Lab (iii): Initializing and ending the program execution

In this practical, we will learn the process of assembling, linking, and executing a program. The instructions below demonstrate how the process works.

Steps	Command	Description
1	MASM filename.asm;	Assemble the source file. This command compiles the assembly code into an object file (.OBJ).
2	LINK filename.obj;	Link the object file. This command links the object file to create an executable file (.EXE).
3	filename.exe	Execute the program. This runs the executable file, allowing you to see the output of your program.

Other than that, we will go through several DOS functions for INT 21H. Below is a table to briefly understand the functions 01H, 02H, 09H, and 4CH.

Function	**Description**	**AH Value**	**Input Register(s)**	**Output Register(s)**
01H	Read a character from standard input	01H	None	AL (character read)
02H	Display a character on standard output	02H	DL (character)	None
09H	Display a string	09H	DX (pointer to string ending with '$')	None
4CH	Terminate process with return code	4CH	AL (return code)	None

Question 1: Declaring Variables and Displaying a Message

Write a program to declare variables with the specified value, then display the value of the result variable after arithmetic computation. Use simplified segment directives and include appropriate comments.

Using the following data definitions:

value1 DB 4

value2 DB 3

value3 DB 2

result DB ?

To calculate result, use the following arithmetic expression:

result = value1 – value2 + value3

Question 2: User Input Prompt

Write a program to fulfil the requirements below:

1. To display a message prompting the user to enter a single-digit integer.
2. Reading user input, convert it from ASCII to integer.
3. Store the input integer into a variable.
4. Convert this variable integer into ASCII.
5. Display the converted ASCII and make sure it's the user input integer.

Expected Output:

```
Enter a single-digit integer: 2
You have entered: 2
```

Question 3: Arithmetic and Debugging

a) Write a program to fulfil the requirements below:

1. Prompt the user to enter the first single-digit integer.
2. Reading user input, convert it from ASCII to integer.
3. Store the input integer into the first variable.
4. Prompt the user to enter the second single-digit integer.
5. Reading user input, convert it from ASCII to integer.
6. Store the input integer into the second variable.
7. Perform addition using the first variable and the second variable.
8. Convert this result integer into ASCII.
9. Display the converted ASCII.

Expected Output:

```
Enter the first integer: 2
Enter the second integer: 3
Result: 5
```

b) Are you able to display the result? Try displaying a two-digit result. Is it successful? Why or why not?

1. Debug the program to find the value of the result variable after addition.
    - Hint: Use DEBUG filename.exe or TD filename.exe
2. Modify the program to display two-digit results. Here are some hints:
    - Each ASCII character is represented by 1 byte of hexadecimal value.
    - To display 2 ASCII characters on the screen, you need to display 2 bytes of hexadecimal value.
    - To separate a 1-byte data into 2 bytes, try dividing the result by 10 (decimal) or 0A (hexadecimal).

Question 4: Multiplication and Division

a) Write a program to perform multiplication using 2 input one-digit integers.

Expected Output:

```
Enter the first integer: 3
Enter the second integer: 3
Result: 9
```

b) Write a program to perform division using 2 input one-digit integers.

Expected Output:

```
Enter the dividend: 5
Enter the divisor: 3
Result: 1
Remainder: 2
```

Question 5: Using the Stack

Write an 8086 assembly language program that:

1. Pushes the value 1234 onto the stack.
2. Pops the value from the stack into a variable num.

Use the .model, .stack, .data, and .code directives appropriately.

Conditional Jumps

Here's a table that summarizes the usage of the JA, JB, JE, JNE, and JMP instructions in 8086 assembly programming:

Instruction	Full Name	Description	Typical Usage
JA	Jump if Above	Jumps to the target label if the first operand is strictly greater than the second (unsigned)	Used after a CMP for unsigned comparison when the first operand is greater.
JB	Jump if Below	Jumps to the target label if the first operand is strictly less than the second (unsigned)	Used after a CMP for unsigned comparison when the first operand is less.
JG	Jump if Greater	Jumps to the target label if the first operand is strictly greater than the second (signed)	Used after a CMP for signed comparison when the first operand is greater.
JL	Jump if Less	Jumps to the target label if the first operand is strictly less than the second (signed)	Used after a CMP for signed comparison when the first operand is less.
JE	Jump if Equal	Jumps to the target label if the first operand is equal to the second	Used after a CMP or TEST to branch if operands are equal.
JNE	Jump if Not Equal	Jumps to the target label if the first operand is not equal to the second	Used after a CMP or TEST to branch if operands are not equal.
JMP	Jump	Unconditionally jumps to the target label	Used to transfer control unconditionally to a different part of the program.

Question 6: Infinite Looping and Condition Checking

a) Write an 8086 assembly language program that prompts the user to press either 'Y' or 'N'. The program should function as follows:

- If the user presses 'Y', the program should print the character 'A'.
- If the user presses 'N', the program should print the character 'B'.
- If the user presses any other key, the program should print the message "Please enter Y or N".

You can use the `INT 21H` interrupt to handle input and output operations. Ensure that your program loops until a valid key ('Y' or 'N') is pressed.

Hints:

1. Use `MOV AH, 01H` and `INT 21H` to get a single character input.
2. Use `CMP` to compare the input with 'Y' and 'N'.
3. Use conditional jumps (e.g., `JE`, `JNE`) to decide what to do based on the input.
4. Use `MOV AH, 09H` and `INT 21H` to print strings.
5. Use `MOV AH, 02H` and `INT 21H` to print characters.

Program Outline:

Start

1. Prompt User for Input
    - Display message: "Press Y or N"
    - Read user input using `INT 21H`
2. Compare Input with 'Y'
    - If input = 'Y', go to step 3
    - `If input ≠ 'Y', go to step 4`
3. Print 'A'
    - Display 'A'
    - End
4. Compare Input with 'N'
    - If input = 'N', go to step 5
    - `If input ≠ 'N', go to step 6`
5. Print 'B'
    - Display 'B'
    - End
6. Invalid Input
    - Display "Invalid input. Please enter Y or N only."
    - Go back to step 1 (Prompt User for Input)

b) Write an assembly program that prompts the user to enter a single digit between `0-9`.

You can use the `INT 21H` interrupt to handle input and output operations. Ensure that your program loops until a valid key (within 0-9) is pressed.

Hints:

1. The program should then compare the user's input to ensure it falls within the ASCII range `30h` to `39h`. If the input is valid, the program should print "Valid"; if not, it should print "Invalid".

2. Use `JG` and `JL` instructions to check if the input is within the ASCII range for digits (`30h` to `39h`).

Program Outline:

Start

1. Prompt User for Input
    - Display message: "Please enter a number within 0-9:"
    - Read user input using `INT 21H`
2. Compare Input with '0'
    - If input > '0', go to step 3
    - If input < '0', go to step 5
3. Compare Input with '9'
    - If input < '9', go to step 4
    - If input > '9', go to step 5
4. Valid input
    - Display 'Valid input'
    - End
5. Invalid Input
    - Display "Invalid input. Please enter a number within 0-9 only."
    - Go back to step 1 (Prompt User for Input)

Question 7: Array Manipulation Using Indexed Addressing in 8086 Assembly

Write an 8086 assembly language program that performs the following tasks:

1. Declare and initialize two arrays:
- `array1 db "hello$"`
- `array2 db "*****$"` (or `array2 db 5 dup ('*'), '$'`)
2. Using the `SI` and `DI` registers, copy the content from `array1` to `array2`.
3. Print both arrays before and after copying to verify the operation.

Sample Output:

Before copying:

```
array1: hello

array2: *****
```

After copying:

```
array1: hello

array2: hello
```

Question 8: Array Iteration and Conversion

Write an 8086 assembly program that accepts several integer characters from the user, and store them in an array.

Add the user input integer with a fixed value 123.

Next, iterate through the array containing the ASCII character of numbers, convert them into the decimal number, then print the result.

Instructions:

Part 1:

1. Declare an array to store 4 ASCII characters `array db 4 dup ('$')`.
2. Write a loop that accepts characters and iterates through the array to store 1 character in each memory address
3. Iterates through the array and converts each ASCII character to its decimal equivalent.
    a. Hint: ASCII character '1' has the hexadecimal value 31H. To convert it to its numerical value (1), subtract 48D or 30H.
4. Use a mathematical formula to construct a variable `var dw ?` using the previous converted value.
    a. Hint: '1','2','3','4' = 1×1000 + 2×100 + 3×10 + 4×1
    = 1234

This is a checkpoint. You should have a variable storing the decimal equivalent value now.

---

5. Perform addition by adding `123` into the variable `var`.

Part 2:

6. Use a mathematical formula to convert the value (in a variable) to the ASCII characters.
    a. Hint: 1234/1000 = 1 remainder 234
    234/100 = 2 remainder 34
    34/10 = 3 remainder 4
    OR
    1234/10 = 123 remainder 4
    123/10 = 12 remainder 3
    12/10 = 1 remainder 2
7. Print the ASCII characters using the appropriate `INT 21H` function.

Sample Output:

```
user input: 4567
output: 4690
```

In Part 1, students are able to learn how to convert from ASCII characters into decimal equivalent values, this is normally implemented during user input for a calculation operation.

In Part 2, students are able to learn how to convert from decimal values into ASCII characters for printing purposes.

Question 9: Array Iteration and String input

Write an 8086 assembly program that takes a string input from the user, then iterates through the string to count how many numeric digits ('0'-'9') and alphabetic characters ('A'-'Z', 'a'-'z') are in the string. Print the count of each.

Instructions:

1. Define a parameter list using `INT 21H, 0Ah` for string input. The maximum string length is 25 characters.

   Parameter list:

   ```
   para_list label byte
   inputBuffer db 25    ; maximum length of the input string
   inputLength db ?      ; stores actual length after input
   inputData db 25 dup('$') ; buffer to store the input data
   ```

2. Use `INT 21H, 0Ah` to get user input.
3. Write a loop that iterates through the input string (`inputData`), counting how many characters are numeric digits ('0'-'9') and how many are alphabetic characters ('A'-'Z', 'a'-'z').
4. Print the counts for numbers and alphabets using the appropriate `INT 21H` function.

Sample Output:

```
Enter a string: a1b2c3d4e
Numbers: 4
Alphabets: 5
```

Hints:

- **Numeric digits**: Check if a character lies between `'0'` (ASCII 48) and `'9'` (ASCII 57).
- **Alphabets**: Check if a character is between `'A'` (ASCII 65) and `'Z'` (ASCII 90) or between `'a'` (ASCII 97) and `'z'` (ASCII 122).
- Use two counters: one for numbers and one for alphabetic characters.