

**Table of Contents**

1. [Introduction](#1-introduction)
2. [Processor Architecture](#2-processor-architecture)  
   2.1. [Block Diagram](#21-block-diagram)
3. [Hardware Description](#3-hardware-description)  
   3.1. [Arithmetic Logic Unit (ALU)](#31-arithmetic-logic-unit-alu)  
   3.2. [Control Unit (CU)](#32-control-unit-cu)  
   3.3. [Registers](#33-registers)  
   3.4. [Memory](#34-memory)
4. [Interface Between Processor and I/O Devices](#4-interface-between-processor-and-io-de)  
   4.1. [Input Devices](#41-input-devices)  
   4.2. [Output Devices](#42-output-devices)
5. [Supported Instructions](#5-supported-instructions)
6. Algorithm and Code Design
7. Hardware Design

7.1 Components Overview

7.2 Pin Configuration

7.3 How the System Works at Hardware Level

1. [Conclusion](#6-conclusion)

**Members:**

Abeera Mehtab (232087)  
Mahnoor (232083)  
Mahnoor Ikram (232115)  
Eman Mansoor (232149)

1. **Introduction**

This project involves the design and implementation of a processor architecture for a basic calculator. The calculator supports arithmetic and logical operations and interacts with input and output devices. Phase 1 focuses on the processor's design, block diagram, hardware description, and the interaction between the processor and I/O devices.

1. **Processor Architecture**

**2.1. Block Diagram**

The block diagram illustrates the components of the calculator processor and their interactions. It includes:

1. **Arithmetic Logic Unit (ALU):** Handles arithmetic and logical operations such as addition, subtraction, multiplication, division, AND, OR, XOR, and NOT.
2. **Control Unit (CU):** Decodes instructions, manages control signals, and ensures synchronization between components.
3. **Registers:** Temporary storage for operands (NUM1, NUM2), operator (OPERATOR), and the result (RESULT).
4. **Memory:** Stores program instructions and data for programmable functionality.
5. **I/O Interface:** Interfaces with input devices (keypad) and output devices (LCD display).

**Note:** The ALU, CU, registers, and flags are part of the microcontroller, while the LCD serves as the output device and the keypad serves as the input device.

1. **Hardware Description**

**3.1. Arithmetic Logic Unit (ALU)**

* Constructed using basic logic gates (AND, OR, XOR, NOT) for logical operations.
* Arithmetic operations (addition, subtraction, multiplication, division) implemented using combinational circuits such as adders and subtractors all within Microcontroller

**3.2. Control Unit (CU)**

* Decodes instructions from the keypad and generates control signals for the ALU and registers.
* Ensures the correct sequence of operations and monitors flags (e.g., zero, carry, error).

**3.3. Registers**

* Stores intermediate values:
  + **NUM1:** First operand entered by the user.
  + **NUM2:** Second operand.
  + **OPERATOR:** Holds the selected operation (+, -, \*, /, etc.).
  + **RESULT:** Stores the computed result before display.

**3.4. Memory**

* Supports program storage for programmable designs or direct logic for a fixed-function calculator.

1. **Interface Between Processor and I/O Devices**

**4.1. Input Devices: Keypad**

* User inputs numbers, operations, and commands.
* Includes a reset button to clear data and restart the operation.

**4.2. Output Devices: LCD Display**

* Shows intermediate and final results.
* Displays error messages for invalid operations, such as division by zero or pressing = at the start.

1. **Supported Instructions**

The following instructions are supported by the processor:

1. **Load:** Load data from memory or input to registers.
2. **Store:** Store data from registers to memory.
3. **Add:** Perform addition of two operands.
4. **Sub:** Perform subtraction of two operands.
5. **Multiply:** Perform multiplication of two operands.
6. **Divide:** Perform division of two operands.
7. **AND:** Perform bitwise AND between two operands.
8. **OR:** Perform bitwise OR between two operands.
9. **XOR:** Perform bitwise XOR between two operands.
10. **NOT:** Perform bitwise NOT of a single operand.
11. **Exit:** Reset the system or terminate the program.
12. **Algorithm And Code Design:**

The MASM codeprovided implements a simple calculator that operates in two modes: **Arithmetic Mode** and **Logical Mode**. Here's an overview of the algorithm, followed by a block-by-block explanation:

**Overall Algorithm:**

1. The program starts by initializing in **Arithmetic Mode**.
2. It prompts the user to enter two operands (numbers).
3. The user is then prompted to enter an operator (+, -, \*, / for Arithmetic Mode or AND, OR, NOT, XOR for Logical Mode).
4. Based on the mode (Arithmetic or Logical), the program performs the corresponding operation:
   * In **Arithmetic Mode**, it performs basic arithmetic operations: addition, subtraction, multiplication, and division.
   * In **Logical Mode**, it performs bitwise operations like AND, OR, NOT, and XOR.
5. The result of the operation is printed, or an error message is shown if there is an invalid operation (e.g., division by zero).
6. The program supports toggling between **Arithmetic Mode** and **Logical Mode**.

**Block by Block Description:**

Here’s a detailed block-by-block description of our assembly program with the code for reference:

**1. Program Setup**

**Code Block:**

.386

.model flat, stdcall

.stack 4096

ExitProcess PROTO, dwExitCode:DWORD

INCLUDE Irvine32.inc

**Description:**

* Sets the processor to .386 (compatible with 386 and above).
* Specifies a flat memory model with standard calling convention (stdcall).
* Allocates a stack of 4096 bytes.
* Declares ExitProcess for program termination.
* Includes Irvine32.inc, a library for standard input/output operations in assembly.

**2. Data Section**

**Code Block:**

.data

; Display messages

msgArithmeticMode BYTE "Arithmetic Mode: Can perform +, -, \*, /", 0

msgLogicalMode BYTE "Logical Mode: Can perform AND, OR, NOT, XOR", 0

msgEnterOperand1 BYTE "Enter operand 1: ", 0

msgEnterOperand2 BYTE "Enter operand 2: ", 0

msgEnterOperator BYTE "Enter operator (+, -, \*, /): ", 0

msgResult BYTE "Result: ", 0

msgErrorDivByZero BYTE "Error: Division by zero", 0

msgInvalidInput BYTE "Invalid input. Please enter valid operands and operator.", 0

msgToggleMode BYTE "Press 'M' to toggle to Logical Mode", 0

msgExit BYTE "Exiting Calculator...", 0

; Variables for operands, result, mode, etc.

arithmeticOperand1 DWORD ?

arithmeticOperand2 DWORD ?

logicalOperand1 BYTE ?

logicalOperand2 BYTE ?

result DWORD ?

mode BYTE 0 ; 0 = Arithmetic, 1 = Logical

operator BYTE ?

inputKey BYTE ?

**Description:**

* Defines string messages used for user prompts and error handling.
* Declares memory locations for operands, results, mode (arithmetic or logical), and operator input.
* mode is initialized to 0 (arithmetic mode).

**3. Main Code Section**

**Code Block:**

main:

; Set up initial mode as Arithmetic

mov mode, 0

; Display welcome message for mode

call displayModeMessage

; Start main loop

startLoop:

; Continuously check for toggle mode and operator

call checkToggleMode

; Get operand1 input

call getOperand1

; Get operator input

call getOperator

; If operation is NOT (Logical), only one operand is required

cmp operator, '-'

je getOperand2Not

; Get operand2 input

call getOperand2

; Perform the calculation based on the mode

cmp mode, 0 ; Arithmetic Mode?

je performArithmetic

cmp mode, 1 ; Logical Mode?

je performLogical

jmp startLoop

**Description:**

* Initializes the mode as arithmetic (0) and displays the corresponding message.
* Main loop:
  + Toggles mode when 'M' is pressed.
  + Prompts for operands and operator input.
  + Decides calculation type (arithmetic or logical) based on the current mode.

**4. Operand and Operator Input Handling**

**Code Blocks:**

getOperand1:

; Display message for operand1

mov edx, offset msgEnterOperand1

call WriteString

; Read operand1

call ReadInt

; If in arithmetic mode, store in 32-bit, otherwise store in 8-bit for logical

cmp mode, 0

je storeArithmeticOperand1

; Logical Mode - Read as byte

movzx eax, al

mov logicalOperand1, al

ret

storeArithmeticOperand1:

mov arithmeticOperand1, eax

ret

**Description:**

* getOperand1 asks the user for the first operand and stores it in arithmeticOperand1 (arithmetic mode) or logicalOperand1 (logical mode).

getOperand2:

; Display message for operand2

mov edx, offset msgEnterOperand2

call WriteString

; Read operand2

call ReadInt

; Store in appropriate size based on mode

cmp mode, 0

je storeArithmeticOperand2

; Logical Mode - Read as byte

movzx eax, al

mov logicalOperand2, al

ret

**Description:**

* getOperand2 handles the input for the second operand and stores it similarly based on the current mode.

getOperator:

; Display message for operator

mov edx, offset msgEnterOperator

call WriteString

; Get the operator

call ReadChar

mov operator, al

ret

**Description:**

* getOperator reads the user’s selected operation and stores it in operator.

**5. Calculation**

**Code Blocks:**

performArithmetic:

; Perform arithmetic operations based on the operator

cmp operator, '+' ; Addition

je addOperands

cmp operator, '-' ; Subtraction

je subOperands

cmp operator, '\*' ; Multiplication

je mulOperands

cmp operator, '/' ; Division

je divOperands

jmp invalidOperator

**Description:**

* Checks the operator and jumps to the corresponding arithmetic operation.

performLogical:

; Perform logical operations based on the operator

cmp operator, '+' ; AND operation

je andOperands

cmp operator, '\*' ; OR operation

je orOperands

cmp operator, '-' ; NOT operation

je notOperand

cmp operator, '/' ; XOR operation

je xorOperands

jmp invalidOperator

**Description:**

* Similar to arithmetic, but performs logical operations

**6. Mode Toggle**

**Code Block:**

checkToggleMode:

; Check if user pressed 'M' to toggle mode

call ReadChar

cmp al, 'M'

je toggleMode

ret

toggleMode:

; Toggle between arithmetic and logical mode

cmp mode, 0

je setLogicalMode

cmp mode, 1

je setArithmeticMode

setLogicalMode:

mov mode, 1

call displayLogicalMsg

ret

setArithmeticMode:

mov mode, 0

call displayArithmeticMsg

ret

**Description:**

* Detects 'M' key press to toggle between arithmetic and logical modes, updating the mode and message accordingly.

**7. Results and Error Handling**

**Code Blocks:**

displayResult:

; Display result message

mov edx, offset msgResult

call WriteString

; Display the result value

call WriteInt

call Crlf

ret

divByZero:

; Handle division by zero

mov edx, offset msgErrorDivByZero

call WriteString

ret

invalidOperator:

; Handle invalid operator input

mov edx, offset msgInvalidInput

call WriteString

ret

**Description:**

* Displays results after calculations or error messages for invalid inputs or division by zero.

Thisblock-by-block explanation captures the program's logic for a calculator with both arithmetic and logical operation modes. Let me know if you need further clarification or modification!

**Summary:**

This MASM code is a simple calculator that supports both **Arithmetic Mode** (basic operations) and **Logical Mode** (bitwise operations). The program reads two operands and an operator from the user, validates inputs, performs the operation based on the current mode, and displays the result. The mode can be toggled, and logical inputs are restricted to 0 or 1.

1. **Hardware Design:**

At the hardware level, the system you've described involves several components: the **LM016L LCD**, **PIC16LF877 microcontroller**, a **4x4 matrix keypad**, and a **reset button**. Here's an overview of how each part of the system would interact, and how the pin configuration would work.

**Components Overview**

1. **LM016L LCD**:
   * This is a basic 16x2 character LCD used to display output to the user (such as results, prompts, etc.). It requires several control and data lines for interfacing with the microcontroller.
2. **PIC16LF877 Microcontroller**:
   * The PIC16LF877 is an 8-bit microcontroller that handles processing and control of the entire system. It will receive input from the matrix keypad, process the input, and send data to the LCD to display results.
   * It also manages the logic for switching between modes (arithmetic and logical), performing operations, and handling the reset button.
3. **4x4 Matrix Keypad**:
   * The keypad consists of 16 keys (4 rows and 4 columns), which can be used to input numbers and operators. The keys are connected in a matrix format, and the microcontroller will scan the rows and columns to detect which key is pressed.
4. **Reset Button**:
   * The reset button will set the calculator back to its default mode (e.g., Arithmetic Mode), clearing any previous inputs and calculations.

**Pin Configuration**

**LM016L LCD Pinout (assuming 4-bit mode):**

* **RS** (Register Select) – Controls the selection of command/data register.
* **RW** (Read/Write) – Controls the direction of data transfer.
* **E** (Enable) – Enables data to be latched.
* **D4-D7** (Data lines) – The data lines used for communication in 4-bit mode (using 4 data lines instead of 8).
* **VSS** – Ground.
* **VDD** – Power supply (typically 5V).
* **V0** – Contrast control.
* **K** – Cathode of backlight (connected to GND).
* **A** – Anode of backlight (connected to VDD through a current-limiting resistor).

**PIC16LF877 Pinout:**

The PIC16LF877 microcontroller has various pins dedicated to different functions:

* **PORTA** – This can be used for analog input or as digital input/output.
* **PORTB** – This port can be configured for I/O operations and is often used for interfacing with external peripherals like keypads or LEDs.
* **PORTC** – Used for communication with external devices like the LCD (RS, E, Data lines).
* **PORTD** – Can be used for additional I/O, often used for keypads or interfacing with other peripherals.
* **PORTB7 (reset pin)** – This pin is used for resetting the microcontroller.

**4x4 Matrix Keypad Pinout:**

* The keypad has 4 rows and 4 columns. The rows and columns will be connected to the microcontroller. When a key is pressed, the row and column will be shorted, allowing the microcontroller to determine which key was pressed.

**Example Pin Assignment**:

* **Row 1** (Connected to PORTC0)
* **Row 2** (Connected to PORTC1)
* **Row 3** (Connected to PORTC2)
* **Row 4** (Connected to PORTC3)
* **Column 1** (Connected to PORTD0)
* **Column 2** (Connected to PORTD1)
* **Column 3** (Connected to PORTD2)
* **Column 4** (Connected to PORTD3)

**Reset Button:**

* The reset button is typically connected to a pin on the microcontroller configured as an input (e.g., PORTB0). Pressing the button will initiate a reset operation, typically via an interrupt or polling mechanism.

**How the System Works at Hardware Level**

1. **Keypad Input**:
   * When a key is pressed on the 4x4 matrix keypad, the microcontroller scans the rows and columns. It detects the key press by identifying the row and column that are shorted together, and then the corresponding character (number, operator, or function key) is processed by the microcontroller.
2. **LCD Display**:
   * The microcontroller sends data to the LCD to display characters. This involves sending commands to the RS, RW, and E pins to control the display in 4-bit mode.
   * The microcontroller may update the LCD display whenever a new operation is performed, a result is calculated, or user input is received.
3. **Mode Switching**:
   * The microcontroller continuously checks the current mode (Arithmetic or Logical). Based on the mode, it will determine which operation to perform. In arithmetic mode, standard operations like addition, subtraction, multiplication, and division will be executed. In logical mode, logical operations like AND, OR, NOT, and XOR are performed.
   * The **reset button** will allow the user to return to the default mode (usually Arithmetic Mode) and reset the calculator's state.
4. **Reset Mechanism**:
   * When the reset button is pressed, the microcontroller reads the signal from the input pin. It then resets the system to its default state, clearing the LCD and setting the mode to Arithmetic. It may also clear any previous operands or results stored in memory.
5. **Pin Control**:
   * The microcontroller uses specific pins to interface with the keypad, LCD, and reset button. It will control the data flow through these pins by setting them as inputs or outputs and using them to communicate with the external hardware.

**Hex File**

The hex file generated from the PIC code would contain the machine code for the PIC16LF877 to execute the programmed operations. It is loaded into the microcontroller's memory and executed to perform the desired functionality (reading keypad input, performing calculations, displaying results on the LCD, etc.).

This system relies on proper pin configuration and efficient use of the microcontroller's I/O ports for interacting with the keypad and LCD. The hex file you mentioned would essentially be the compiled program that is burned into the PIC16LF877, which contains all the logic and configurations to perform the calculator's operations.

1. **Conclusion**

This project successfully demonstrates the design and implementation of a basic calculator processor capable of performing arithmetic and logical operations. By integrating a robust architecture with essential components such as an ALU, Control Unit, registers, and memory, the processor ensures efficient and reliable performance. The use of a PIC16LF877 microcontroller, paired with a 4x4 matrix keypad and LM016L LCD, facilitates seamless interaction between the user and the hardware.

The implementation of dual-mode functionality—Arithmetic and Logical modes—enhances the calculator's versatility, enabling it to handle a wide range of operations. The MASM code and the corresponding hardware configurations ensure precise calculations and user-friendly output displays. Furthermore, the inclusion of error handling and a reset mechanism enhances the system's usability and reliability.

This project highlights the effective integration of hardware and software in building a functional and user-centric computational device. It lays a strong foundation for future enhancements, such as expanding the instruction set, integrating more complex operations, or adding programmable features for greater flexibility.