPREGS
```

```
      MOV AX,TOTAL   ; say TOTAL can be 2 or 4
      SUB AX,2
      JZ L2          ; if ZF = 1, jump to L2
      MOV  BX,0AAAAH  ; this is  done  if TOTAL = 4
      JMP HERE
L2:                  ; this is done if TOTAL = 2
      MOV BX,0BBBBH   ;
HERE:
      CALL DUMPREGS
```

Example 3

A look into signed and unsigned comparisons. The same value compared as signed and unsigned will yield different results. JA is a jump based on unsigned comparison while JG is ajump based on signed comparison. In the example below, JA will not go to L3 (unsigned 7Fh is smaller than unsigned 80h) but JG will jump to L4 (signed 7Fh is larger than signed 80h).

```
        MOV AX,7FH
        MOV BX,80H
        CMP AX,BX
        JA L3                          ; jump based on
unsigned comparison
        MOV CX,0AAAAH
        JMP HERE
L3:
       MOV CX, 0BBBBH

HERE:
       CALL DUMPREGS
       MOV AX,+127           ; signed version of 7FH
       MOV BX,-128           ; signed version of 80H
       CMP AX,BX
       JG L4                  ; jump based on signed
comparison
       MOV DX,0DDDDH
       JMP SINI

L4:
      MOV DX,0EEEEH

SINI:
      CALL DUMPREGS

EXIT
```

Part 2C – Let's do a little programming on your own

Combine in Part 1- Program 1 and Program 2 and create a MENU OPTION:

Example:

```
Welcome to Simple Math Activities:

Select Your Option (Main Menu):

1. To calculate Perimeter Hexagon (Loop and ADD instructions)
2. To calculate SUM (unsign int) index (Odd or Even) in an Array Matrix

Continue – type (y) and Exit type (n): y or n

Thank you ... BYE !!!
```

Branch Condition:

```
If (select = = 1)
        jmp_to periHex_loopAdd

If (select = = 2)
        jum_to calSum_oddeven

If (continue) = = y
            clear screen & Main Menu
        else
             Exit program
```

Operation Label:

```
periHex_loopAdd:
        do calculate Perimeter Hexagon (Loop and ADD instructions)

calSum_oddeven
        calculate SUM (unsign int) index (Odd or Even) in an Array Matrix
```

Sampel Output is shown in the following Figure 3.1 to 3.4.

Main Menu

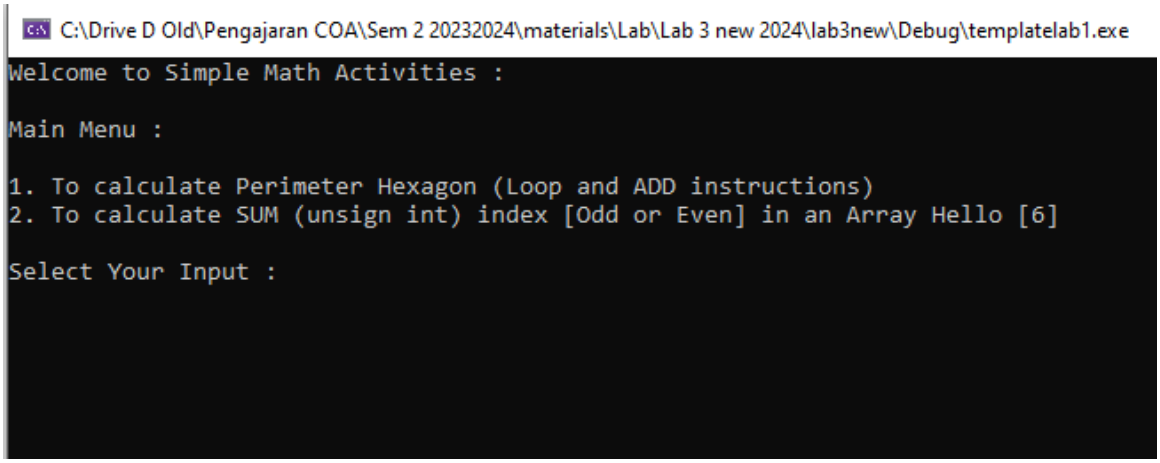


Figure 3.1. Main Menu

Select Option:

1. To calculate Perimeter Hexagon (Loop and ADD instructions)

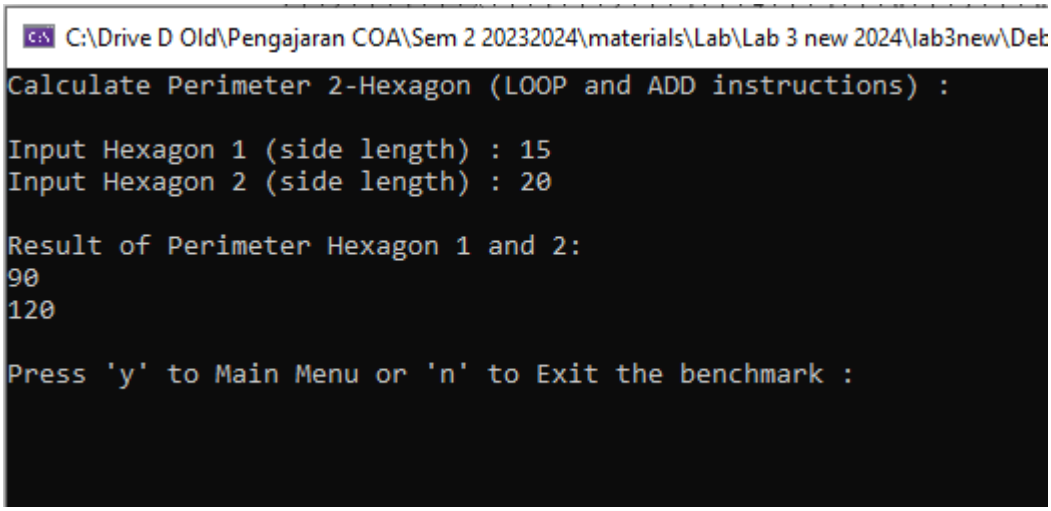


Figure 3.2. Perimeter Hexagon

2. To calculate SUM (unsign int) index (Odd or Even) in an Array Matrix

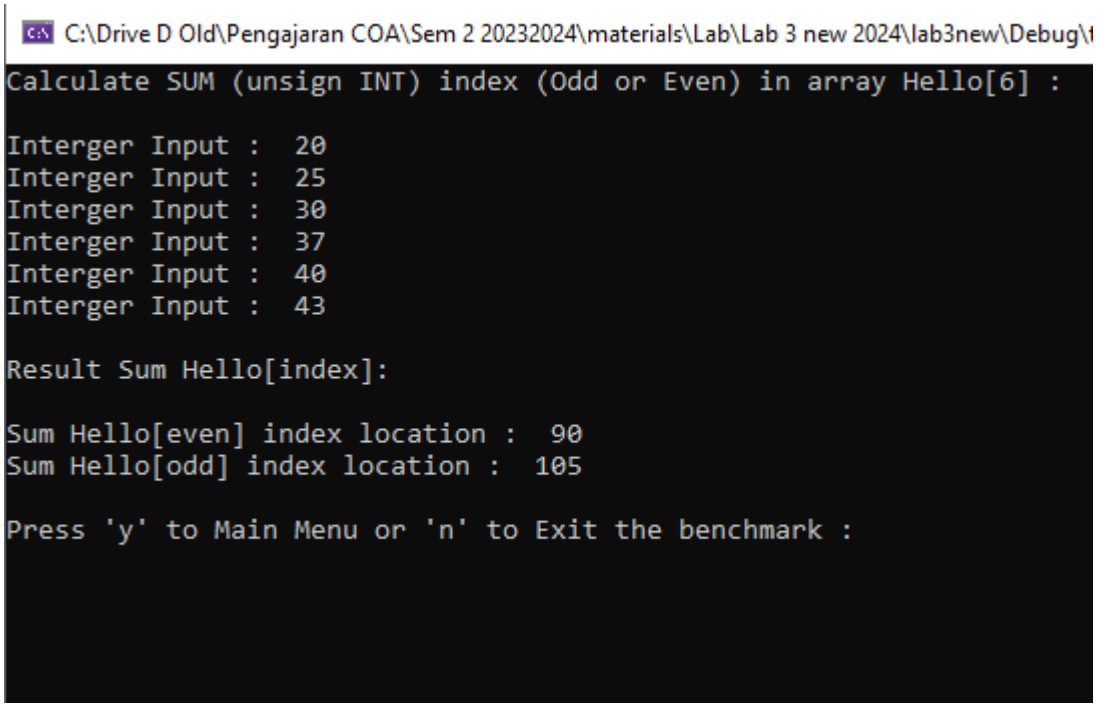


Figure 3.3. Array Sum Index Even and Odd in Hello [6]

3. Exit

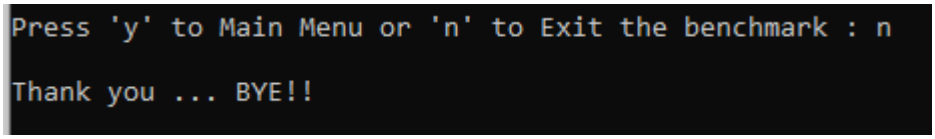


Figure 3.4. Exit Program