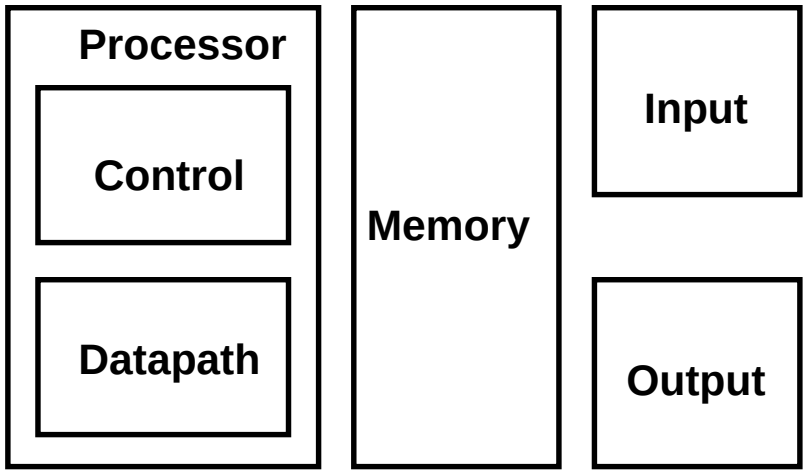
Computer Architecture
The Final Chapter

Spring, 2005

Sao-Jie Chen (csj@cc.ee.ntu.edu.tw)

The Big Picture

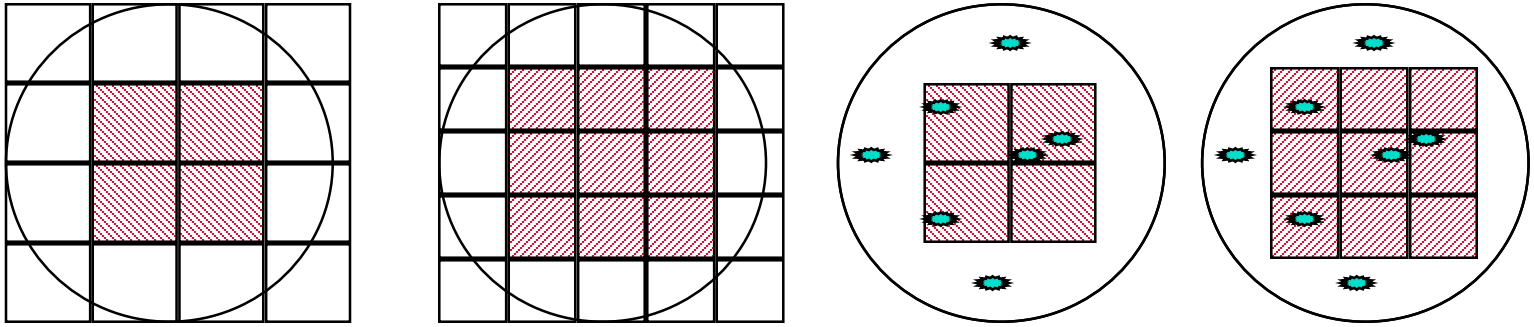
- Since 1946 all computers have had 5 components



Integrated Circuits Costs

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer-diam} / 2)^2}{\text{Die-Area}} - \frac{\pi * \text{Wafer-diam}}{\sqrt{2 * \text{Die-Area}}} - \text{Test dies} = \frac{\text{Wafer Area}}{\text{Die-Area}}$$



$$\text{Die Yield} = \text{Wafer yield} * \left\{ 1 + \frac{\text{Defects-per-unit-area} * \text{Die-Area}}{\alpha} \right\}^{-\alpha}$$

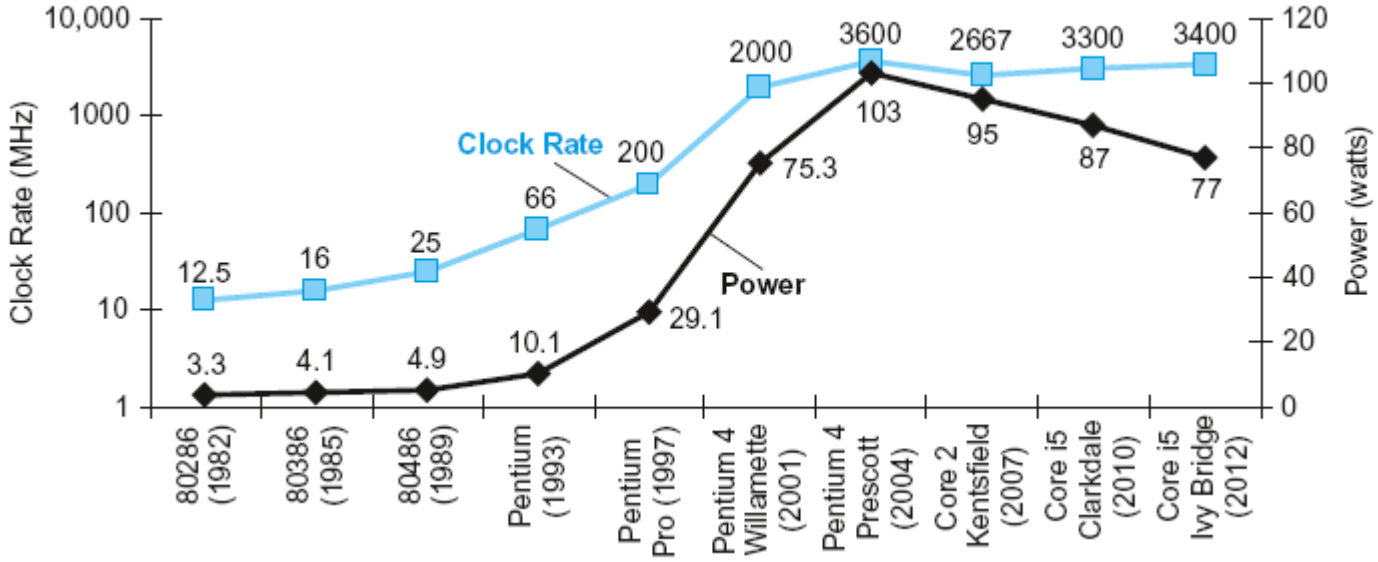
Die Cost goes up roughly with the cube of the area.

Performance Evaluation Summary

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Time is the measure of computer performance!
- Good products created when have:
  - ⇒ Good benchmarks
  - ⇒ Good ways to summarize performance
- If not good benchmarks and summary, then choice between improving product for real programs vs. improving product to get more sales=> sales almost always wins
- Remember Amdahl's Law: Speedup is limited by unimproved part of program

Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

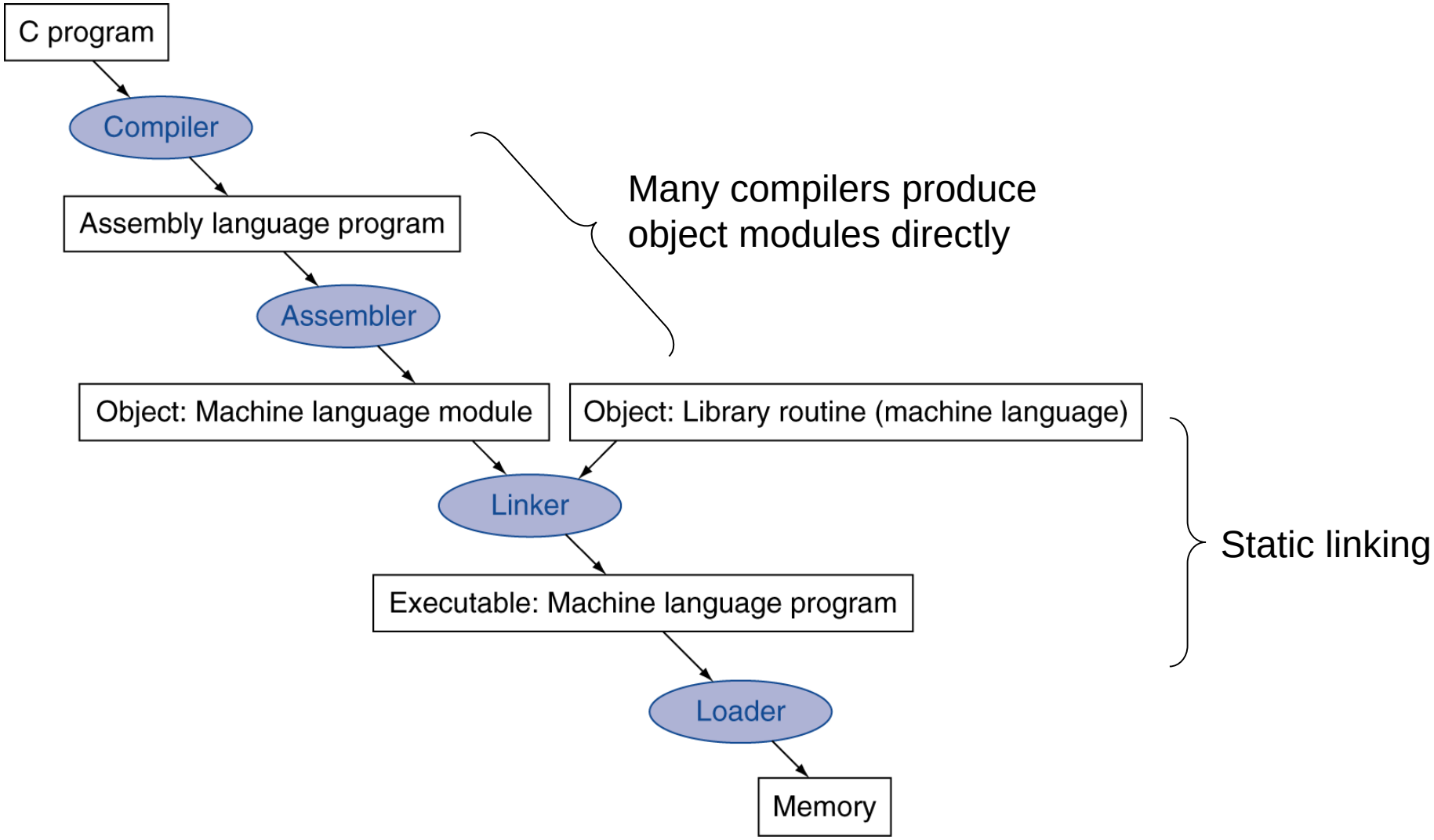
$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

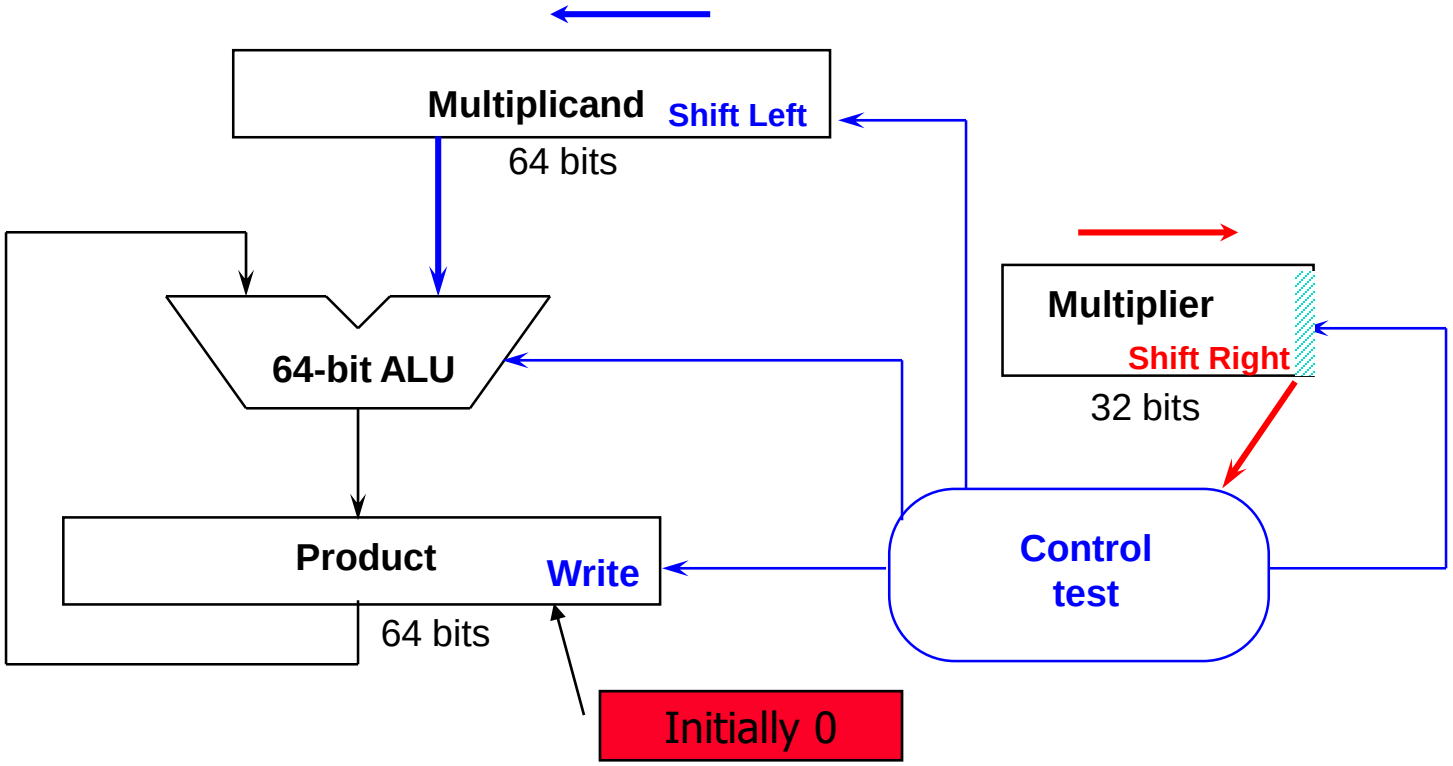
  - Can't be done!
- Corollary: make the common case fast

Translation and Startup



Multiplication Hardware

- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit Multiplier reg.



Booth's Algorithm

1. Depending on the current and previous bits, do one of the following:

   00: a. Middle of a string of 0s, so no arithmetic operations.

   01: b. End of a string of 1s, so add the multiplicand to the left half of the product.

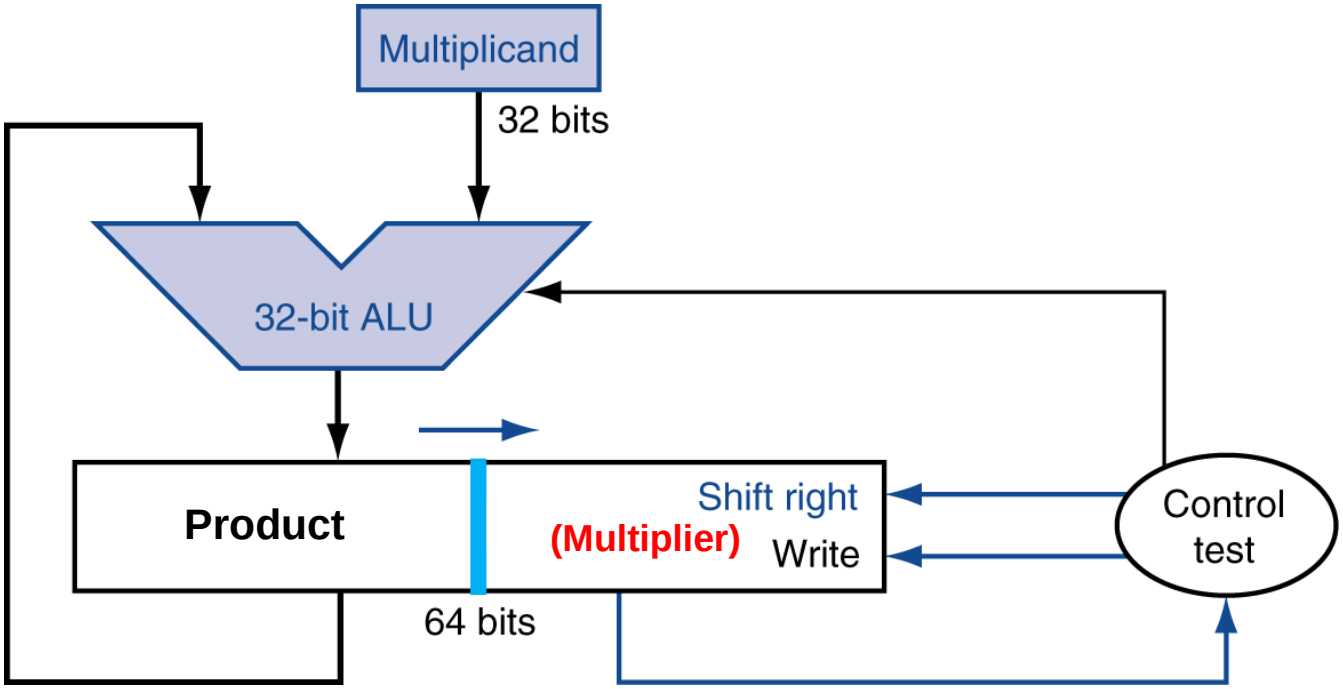
   10: c. Beginning of a string of 1s, so subtract the multiplicand from the left half of the product.

   11: d. Middle of a string of 1s, so no arithmetic operation.

2. As in the previous algorithm, shift the Product register right (arith) 1 bit.

Optimized Multiplier

- Perform steps in parallel: add/shift

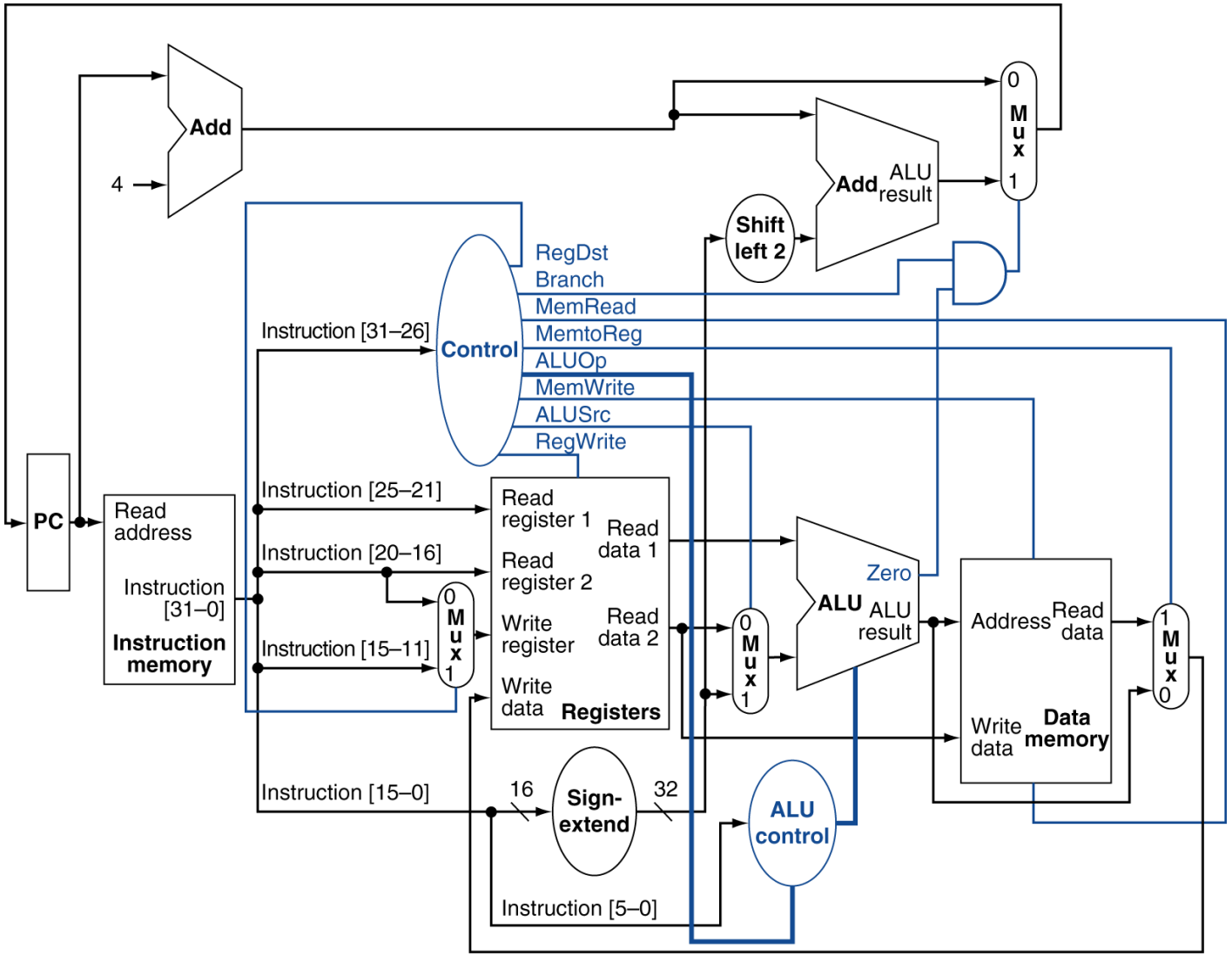


- One cycle per partial-product addition
  - That's ok, if frequency of multiplications is low

Arithmetic

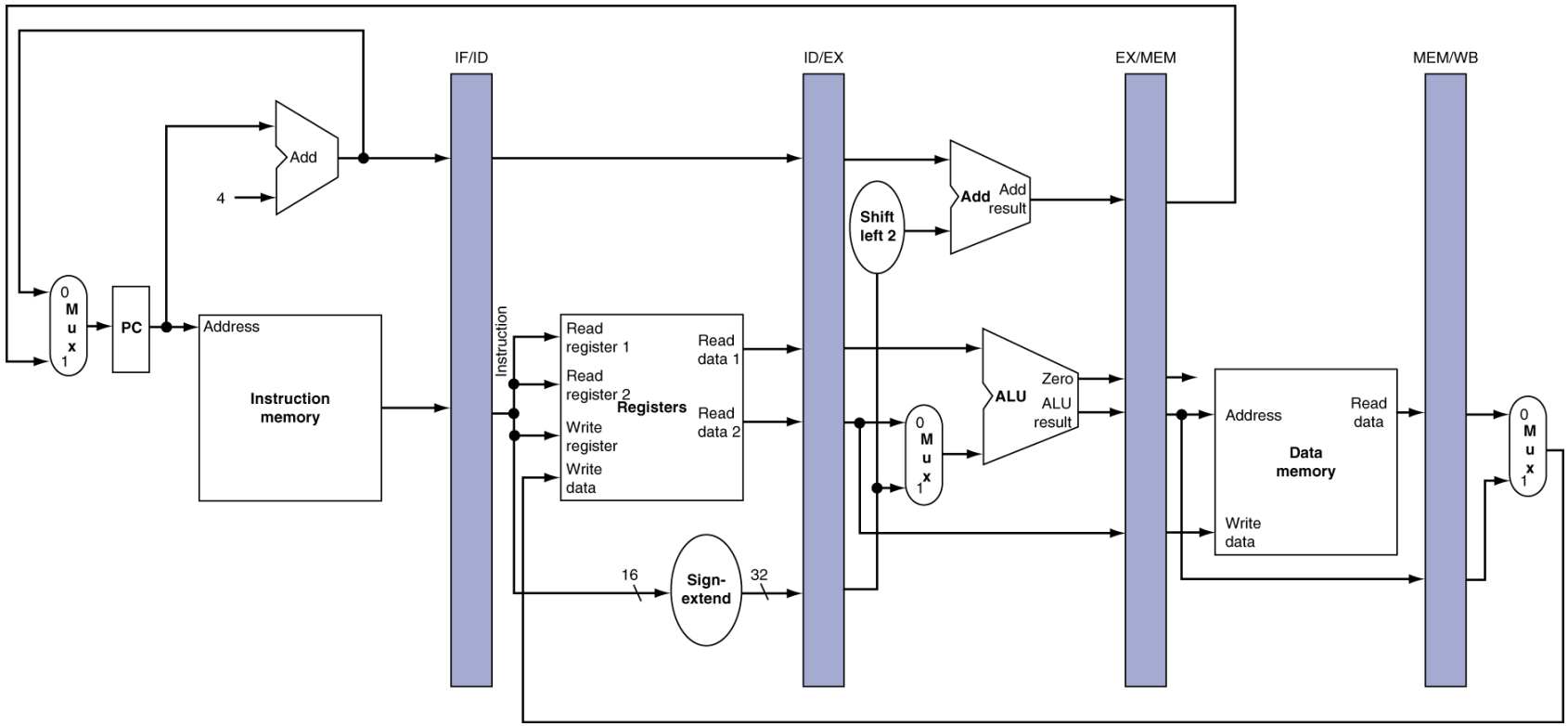
- Bits have no inherent meaning: operations determine whether they are really ASCII characters, integers, floating point numbers
- Divide can use same hardware as multiply: Hi & Lo registers in MIPS
- Floating point basically follows paper and pencil method of scientific notation using integer algorithms for multiply and divide of significands
- IEEE 754 requires good rounding; special values for NaN, Infinity
- Pentium: Difference between bugs that board designers must know about and bugs that potentially affect all users
  - ⇒ Why not make public complete description of bugs in later category?
  - ⇒ $200,000 cost in June to repair design
  - ⇒ $500,000,000 loss in December in profits to replace bad parts
  - ⇒ How much to repair Intel's reputation?
- What is technologists responsibility in disclosing bugs?

Datapath With Control



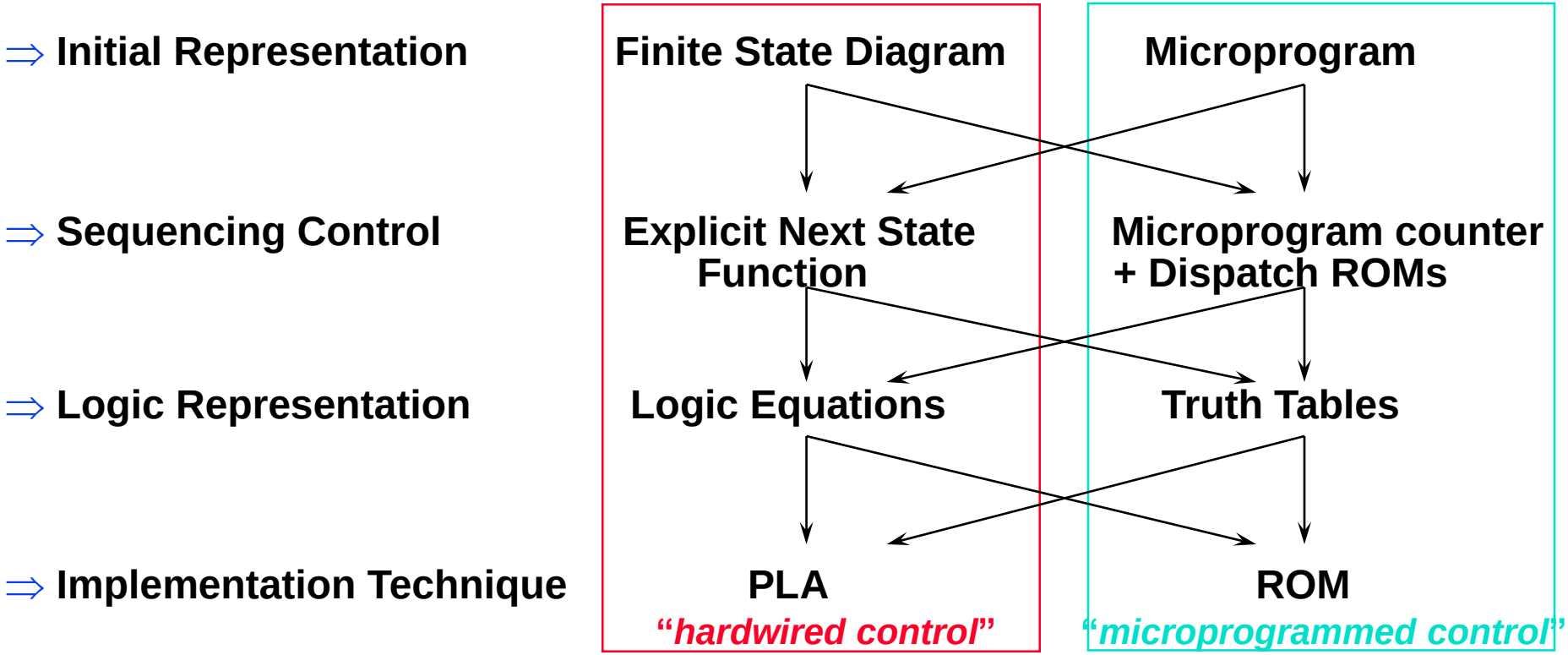
Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle



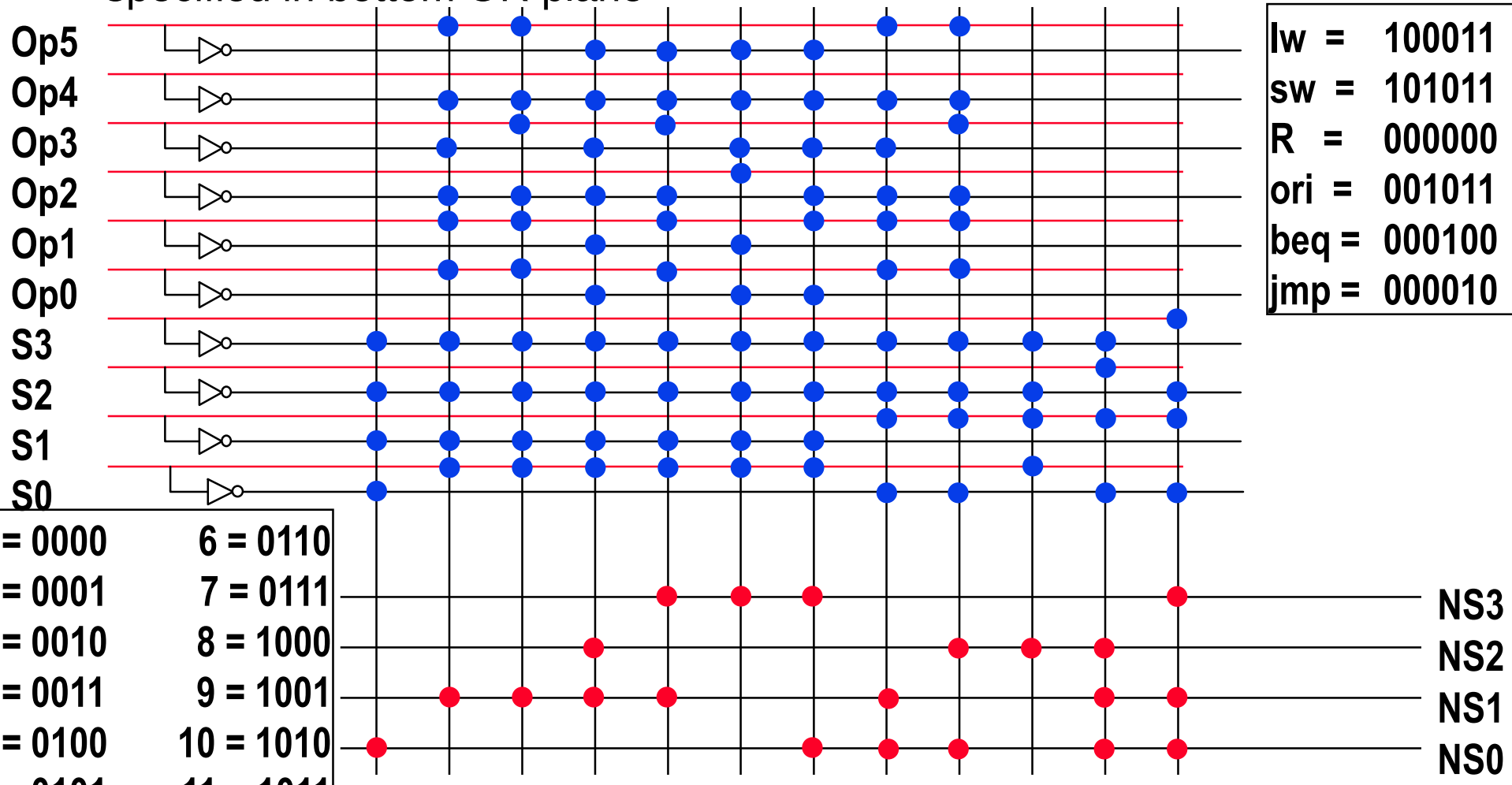
Control: Hardware vs. Microprogrammed

- **Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.**



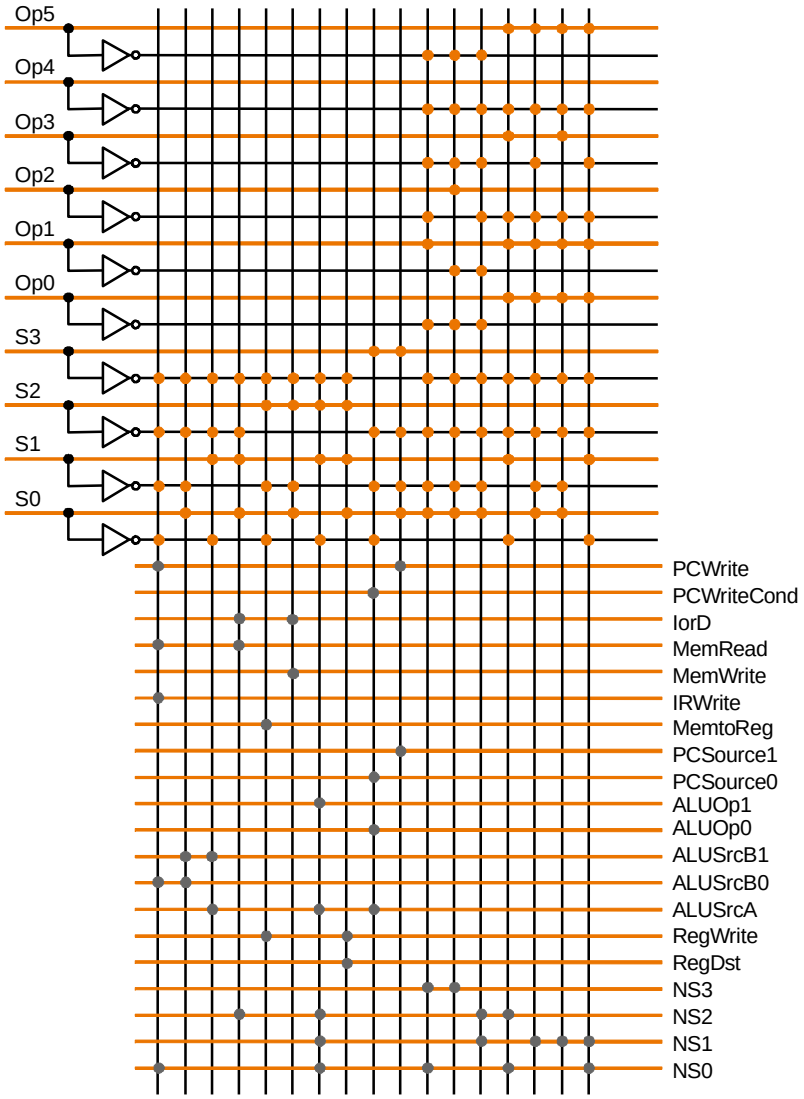
Implementation Technique: Programmed Logic Arrays

- Each output line the logical OR of logical AND of input lines or their complement: AND minterms specified in top AND plane, OR sums specified in bottom OR plane



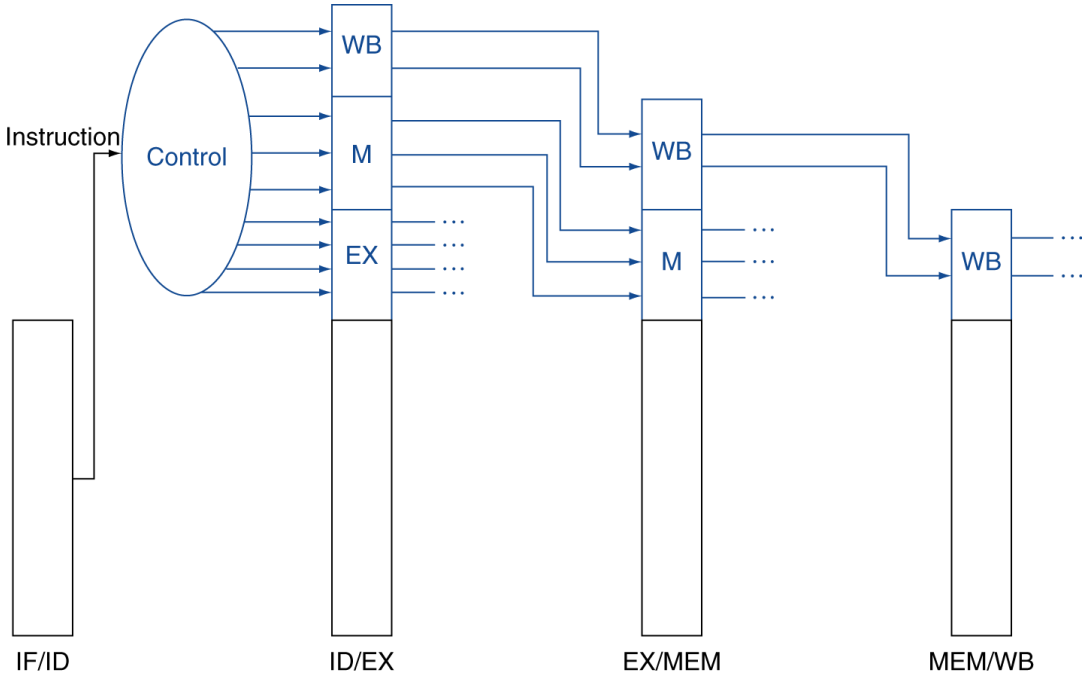
PLA Implementation

- If I picked a horizontal or vertical line could you explain it?



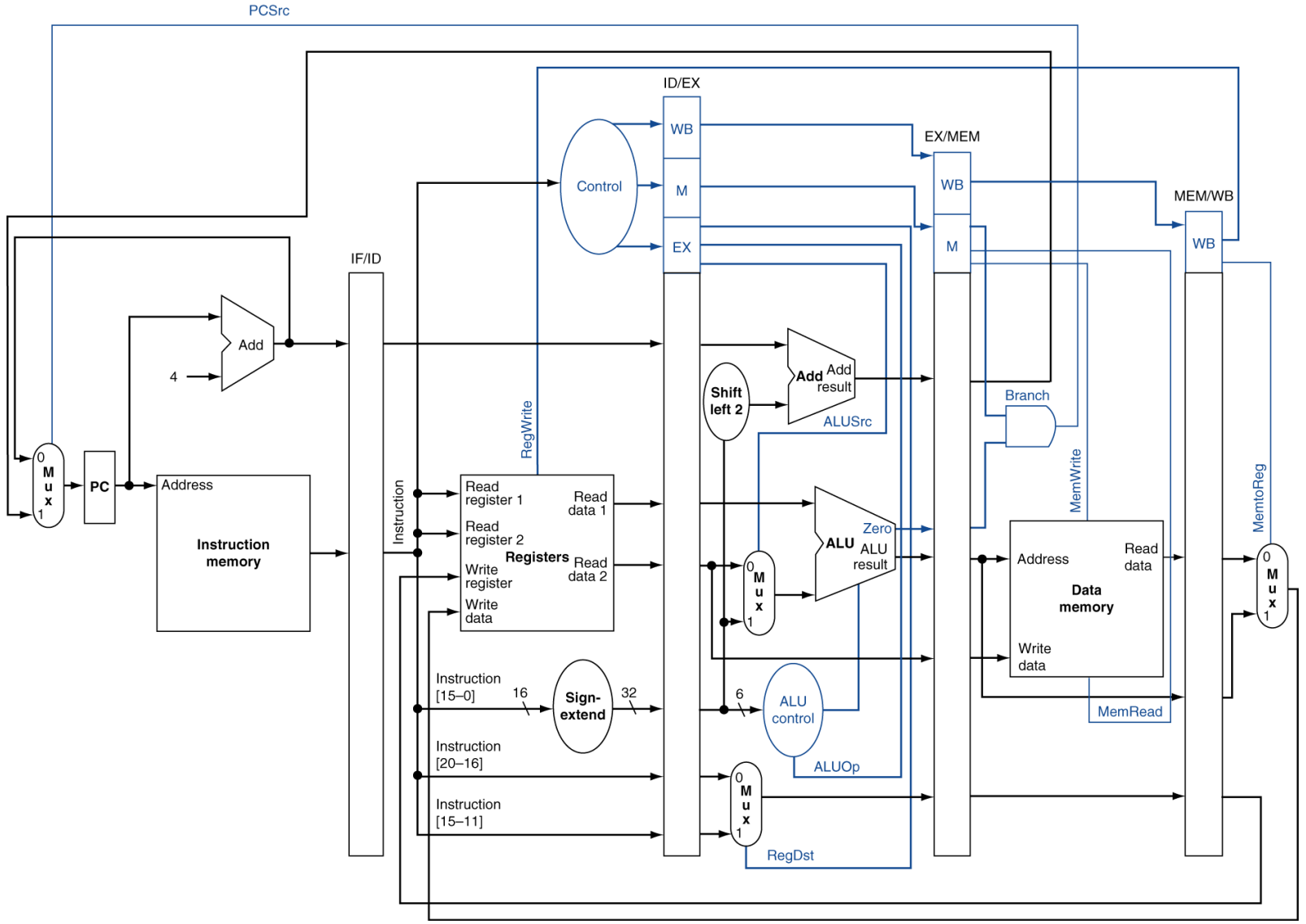
Pipelined Control

- Control signals derived from instruction

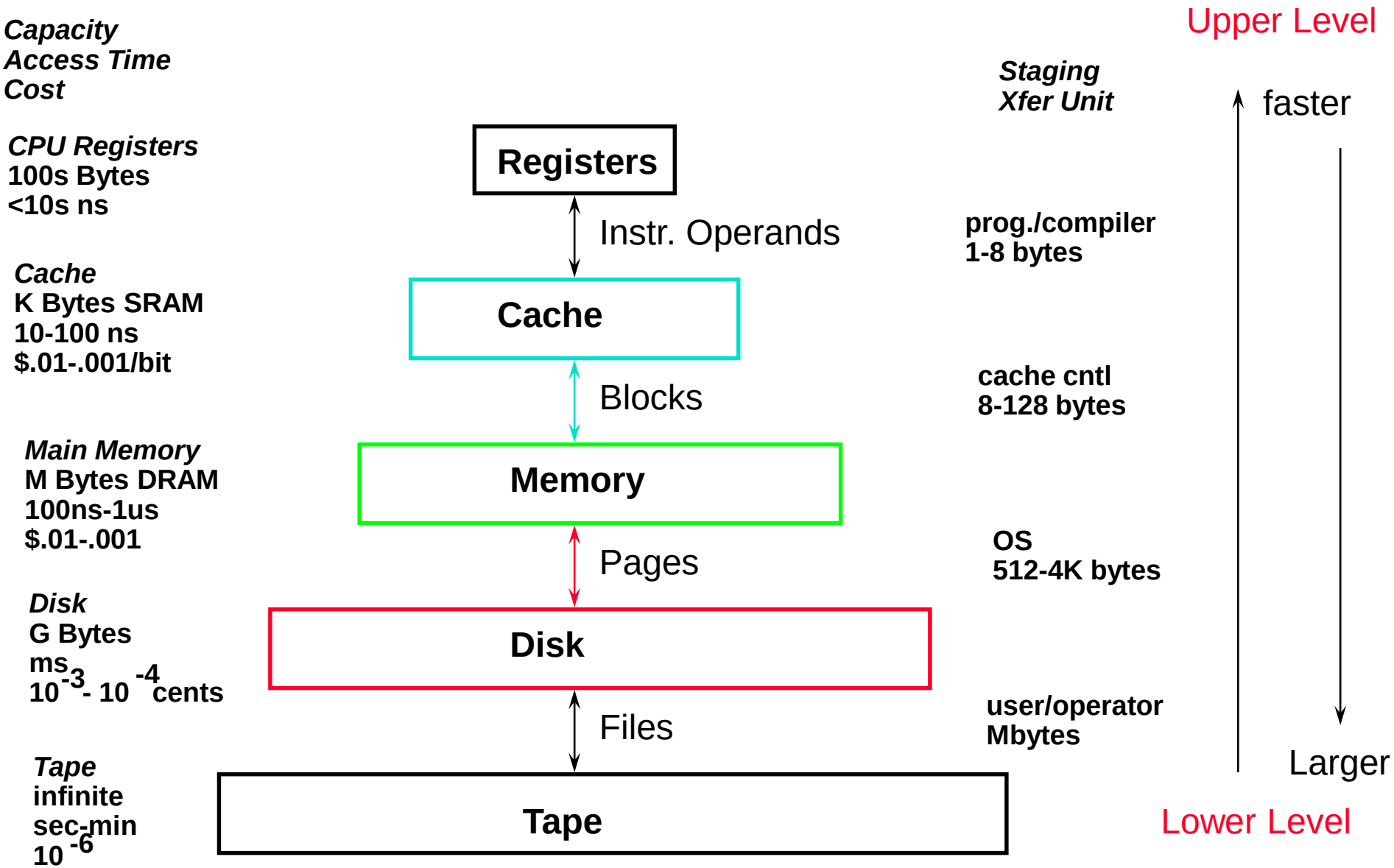


Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Pipelined Control



Levels of the Memory Hierarchy

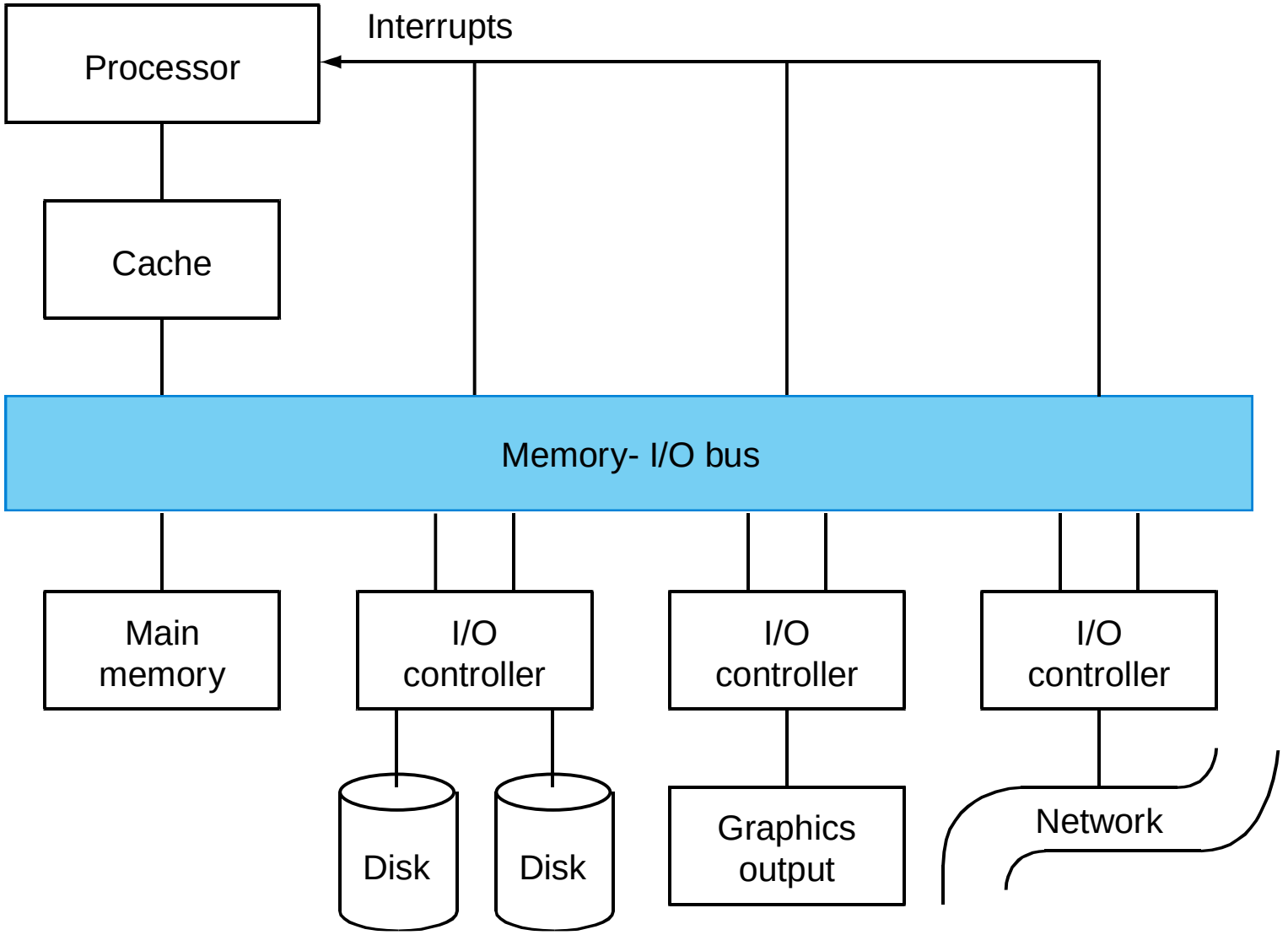


Memory Hierarchy

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - ⇒ Temporal Locality: Locality in Time
    - ⇒ Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
- Virtual Memory invented as another level of the hierarchy
  - Today VM allows many processes to share single memory without having to swap all processes to disk, protection more important
  - TLBs are important for fast translation/checking

I/O System Design Issues

- **Systems have a hierarchy of busses as well (PC: memory, PCI, ESA)**

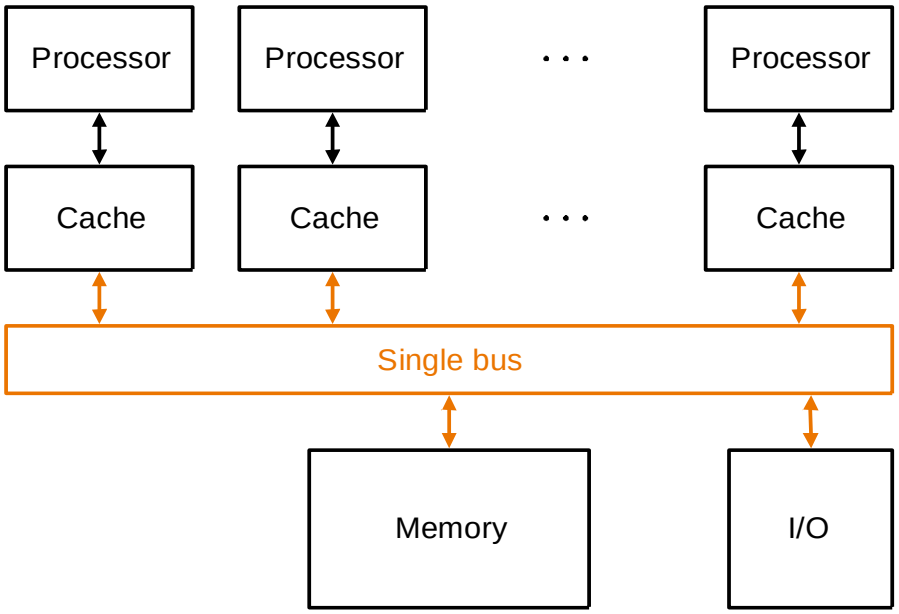


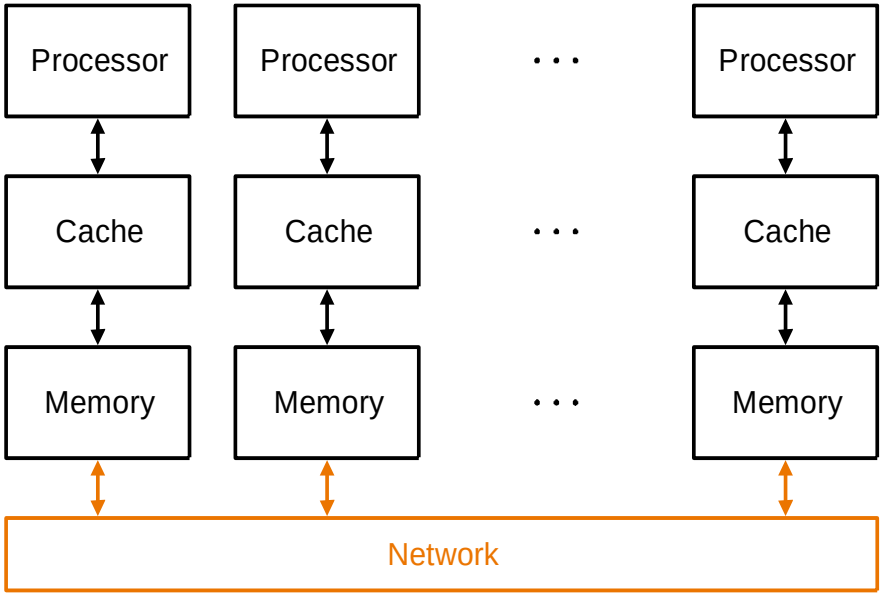
Multiprocessors

- **Idea: create powerful computers by connecting many smaller ones**

  ⇒ good news: works for timesharing (better than supercomputer)
  vector processing may be coming back

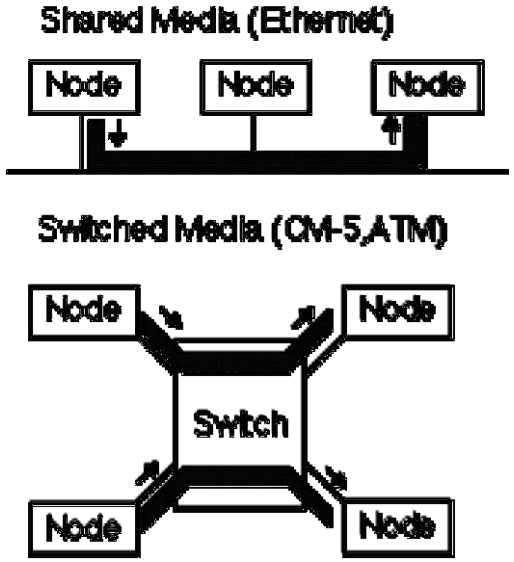
  ⇒ bad news: its really hard to write good concurrent programs
  many commercial failures



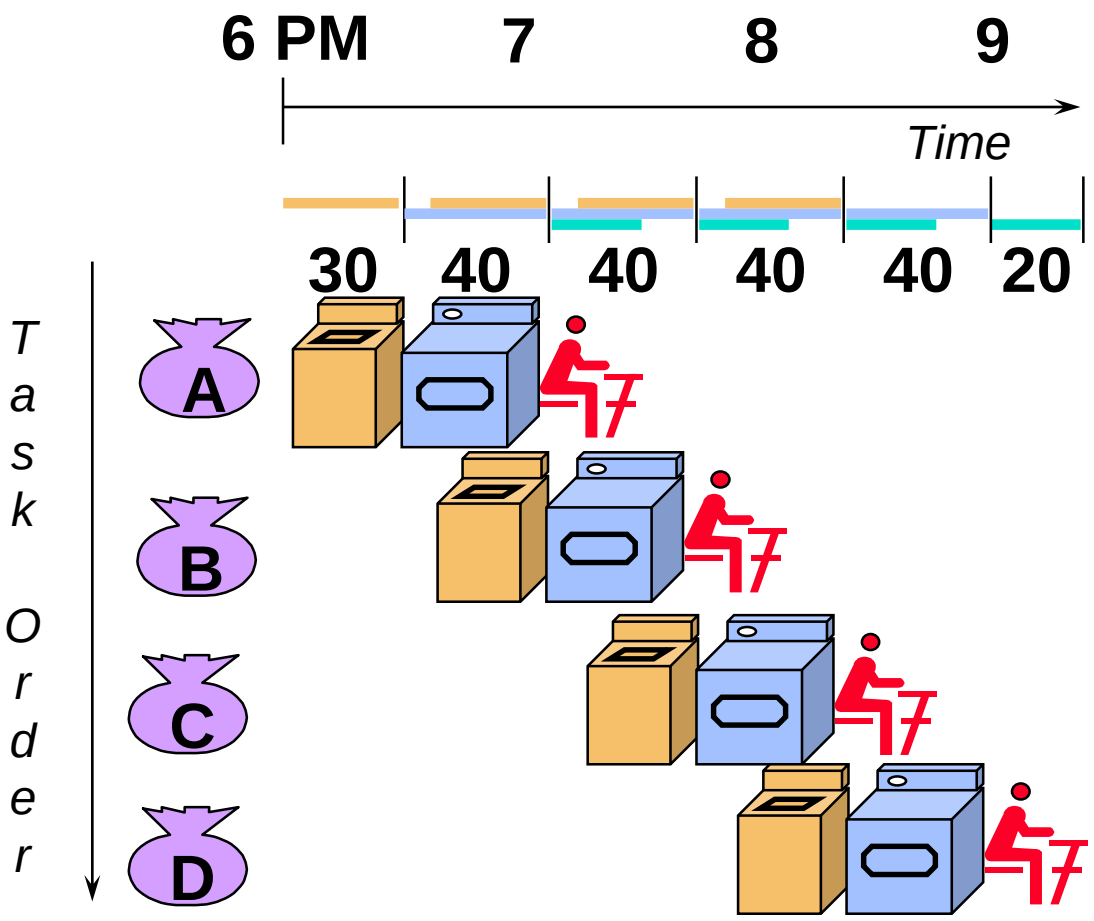


Interconnection Network Issues

- **Advantages of Serial vs. Parallel lines:**
  - **No synchronizing signals**
  - **Higher clock rate and longer distance than parallel lines. (e.g., 60 MHz x 256 bits x 0.5 m vs. 155 MHz x 1 bit x 100 m)**
    - ⇒ **Imperfections in the copper wires or integrated circuit pad drivers can cause skew in the arrival of signals, limiting the clock rate, and the length and number of the parallel lines.**
- **Switched vs. Shared Media: pairs communicate at same time: "point-to-point" connections**



Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

First Generation RISC Pipelines

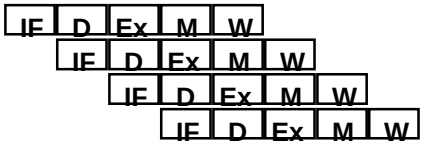
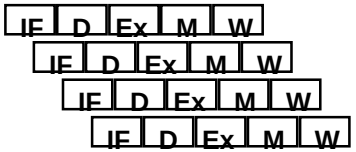
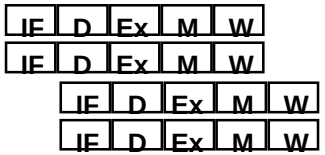
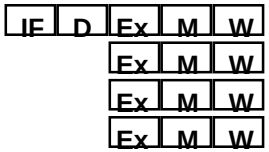
- All instructions follow same pipeline order (“static schedule”).
- Register write in last stage
  $\Rightarrow$ Avoid WAW hazards
- All register reads performed in first stage after issue.
  $\Rightarrow$ Avoid WAR hazards
- Memory access in stage 4
  $\Rightarrow$ Avoid all memory hazards
- Control hazards resolved by delayed branch (with fast path)
- RAW hazards resolved by bypass, except on load results
  which are resolved by fiat (delayed load).

Substantial pipelining with very little cost or complexity.

Machine organization is (slightly) exposed!

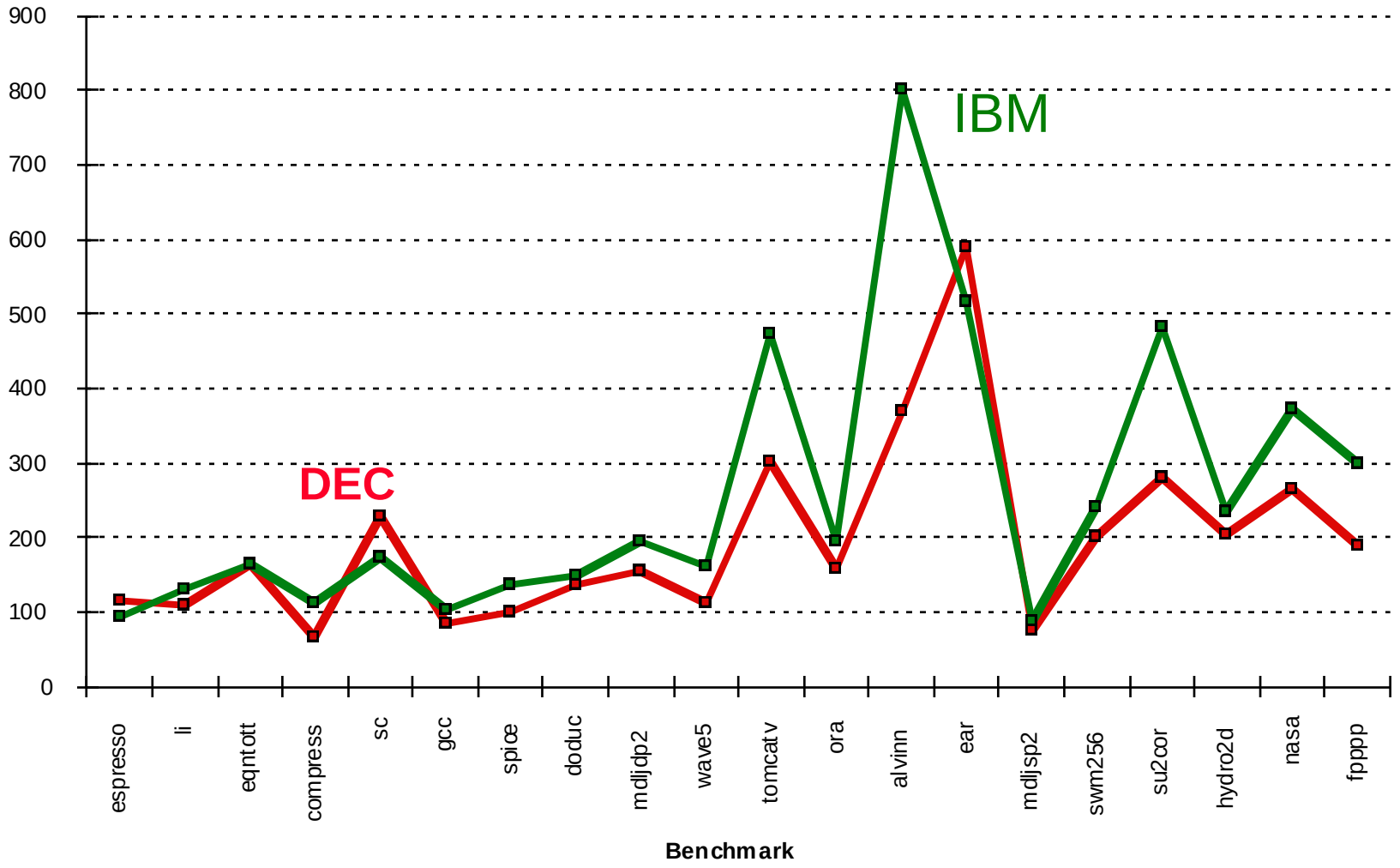
Relies very heavily on “hit assumption” of memory accesses in cache

How can the machine exploit available ILP?

Technique		Limitation
• Pipelining		Issue rate, FU stalls, FU depth
• Super-pipeline ⇒ Issue 1 instr. / (fast) cycle ⇒ IF takes multiple cycles		Clock skew, FU stalls, FU depth
• Super-scalar ⇒ Issue multiple scalar instructions per cycle		Hazard resolution
• VLIW ⇒ Each instruction specifies multiple scalar operations		Packing

Braniac vs. Speed Demon (1994)

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe)
  vs. 2-scalar DEC Alpha @ 200 MHz (7 stage pipe)



Performance Retrospective

- Theory of Algorithms & Compilers based on number of operations
- Compiler remove operations and “simplify” ops:
  Integer adds << Integer multiplies << FP adds << FP multiplies
  - ⇒ Advanced pipelines => these operations take similar time
- As Clock rates get higher and pipelines are longer, instructions take less time but DRAMs only slightly faster (although much larger)
- Today time is a function of (ops, cache misses);
  FP multiply faster than integer multiply
- Given importance of caches, what does this mean to:
  - ⇒ Compilers?
  - ⇒ Data structures?
  - ⇒ Algorithms?
- How do you tune performance on R10000?