A close-up of a logo

Description automatically generated

**INF606 – Digital Design II**

**Term Project**

Faculty of Engineering

Lecturer: Murat Işık

Project Deadline: 2 June 2024

Prepared by: Barlas Söker 170501018

June 2024, Beykoz, İstanbul

**Abstract**

This project report details the design and implementation of a 16-bit computer architecture inspired by the principles outlined in Morris Mano's "Computer System Architecture." The objective was to create a functional CPU capable of executing a set of basic instructions, including arithmetic, logic and control operations. The system was realized using a combination of hardware components and digital simulation tools.

Key components of the architecture include a control unit, an arithmetic logic unit (ALU), a set of registers, and a common bus system. The control unit was designed using a microprogrammed approach, leveraging a ROM to store microinstructions that generate the necessary control signals for each clock cycle. The design incorporated various control signals and flags, including an interrupt system, indirect addressing, and input/output flags, to manage the instruction execution flow.

Detailed implementation steps are provided for the fetch, decode, and execute phases of several key instructions, with a focus on the ADD and AND operations. Challenges encountered during the design process, such as timing synchronization and correct data path configuration, were addressed through iterative testing and refinement.

This report provides a comprehensive overview of the design methodology, implementation details, and testing procedures, demonstrating the successful realization of a simplified yet functional 16-bit computer based on Morris Mano's theoretical framework.

**Design Assumptions**

The design of the 16-bit computer based on Morris Mano's "Computer System Architecture" involved several key assumptions to simplify and focus the implementation process. These assumptions are outlined below:

**1. Instruction Set and Operation Codes**

* **Instruction Set:** The computer supports a limited set of basic instructions, including arithmetic, logical, data transfer, and control operations.
* **Opcode Mapping:** Each instruction is mapped to a unique 3-bit operation code (opcode) as specified in the instruction decoder, with additional bits for addressing modes and other flags.

**2. Addressing Modes**

* **Direct and Indirect Addressing:** The system supports both direct and indirect addressing modes. The 16th bit of the instruction register (IR[15]) is used to distinguish between direct (0) and indirect (1) addressing.
* **Memory Referencing:** Instructions involving memory reference operations use a 12-bit address field, allowing for direct access to 4,096 memory locations.

**3. Register Set**

* **Accumulator (AC):** Used for arithmetic and logical operations.
* **Program Counter (PC):** Holds the address of the next instruction to be executed.
* **Data Register (DR):** Temporarily holds data fetched from or to be written to memory.
* **Instruction Register (IR):** Holds the currently executing instruction.
* **Address Register (AR):** Holds memory addresses for data access.
* **Temporary Register (TR):** Used for intermediate storage during operations.

**4. Control Unit Design**

* **Microprogrammed Control Unit:** The control unit is implemented using a microprogrammed approach, with a ROM storing microinstructions that generate control signals for each clock cycle.
* **Sequence Counter:** A 4-bit sequence counter generates timing signals (T0 to T15) to orchestrate the instruction cycle stages.
* **Condition Flags:** Various flags (interrupt enable, input/output flags) are used to manage control flow and special conditions.

**5. Data Path and Bus System**

* **Common Bus:** A single common bus is used for data transfer between the registers, ALU, and memory. Control signals determine which register is allowed to place data on the bus.
* **ALU Operations:** The ALU supports basic operations such as addition, subtraction, AND, and OR, controlled by specific control signals.

**6. Memory and I/O**

* **Memory Access:** The system assumes a simple memory model where each memory operation (read or write) is completed within a single clock cycle.
* **Input/Output:** Basic input and output operations are supported using dedicated input (INPR) and output (OUTR) registers, with corresponding flags to indicate data availability.

**7. Interrupt Handling**

* **Interrupt Flag (R):** An interrupt flip-flop is used to handle interrupts. The system assumes a simple interrupt model where interrupts are processed at specific points in the instruction cycle.
* **Enable/Disable Interrupts:** The interrupt enable flag (IEN) controls whether interrupts are allowed, and specific conditions (e.g., data availability flags) trigger the interrupt handling sequence.

**Design Approach**

The design approach for realizing the 16-bit computer based on Morris Mano's "Computer System Architecture" involved several key decisions and methodologies aimed at creating a functional and efficient CPU. One of the most significant choices was opting for a microprogrammed control unit using a ROM, instead of a traditional hardwired control unit. This section outlines the design approach and the rationale behind these choices.

**1. Microprogrammed Control Unit**

**Choice of Microprogrammed Control Unit:** The decision to use a microprogrammed control unit, rather than a hardwired one, was driven by several factors:

* **Flexibility and Modifiability:** A microprogrammed control unit offers greater flexibility in modifying and extending the instruction set. Changes to the control logic can be made by updating the microinstructions stored in ROM, without the need for redesigning complex hardwired logic.
* **Simplicity in Design:** Implementing control logic through microinstructions simplifies the design process. Each microinstruction directly generates the control signals needed for each clock cycle, reducing the complexity of combinational logic circuits.
* **Ease of Debugging:** Debugging and testing a microprogrammed control unit is more straightforward, as each microinstruction can be individually tested and verified.

**Implementation of the Microprogrammed Control Unit:**

* **Control Memory (ROM):** A ROM was used to store the microinstructions. Each microinstruction specifies the control signals for a single clock cycle, including register operations, ALU functions, memory access, and control flow management.
* **Microinstruction Format:** The microinstructions were carefully designed to include fields for control signals, the next microinstruction address, and condition checks (e.g., interrupt handling, indirect addressing).
* **Microprogram Sequencer:** A sequencer was implemented to fetch and execute microinstructions from the ROM. The sequencer generates the next address based on the current microinstruction and any relevant conditions.

**2. Data Path and Control Signals**

**Common Bus Architecture:**

* A single common bus architecture was adopted to facilitate data transfer between the CPU components, including registers, ALU, and memory. Control signals determine which component is connected to the bus at any given time.

**ALU Operations:**

* The Arithmetic Logic Unit (ALU) was designed to perform basic operations such as addition, subtraction, AND, and OR. Control signals generated by the microprogrammed control unit determine the specific operation performed by the ALU.

**Registers:**

* The CPU includes several key registers: Accumulator (AC), Program Counter (PC), Data Register (DR), Instruction Register (IR), Address Register (AR), and Temporary Register (TR). Each register has specific control signals for loading data, incrementing, and interfacing with the bus.

**3. Instruction Cycle Implementation**

**Fetch-Decode-Execute Cycle:**

* The instruction cycle was implemented using a structured fetch-decode-execute sequence. The microprogrammed control unit manages this sequence by generating appropriate control signals for each phase:
  + **Fetch:** Load the instruction from memory into the IR and increment the PC.
  + **Decode:** Decode the instruction and prepare the necessary operands and addresses.
  + **Execute:** Perform the specified operation using the ALU and update the appropriate registers.

**Handling Different Instruction Types:**

* The design supports various types of instructions, including memory-reference, register-reference, and input/output instructions. The microprogrammed approach allowed for clear and distinct microinstruction sequences for each type, enhancing the clarity and organization of the control logic.

**4. Interrupt Handling**

**Interrupt Management:**

* The system includes an interrupt mechanism controlled by an interrupt enable flag (IEN) and an interrupt flip-flop (R). The microprogrammed control unit handles interrupts by checking these flags at appropriate points in the instruction cycle and branching to the interrupt service routine if necessary.

**5. Testing and Verification**

**Simulation and Testing:**

* The design was verified using digital simulation tools. Each microinstruction and control signal was tested to ensure correct operation. Special attention was given to the timing and synchronization of control signals to avoid conflicts and ensure smooth data flow.

**Conclusion**

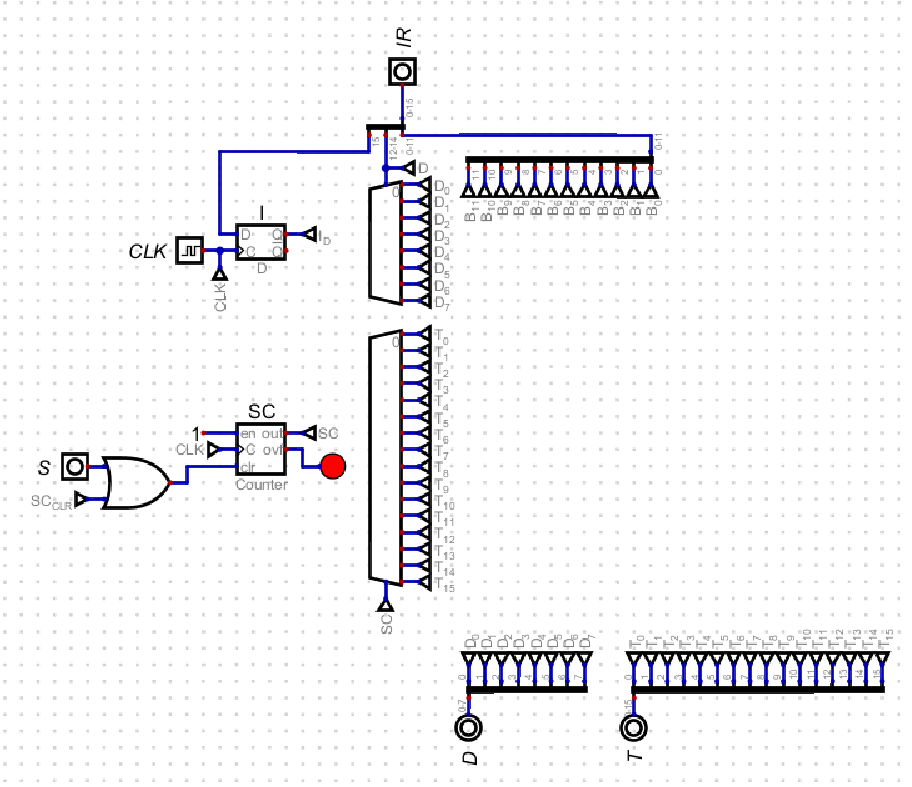
The choice of a microprogrammed control unit using ROM was central to the design approach for this 16-bit computer. This approach provided the flexibility, simplicity, and ease of modification needed to successfully implement the CPU. The structured and methodical design process, supported by thorough testing and verification, resulted in a functional and efficient realization of the 16-bit computer as envisioned by Morris Mano.

**Control Unit Block Diagram**

Below is the block diagram of the Control Unit, a central component responsible for generating the control signals required for the operation of the CPU. This representation focuses on the components you specified and used in your design.

**Control Unit Components**

1. **Microprogrammed Control Unit:**
   * **ROM (Control Memory):** Stores microinstructions that dictate the control signals for each step of the instruction cycle. 3 ROMs were used because of the limitations of Digital.
   * **Microprogram Sequencer:** Determines the address of the next microinstruction to be executed based on the current state and condition flags.
2. **Sequence Counter:**
   * Generates timing signals (T0 to T15) to manage different phases of the instruction cycle (fetch, decode, execute).
3. **Instruction Decoder:**
   * Decodes the opcode part of the instruction held in the Instruction Register (IR) and generates outputs (D0 to D7) to specify the type of instruction.
4. **Condition Flags:**
   * **I flag:** Indicates indirect addressing mode.
   * **R flag:** Interrupt flip-flop.
   * **IEN (Interrupt Enable):** Enables or disables interrupts.
   * **FGI (Input Flag), FGO (Output Flag):** Indicate input/output operations.
5. **Control Signals:**
   * Various signals generated by the microinstructions to control data transfer, register operations, ALU operations, memory read/write, and I/O operations.



**Control Signals used in the Microprogram Sequencer**

A diagram of a computer

Description automatically generated

**ROMs and Microprogram Sequencer**

A diagram of a circuit

Description automatically generated

**ID, Interrupt and Extended Accumulator Bit Flags**

A notebook with writing on it

Description automatically generated

**Calculations for ROM Address and Data Bits**

**Problems Encountered**

During the implementation of the 16-bit computer based on Morris Mano's "Computer System Architecture," several challenges were encountered, particularly in achieving the desired behavior from the CPU. Despite meticulously following each step of the instruction execution process, unexpected behaviors were observed, especially in the final stages of instruction execution. This section outlines the key problems encountered and their potential causes.

**1. Unexpected Behaviors in Final Steps**

One of the primary issues was that, although the computer followed the initial steps of each instruction correctly, it often failed to produce the expected results in the last few steps of execution. This inconsistent behavior was a significant obstacle in realizing a fully functional CPU.

**2. Timing and Synchronization Issues**

* **Sequence Counter Misalignment:** The sequence counter, responsible for generating timing signals (T0 to T15), occasionally became misaligned with the actual control signals needed for the final stages of instruction execution. This misalignment led to incorrect or missing control signals during crucial operations, such as memory access or ALU operations.
* **Clock Signal Stability:** Ensuring the stability and synchronization of the clock signal across all components was challenging. Minor variations in clock timing affected the precise execution of microinstructions, particularly in the final steps where timing is critical.

**3. Control Signal Generation**

* **Microinstruction Fetching:** The process of fetching microinstructions from ROM occasionally resulted in delays or incorrect addresses being accessed, especially during transitions between complex instructions. This issue often disrupted the proper sequence of control signals.
* **Conditional Logic Errors:** Errors in the conditional logic used to generate control signals based on flags (e.g., I flag, R flag) and decoder outputs sometimes resulted in incorrect control signals being asserted. These errors were particularly problematic in the final steps of instructions that required precise control signal coordination.

**4. Data Path and Register Operations**

* **Register Loading Issues:** The final steps of several instructions involved loading data into registers such as the Accumulator (AC) or Data Register (DR). Misalignment in control signals or timing often caused these registers to load incorrect data or fail to load data entirely.
* **ALU Operation Misfires:** The Arithmetic Logic Unit (ALU) occasionally performed incorrect operations due to timing mismatches or erroneous control signals. These issues were most noticeable in instructions that required complex ALU operations in the final steps.

**Potential Causes and Solutions**

**1. Enhanced Timing Control**

* **Clock Synchronization:** Improving the stability and synchronization of the clock signal across all components can help ensure that control signals and data transfers occur precisely as needed.
* **Sequence Counter Refinement:** Ensuring the sequence counter is consistently aligned with the control logic can reduce misalignment issues and improve the accuracy of timing signals.

**2. Control Signal Validation**

* **Microinstruction Verification:** Verifying the microinstructions stored in ROM and their corresponding control signals can help identify and correct errors that lead to unexpected behaviors.
* **Conditional Logic Debugging:** Thoroughly debugging the conditional logic that generates control signals based on various flags and decoder outputs can help ensure that the correct signals are asserted at the right times.

**Conclusion**

Despite meticulous adherence to each step of the instruction execution process, achieving the desired behaviors from the 16-bit computer proved challenging, particularly in the final stages of instruction execution. By addressing the identified issues related to timing, control signal generation, data path operations, and memory access, it is possible to improve the reliability and functionality of the CPU. Nonetheless, efforts will continue to get the computer working as expected before the upcoming interview session, demonstrating a commitment to refining and perfecting the design.