Semiconductor Technology from A to Z

Digital technology

Control Unit and Instruction Execution

Datapath and Control Unit: Who Does What

A processor can be roughly divided into two parts: the datapath and the control unit. The datapath consists of exactly the building blocks covered in previous chapters – an ALU (adders, multipliers), a set of registers (flip-flops, usually grouped into what is called a register file with 16 to 32 entries), and buses that move data between them, with multiplexers at every point where a choice must be made about which data source is currently being passed through. The datapath itself makes no decisions; every clock cycle, it does exactly what the signals present at its control inputs force it to do.

Concretely, these are typically: the address lines of the register file, which determine which two registers are placed on the ALU inputs and which register the result is written back to; a write-enable signal that actually authorizes this write-back; an ALU opcode that determines whether the ALU adds, subtracts, forms a logical combination, or shifts; and various mux-select lines that determine, for example, whether the second ALU input comes from a register or directly from a constant value (immediate) contained in the instruction. All of these signals are generated by the control unit, purely from two inputs: the current clock cycle within instruction execution, and the so-called opcode, the first few bits of the instruction currently being executed. The control unit "understands" nothing in the human sense here – it is itself just a (potentially very complex) combinational and sequential circuit that outputs a fixed, predetermined pattern of control signals for every combination of opcode and cycle.

Opcode and the Fetch-Decode-Execute Cycle

An instruction in memory is nothing but a fixed bit sequence, typically 16 to 32 bits wide, divided into an opcode (which operation) and operands (with which registers or values). In a heavily simplified 32-bit add instruction, for example, bits 26 through 31 might encode the opcode for "add," bits 21 through 25 the first source register, bits 16 through 20 the second source register, and bits 11 through 15 the destination register – the remaining bits would go unused or, for other opcodes, hold an immediate value. A special register, the program counter (PC), always holds the memory address of the instruction to be executed next.

Execution follows a constantly repeating pattern: in the fetch phase, the instruction at the address given by the PC is read from memory and loaded into a special instruction register; in the decode phase, the control unit breaks down

the opcode and derives the control signals needed for this instruction, while in parallel the registers named in the opcode field are already being read from the register file; in the execute phase, the datapath, driven by exactly these signals, carries out the actual operation, such as an ALU computation, followed by an optional memory-access phase (for load/store instructions) and a final write-back phase into the destination register. At the end, the PC is incremented by the width of the current instruction (for most instructions, simply by 4 bytes) – or, for a branch instruction, set directly to a new address computed from the instruction, for example when a condition such as "register equals zero" is met. The cycle then begins again with the next instruction. A program is therefore ultimately nothing more than a long, compiler-generated sequence of such bit patterns in memory, working through this cycle instruction by instruction with a changing opcode – branch instructions ensure that this sequence does not necessarily run linearly through memory but takes different paths depending on intermediate results, which is ultimately what makes loops and conditional statements possible at the hardware level.

Hardwired versus Microcoded Control Units

Two fundamentally different approaches have historically emerged for implementing the control unit itself. In a hardwired control unit, the mapping from opcode and cycle to control signals is implemented directly as combinational logic and a small state machine in gates – comparable to the circuits covered in previous chapters, just at a higher level of abstraction, where the "inputs" are entire opcode fields rather than individual bits. This approach is very fast, since the control signals arise directly from the wiring without an additional memory access, but for a large, irregular instruction set with many special cases it quickly becomes unwieldy, hard to verify, and practically impossible to correct once errors are discovered after the fact, since any change would require a new mask-set revision.

In a microcoded control unit, by contrast, a dedicated sequence of "microinstructions" is stored for each opcode in a small internal ROM or PLA, delivering step by step exactly the control signals that a hardwired control unit would otherwise generate directly through wiring; a complex instruction is thereby internally broken down into a sequence of simpler micro-operations, similar to a small, permanently programmed subroutine at an even lower level than actual machine code. This approach is easier to design and verify, since new or corrected instructions in principle require only a new microcode table rather than new gate wiring, and modern x86 processors still exploit this property today for so-called microcode updates, which can fix bugs or security vulnerabilities in instruction execution via firmware even after shipping. The

microcode access, however, costs additional clock cycles compared to a purely hardwired solution, and was one of the main drivers behind the RISC philosophy of the 1980s, which deliberately adopted a heavily simplified, uniform instruction set with a consistent instruction length in order to go back to being purely hardwired and thus faster to control – a trade-off that today processors often resolve in hybrid fashion, decoding simple, frequent instructions in hardwired fashion while only rare, complex instructions take the slower microcode path.