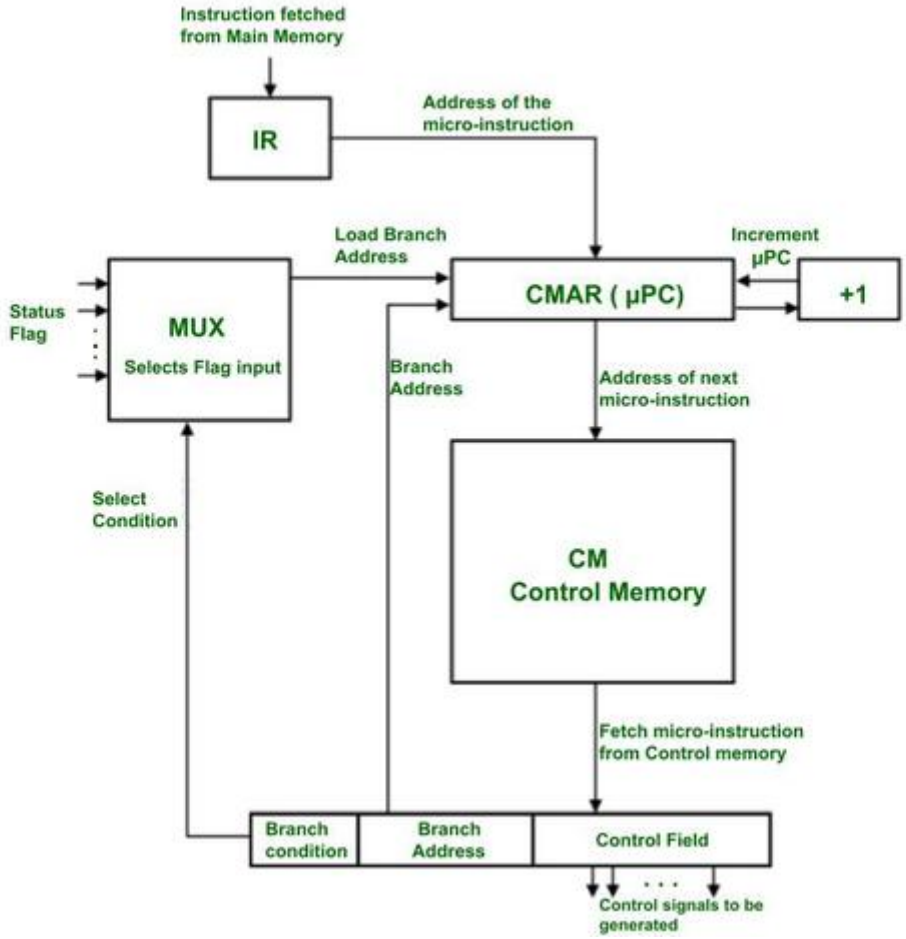
3. Principles of Microprogrammed Control

This section details the core theory of the microprogrammed control unit, the flexible alternative to the hardwired FSM. This approach fundamentally changes the design from a complex, fixed logic circuit to a simple, programmable one.

- **3.1 Core Concept: The "Computer-within-a-Computer"**
  - The most effective analogy for a microprogrammed control unit is that it is a "computer-within-a-computer." The main CPU (which has assembly instructions like `ADD`, `LDA`, `JMP`) is the "outer" computer. The Control Unit itself is a tiny, hidden, "inner" computer.
  - This inner computer has its own simple program, called a **microprogram**. The microprogram is composed of a sequence of instructions called **micro-instructions**.
  - The key relationship is this: **One assembly instruction** (like `LDA`) is interpreted by the CU as a command to run a specific **micro-routine** (a "subroutine" of micro-instructions). The micro-routine is the step-by-step recipe that defines *how* to execute that single assembly instructior.
- **3.2 Architectural Components**
  - To function as a simple computer, the microprogrammed CU has its own set of internal hardware components, separate from the main datapath.
  - **3.2.1 Control Store (or Control Memory):**
    - This is a small, high-speed Read-Only Memory (ROM) that is located *inside* the Control Unit.
    - Its sole purpose is to store the entire microprogram—that is, all the micro-routines for every assembly instruction in the CPU's instruction set (e.g., the routines for `LDA`, `STA`, `ADD`, `JMP`, etc.).
  - **3.2.2 Micro-Sequencer (Next Address Logic):**
    - This is the "Program Counter" for the Control Unit (often called a `uPC` or "micro-Program Counter").
    - Its job is to determine the address of the *next* micro-instruction to be fetched from the Control Store.
    - It decides this address based on several inputs: the main instruction's `opcode` (for the initial "decode" jump), CPU status flags (for conditional branches like `JEQ`), and sequencing fields from the current micro-instruction (e.g., `SEQ_NEXT`, `SEQ_FETCH`).
  - **3.2.3 Micro-instruction Register (uIR):**
    - This is a register that holds the *current* micro-instruction that was just fetched from the Control Store.
    - This is the most critical component for execution: the output bits of this register **are** the actual control signals that are sent to the datapath.

- For example, a 16-bit `uIR` might directly output 16 control signals (like `Register_Enable`, `ALU_Subtract`, `Memory_Write`, etc.) to the rest of the CPU.



Revision #3
Created 2025-11-11 17:24:41 UTC by AX
Updated 2025-11-11 18:25:39 UTC by AX