4. The Micro-instruction

If the Control Store is the "recipe book" for the CPU, then a micro-instruction is a single "line" in that recipe. It is the most fundamental unit of control in a microprogrammed CPU, defining the exact set of hardware operations that will occur in a single clock cycle.

- **4.1 Definition** A micro-instruction is a single word fetched from the Control Store. Its primary function is to specify the control signals to be generated for one clock tick. Each micro-instruction is held in the Micro-instruction Register (uIR), and the bits of this register are used—either directly or after decoding—to activate or deactivate every control line in the CPU's datapath [1].
- **4.2 Format and Fields** While the exact layout varies, a micro-instruction is typically divided into two primary types of fields [2].
  - **4.2.1 Control Field:** This is the main part of the micro-instruction. It contains the bits that directly control the datapath components. For example, a "1" in a specific bit position might enable a register to output its value to the main bus (like `RA0`), while a "0" keeps it disabled. Another set of bits might specify the operation for the ALU (e.g., `SUB=1`).
  - **4.2.2 Sequencing Field (Next Address Field):** This field doesn't control the datapath; it controls the Micro-Sequencer itself. These bits provide the sequencer with information to determine the address of the *next* micro-instruction. This field might contain:
    - A specific "next address" to jump to.
    - A "branch condition" code (like `SEQ_DECODE` or `SEQ_JUMP_ON_ZERO`) that tells the sequencer *how* to find the next address by checking opcodes or status flags [3].
- **4.3 Horizontal vs. Vertical Microprogramming** The "Control Field" can be designed in two primary ways, which presents a classic trade-off between speed and memory efficiency [4].
  - **4.3.1 Horizontal Microprogramming:** This is a "decoded" or "unencoded" approach. The micro-instruction is very wide, and each bit in the control field corresponds directly to a single control line. If the CPU has 60 control signals, the control field is 60 bits wide.
    - **Pros:** It is extremely fast. No additional decoding logic is needed; the bits from the uIR can be used directly. It also allows for high parallelism, as many signals can be activated in the same cycle.
    - **Cons:** It is very inefficient. A micro-instruction that only activates 2 signals (e.g., `RA0` and `RBI`) still requires the full 60 bits, wasting space in the Control Store.
  - **4.3.2 Vertical Microprogramming:** This is an "encoded" approach. Instead of one bit per signal, groups of signals are encoded into fields. For example, if an ALU has 8 possible operations (ADD, SUB, AND, OR, etc.), a 3-bit field (`2^3 = 8`) can be used to select which operation to perform.

- **Pros:** It is highly efficient and saves a significant amount of space in the Control Store. The micro-instructions are much narrower.
- **Cons:** It is slower. The encoded fields (like the 3-bit ALU field) must be passed through an external decoder circuit to generate the final control signals, adding a layer of gate delay.

In practice, most modern designs are a hybrid, using vertical encoding for mutually exclusive signals (like ALU operations) and horizontal bits for independent signals (like `Memory_Write`).

Revision #2
Created 2025-11-11 18:08:57 UTC by AX
Updated 2025-11-11 18:25:32 UTC by AX