
CSE 220 Computer Organization

Lecture 22

Hussein Badr

Computer Science, Stony Brook University

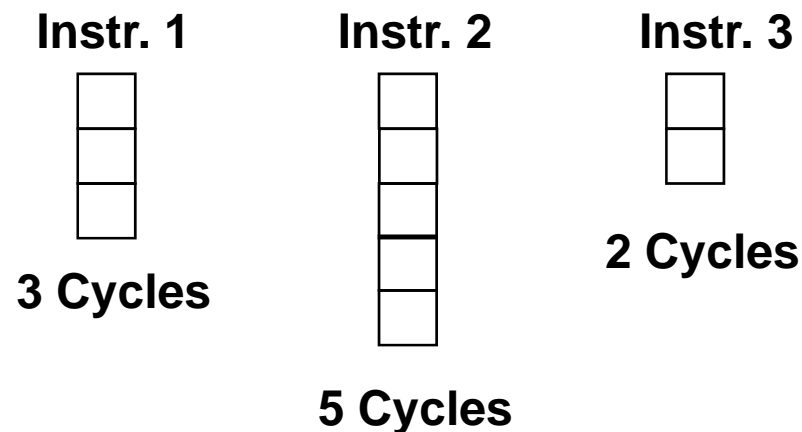
<http://www.cs.sunysb.edu/~cse220>

Processor Cycle (1/2)

- **Another important issue is the processor cycle.**
- **Do we go for a single cycle or a multicycle implementation?**
- **Single cycle: Each instruction is completely executed in one cycle of the processor.**
- **Here the duration of the cycle must be long enough to accommodate every possible instruction.**
- **Obviously inefficient.**
- **Some instructions are done quickly, while others require a longer time.**
- **So time is wasted when executing simple instructions that are done quickly.**

Processor Cycle (2/2)

- A multicycle implementation breaks down each instruction into subtasks or stages.
- Each subtask is done in one (short) processor cycle.
- Every MIPS instruction requires one or more cycles.
- This is efficient because not much time is wasted.



- In a single-cycle implementation, the cycle must be long enough to accommodate the longest instruction, Instr. 2.

Microprogramming (1/5)

- **Multicycle implementation.**
- **How to simplify the control?**
- **One approach is to use microprogramming.**
- **Multicycle implementation is the first step in the right direction.**
- **Here, each MIPS instruction may require a few (more than 1) CPU cycles.**
- **What has been done is that one MIPS instruction has been split into successive microinstructions / stages.**
- **CPU executes one microinstruction per cycle.**

Microprogramming (2/5)

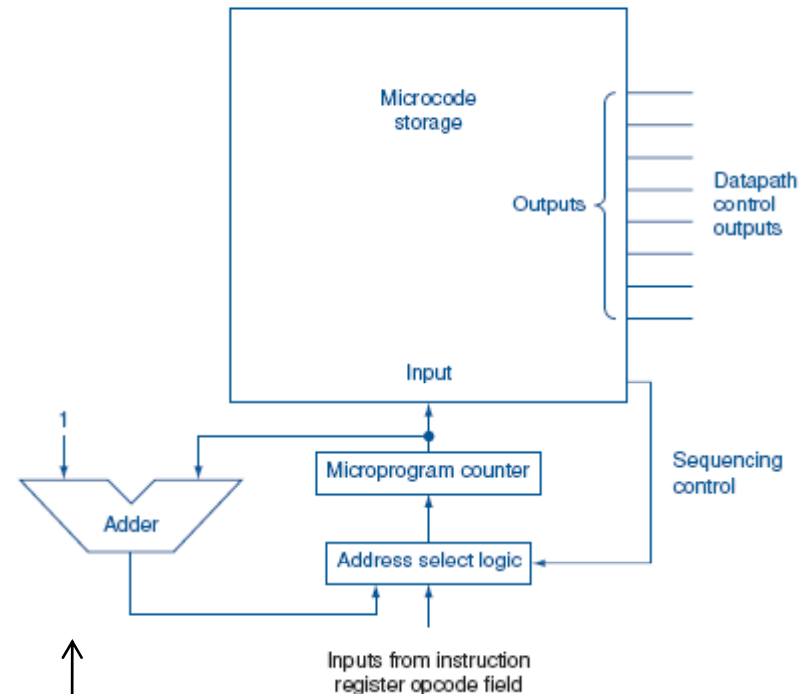
- **During the execution of a microinstruction, the CPU does only a part of the original MIPS instruction.**
- **What is a microinstruction?**
- **It consists of several control signals. If a particular control bit is 1 then it controls the corresponding part of the MIPS datapath.**
- **What is the size of a microinstruction?**
- **It depends on how many control bits we want.**
- **For each MIPS instruction, we have its microprogram.**
- **Execution of an instruction means executing its microprogram.**

Microprogramming (3/5)

- **Where do we store the microprograms?**
- **In a 'control store' – this is a ROM. Why?**
- **A microprogram never changes. The control store is written when the processor chip is made. There is no need to rewrite it.**
- **A given instruction is always executed in the same way.**
- **Sequencing of microinstructions.**
- **How do we get the first microinstruction?**
- **The opcode usually provides the address of the first microinstruction in the corresponding microprogram.**

Microprogramming (4/5)

Figure D.4.6 A typical implementation of a microcode controller would use an explicit incrementer to compute the default sequential next state and would place the microcode in a read-only memory. The microinstructions, used to set the datapath control, are assembled directly from the microprogram. The microprogram counter, which replaces the state register of a finite state machine controller, determines how the next microinstruction is chosen. The address select logic contains the dispatch tables as well as the logic to select from among the alternative next states; the selection of the next microinstruction is controlled by the sequencing control outputs from the control logic. The combination of the current microprogram counter, incrementer, dispatch tables, and address select logic forms a sequencer that selects the next microinstruction. The microcode storage either may consist of read only memory (ROM) or may be implemented by a PLA. PLAs may be more efficient in VLSI implementations, while ROMs may be easier to change.



- 'add 1' to fetch the next microinstruction

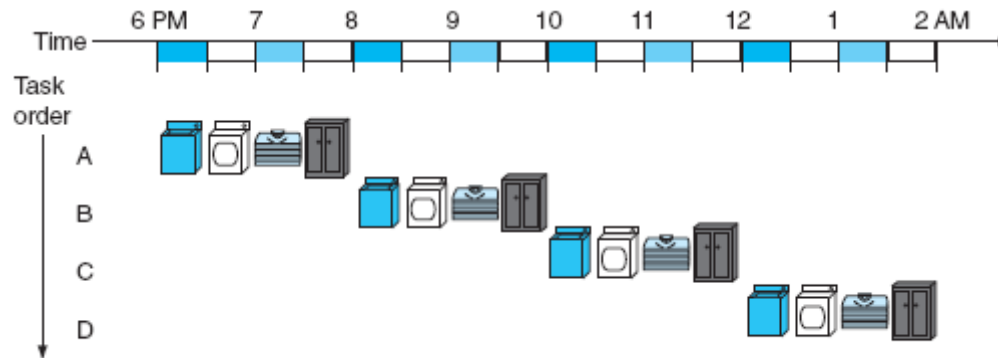
Microprogramming (5/5)

- **A microinstruction also carries some explicit information about the next microinstruction (address).**
- **This is the sequencing control part.**
- **Sometimes we add 1 to obtain the next address, but not always.**
- **We also have a microprogram counter (MPC).**
- **It stores the address of the next microinstruction.**
- **Many (MIPS) instructions may share some microcode.**
- **Using explicit sequencing control keeps the size of the ROM small.**

Pipelined Execution (1/8)

- **What we have seen so far is a very simplified approach to the execution of MIPS instructions.**
- **We fetch an instruction, decode it, and execute it completely.**
- **Then we start with the next instruction.**
- **Only one instruction is handled at a time by the CPU.**
- **But our datapath has several parts/components.**
- **When one part is in use, the other parts remain idle.**
- **Is it possible to handle several instructions at a time?**

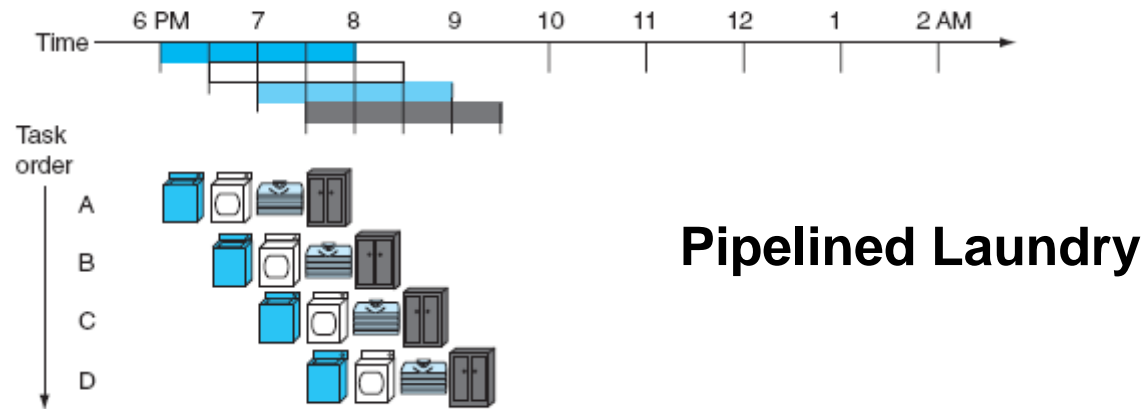
Pipelined Execution (2/8)



Laundry Example

- **There are 4 steps:**
 - a. Load the washer
 - b. Clothes from washer to dryer
 - c. Fold laundry
 - d. Store laundry in closet
- **In this example, four loads are done.**
- **We do one load completely, and only then start with the next load.**

Pipelined Execution (3/8)



- Here, we do not wait for the first load to finish completely.
- Between 7:30-8 PM all 4 loads are being done.
- Each is at a different stage.
- Total time required is almost half as before.
- Once the 'pipeline' is full (when the fourth load – Task D – starts), we can do the laundry four times as fast.

Pipelined Execution (4/8)

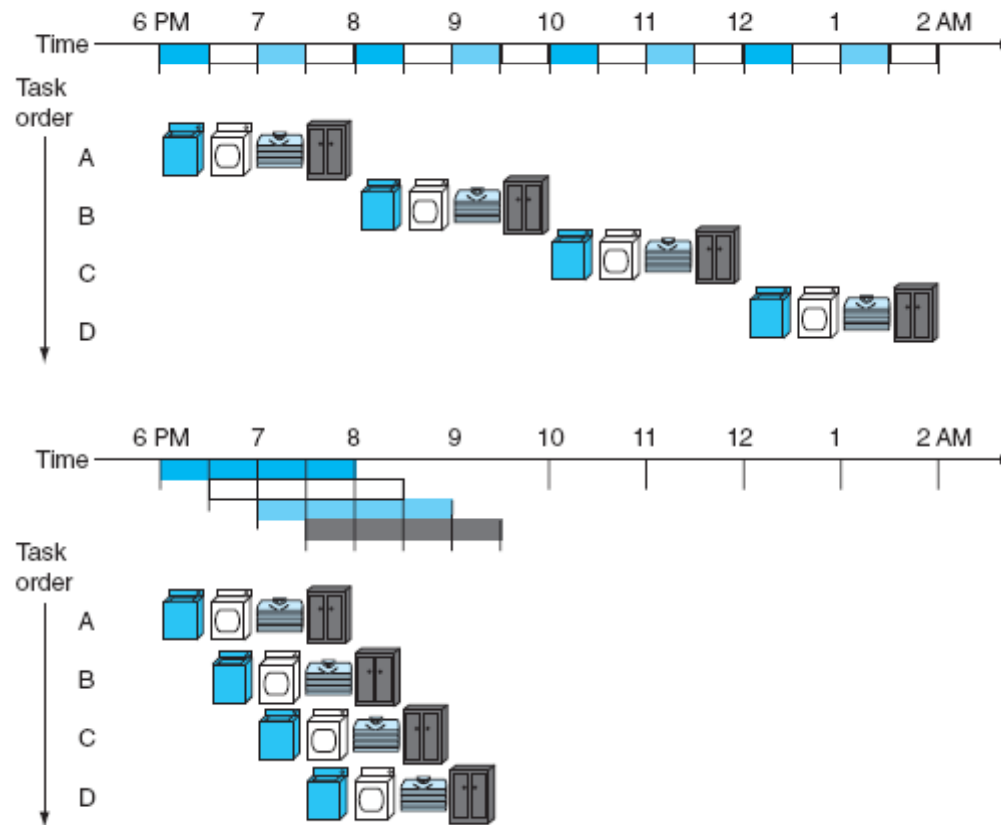
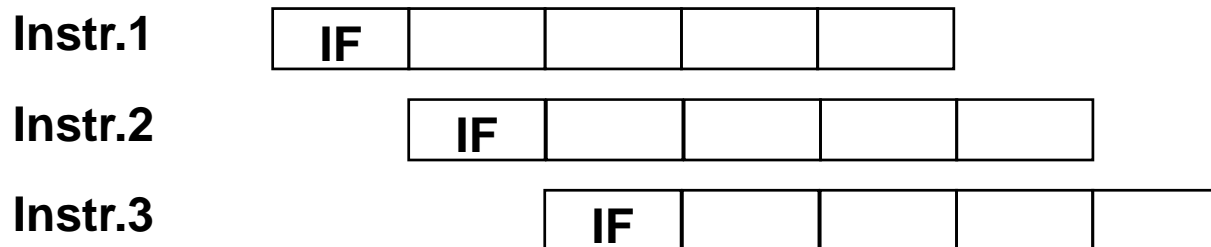


FIGURE 4.25 The laundry analogy for pipelining. Ann, Brian, Cathy, and Don each have dirty clothes to be washed, dried, folded, and put away. The washer, dryer, “folder,” and “storer” each take 30 minutes for their task. Sequential laundry takes 8 hours for four loads of wash, while pipelined laundry takes just 3.5 hours. We show the pipeline stage of different loads over time by showing copies of the four resources on this two-dimensional time line, but we really have just one of each resource.

Pipelined Execution (5/8)

- We have already thought about multicycle implementations.
- One possible break-up of tasks in a MIPS instruction could be
 - a. Instruction fetch (IF)
 - b. Instruction decode & register file read (ID)
 - c. Execution of arithmetic operation or address calculation (EX)
 - d. Data memory access (MEM)
 - e. Write back to register (WB)



Pipelined Execution (6/8)

- **With some extra hardware for control and the storing of intermediate values, we can get a lot of work done.**
- **Maximum speed of the CPU will depend on how many stages we have.**
- **More stages → more speed.**
- **But this is a rather simplistic view.**
- **Not all instructions require all stages.**
- **Furthermore, each stage may not take exactly the same amount time.**
- **These issues complicate the design and overall speed.**
- **How do we handle (conditional) branches? This is the main problem.**

Pipelined Execution (7/8)

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

FIGURE 4.26 Total time for each instruction calculated from the time for each component. This calculation assumes that the multiplexors, control unit, PC accesses, and sign extension unit have no delay.

Pipelined Execution (8/8)

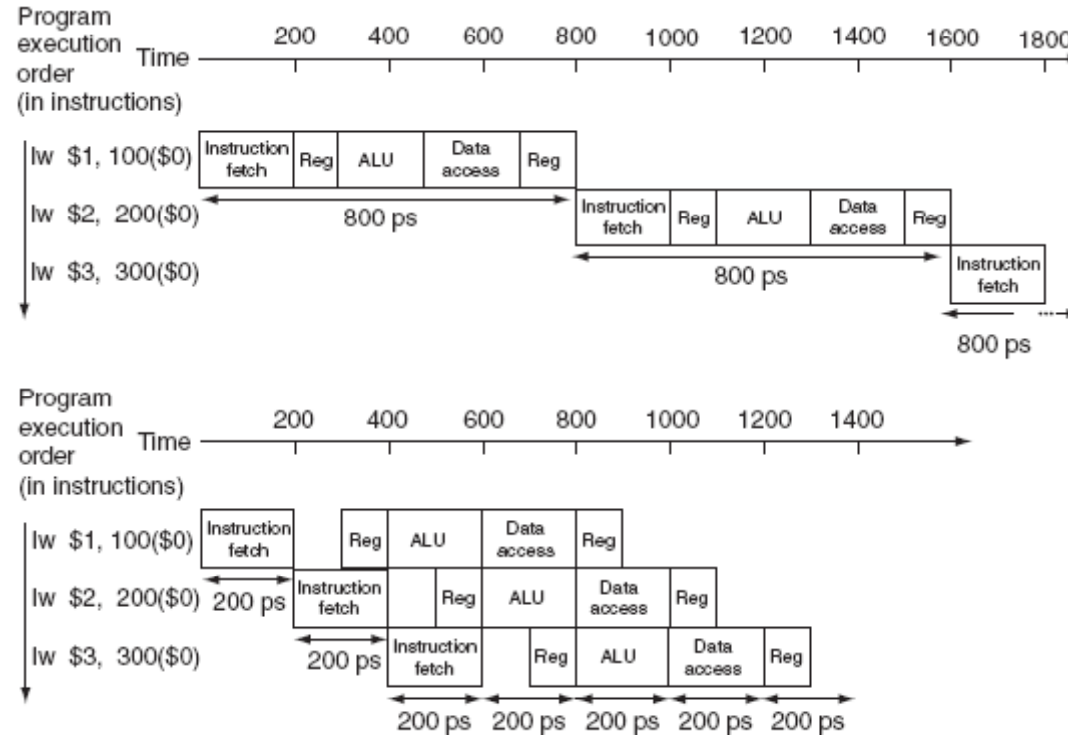


FIGURE 4.27 Single-cycle, nonpipelined execution in top versus pipelined execution in bottom. Both use the same hardware components, whose time is listed in Figure 4.26. In this case we see a fourfold speedup on average time between instructions, from 800 ps down to 200 ps. Compare this figure to Figure 4.25. For the laundry, we assumed all stages were equal. If the dryer were slowest, then the dryer stage would set the stage time. The computer pipeline stage times are limited by the slowest resource, either the ALU operation or the memory access. We assume the write to the register file occurs in the first half of the clock cycle and the read from the register file occurs in the second half. We use this assumption throughout this chapter.