Maximally vs. Minimally Encoded

- No encoding:
  - 1 bit for each datapath operation
  - faster, requires more memory (logic)
  - used for Vax 780 — an astonishing 400K of memory!
- Lots of encoding:
  - send the microinstructions through logic to get control signals
  - uses less memory, slower
- Historical context of CISC:
  - Too much logic to put on a single chip with everything else
  - Use a ROM (or even RAM) to hold the microcode
  - It’s easy to add new instructions

Microcode: Trade-offs

- Distinction between specification and implementation is sometimes blurred
- Specification Advantages:
  - Easy to design and write
  - Design architecture and microcode in parallel
- Implementation (off-chip ROM) Advantages
  - Easy to change since values are in memory
  - Can emulate other architectures
  - Can make use of internal registers
- Implementation Disadvantages. SLOWER now that:
  - Control is implemented on same chip as processor
  - ROM is no longer faster than RAM
  - No need to go back and make changes

Historical Perspective

- In the ’60s and ’70s microprogramming was very important for implementing machines
- This led to more sophisticated ISAs and the VAX
- In the ’80s RISC processors based on pipelining became popular
- Pipelining the microinstructions is also possible!
- Implementations of IA-32 architecture processors since 486 use:
  - “hardwired control” for simpler instructions
  - (few cycles, FSM control implemented using PLA or random logic)
  - “microcoded control” for more complex instructions
  - (large numbers of cycles, central control store)
- The IA-64 architecture uses a RISC-style ISA and can be implemented without a large central control store

Pentium 4

- Pipelining is important (last IA-32 without it was 80386 in 1985)
- Pipelining is used for the simple instructions favored by compilers
  "Simply put, a high performance implementation needs to ensure that the simple instructions execute quickly, and that the burden of the complexities of the instruction set penalize the complex, less frequently used, instructions”
Pentium 4

- Somewhere in all that “control we must handle complex instructions
- Processor executes simple microinstructions, 70 bits wide (hardwired)
- 120 control lines for integer datapath (400 for floating point)
- If an instruction requires more than 4 microinstructions to implement, control from microcode ROM (8000 microinstructions)
- It's complicated!

Chapter 5 Summary

- If we understand the instructions...
  We can build a simple processor!
- If instructions take different amounts of time, multi-cycle is better
- Datapath implemented using:
  - Combinational logic for arithmetic
  - State holding elements to remember bits
- Control implemented using:
  - Combinational logic for single-cycle implementation
  - Finite state machine for multi-cycle implementation

Chapter Six

Pipelining

- Improve performance by increasing instruction throughput

Ideal speedup is number of stages in the pipeline. Do we achieve this?
**Pipelining**

- What makes it easy
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores

- What makes it hard?
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction

- We’ll build a simple pipeline and look at these issues

- We’ll talk about modern processors and what really makes it hard:
  - exception handling
  - trying to improve performance with out-of-order execution, etc.

**Basic Idea**

- What do we need to add to actually split the datapath into stages?

**Pipelined Datapath**

Can you find a problem even if there are no dependencies?
What instructions can we execute to manifest the problem?

**Corrected Datapath**
Graphically Representing Pipelines

- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths

Pipeline Control

- We have 5 stages. What needs to be controlled in each stage?
  - Instruction Fetch and PC Increment
  - Instruction Decode / Register Fetch
  - Execution
  - Memory Stage
  - Write Back

- How would control be handled in an automobile plant?
  - a fancy control center telling everyone what to do?
  - should we use a finite state machine?
Datapath with Control