System Hardware

Tutorial 7 RISC Pipelining

The Need for Speed

- Computer processor (CPU) speed is one of the most important criteria when comparing computers.
- Faster computers often translate into increased productivity and efficiency.
- To improve CPU performance, we have two options:
  1- Improve the hardware by introducing faster circuits.
  2- Arrange the hardware such that more than one operation can be performed at the same time.
- Since, there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option.

Introduction to Instruction Pipelining

- **Instruction pipelining** is a technique used in the design of modern microprocessors and CPUs in order to increase their ***instruction throughput*** (the number of instructions that can be executed in a unit of time).
- Instruction pipelining implements instruction-level parallelism within a single processor.
- Pipelining attempts to keep every part of the processor busy with some instruction by dividing instructions into a series of sequential steps performed by different processor components such that different parts from different instructions are being processed in parallel.

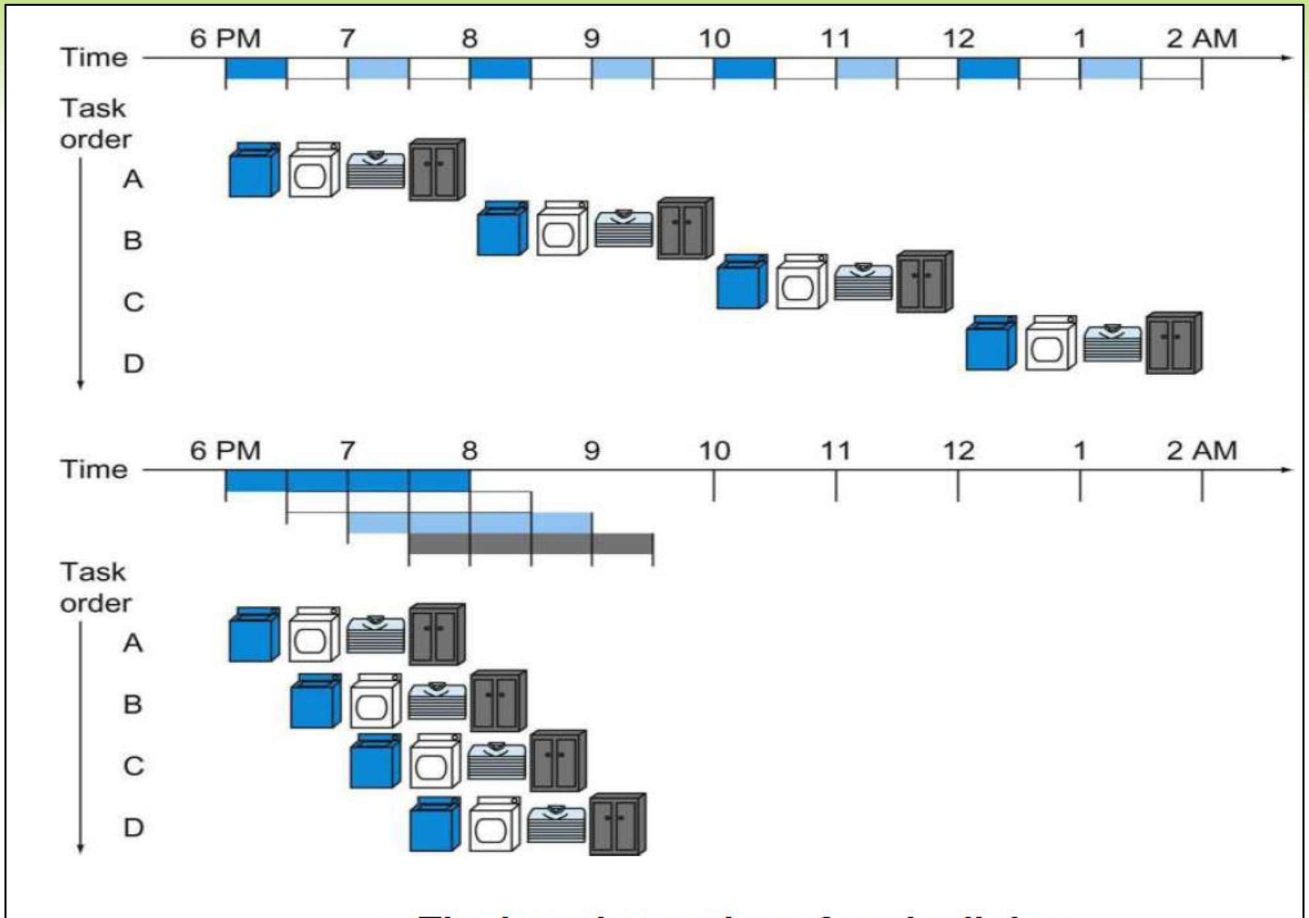
Pipelined Laundry - An Illustration

- Start by putting a load of laundry in the washer
- As soon as the washer is done with the 1st load, place it in the dryer, and load the washer with the 2nd dirty load.
- When the 1st load is dry, place it on a table to fold, move the wet load to the dryer, and put the next dirty load into the washer.
- When the 1st load is folded, get someone to put the 1st load away, start folding the second load, the dryer has the third load, and you put the fourth load into the washer.

Pipelined Laundry - An Illustration

- At this point all steps—called *stages* in pipelining—are operating concurrently. As long as we have separate resources for each stage, we can pipeline the tasks.
- Note that the time from placing a single dirty sock in the washer until it is dried, folded, and put away is not shorter for pipelining;
- The reason that pipelining is faster for many loads is that everything is working in parallel, so more loads are finished per hour.
- Pipelining improves the overall *throughput* of our laundry system.

Pipelined Laundry - An Illustration



The laundry analogy for pipelining.

Source Patterson & Hennessy, Computer Organization and Design, Morgan Kaumann, RISC-V Edition (Sixth Edition), 2018

Introduction to Instruction Pipelining

- When looking at microprocessor history starting from the 1980s, it can be seen that many processor architectures died off. E.g. VAX machines.
- The reason that it was very hard to implement fast pipelining on computers with a complex instruction set.
- In 1980, the idea of RISC (reduced instruction set computing) architecture gained ground, and led to radical simplifications in the implementation of pipelining.

Introduction to Instruction Pipelining

- In the 1970s, machines had hundreds of different instructions, in a variety of formats, leading to thousands of distinct combinations when addressing-mode variations were taken into account.
- Interpretation of all possible combinations of opcodes and operands required very complex control circuitry.
- RISC machines were successful because they radically reduced the amount of control circuitry, i.e., they reduced the ***control overhead***.
- RISC machines allowed a fully hardwired implementation of control circuitry, breaking the tradition of microprogrammed control and dramatically increasing execution speed.

The RISC Philosophy For Instruction Set Design

- A small set of instructions, that can be each executed in approximately the same amount of time using hardwired control (you may need several RISC instructions to do the work of one complex instruction).
- A ***load/store architecture*** that confines memory-address calculation, and memory-latency delays, to a small set of instructions.
- All other (register-register) instructions obtain their operands from faster, and compactly addressable, processor registers.
- A limited number of simple addressing modes that eliminate or speed up address calculations for the vast majority of cases.
- Simple, uniform instruction formats that facilitate extraction and decoding of the various fields. This allows overlap between opcode interpretation and register readout.

The RISC Architecture Summary

- All operations on data apply to the data in registers.
- The only operators that affect memory are loads (which move data from memory to a register) and stores (which move data from a register to memory).
- The instruction formats are few in number, with all instructions typically being one size.
- Examples of RISC Architecture: MIPS, ARM, PowerPC, PA-RISC, SPARC

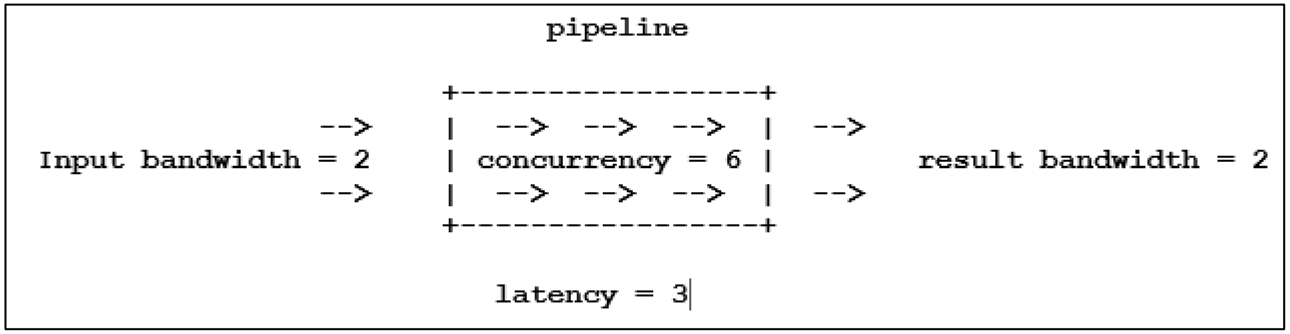
Circuitry in RISC Microarchitecture

- The processor is logically composed of two main types of components: Control and datapath.
- ***Active Control Circuitry*** that tells the datapath, memory, and I/O devices what to do according to the wishes of the instructions of the program.
- ***Passive datapath circuitry*** that follows orders. E.g. The **datapath** performs the arithmetic operations.
- Over time, many control functions were incorporated (delegated) into the datapath.
- **We will not be emphasizing the** distinction between datapath and control because it is nearly impossible to say where the datapath ends and the control circuitry begins.

Basic Pipeline Characteristics

A pipeline is characterized by three parameters:

- **Peak Input Bandwidth**: The maximum rate the pipeline will tolerate.
- **Operation Latency**: The time to complete an operation.
- **Pipeline Occupancy (Concurrency)**: The number of uncompleted operations in the pipeline at any time.



```
                                  pipeline

                          +------------------+
                   -->    |  -->  -->  -->   |   -->
Input bandwidth = 2       | concurrency = 6  |         result bandwidth = 2
                   -->    |  -->  -->  -->   |   -->
                          +------------------+

                              latency = 3
```

Basic Pipeline Example

- Pipelines take time to reach their equilibrium state.
- When we start by feeding operations into an empty pipeline, we need to wait until we get our first result.
- It is only after the pipeline has reached its equilibrium state that the result bandwidth on the right will exactly match the input bandwidth on the left.
- The sustained input bandwidth is 2 operations per cycle
- After equilibrium, the sustained output will be 2 results per cycle
- **Little's Law**: at equilibrium, concurrency = latency x bandwidth

The MIPS 'fdxmw' Pipeline

- The MIPS **fdxmw** instruction-execution pipeline is a special case of the general pipeline.
- It is divided into five stages (or "boxes"), each of which performs a well-defined task.
- At equilibrium:
  - The input bandwidth is 1,
  - The output bandwidth is 1,
  - The latency is 5,
  - The occupancy (concurrency) is 5.

The MIPS Pipeline Characteristics

```
+-------------------------------------------------------------------------+ P
|     <I-cache>               <Register file>               <D-cache>     | r
|                                                                         | o
|                             <Control circuitry>                         | c
|              +-+           +-+           +-+           +-+              | e
|     <f-box>  | |  <d-box>  | |  <x-box>  | |  <m-box>  | |  <w-box>     | s
|              +-+           +-+           +-+           +-+              | s
|     .PC.     f/d           d/x           x/m           m/w              | o
+-------------------------------------------------------------------------+ r
```

- The rectangular boxes, called "latches" use sequential logic and are collections of state elements.
- Data flows, from left to right, through boxes and latches.
- At the top of the processor, we see hardware resources. E.g. I-cache, Register file, D-cache.
- In addition to the normal pipeline flow (left to right), there is flow of data between pipeline boxes and hardware resources.

The MIPS Pipeline Characteristics

- Each box must complete its work, including all latching, in one clock cycle.
- The memory is viewed as an array of bytes.
- **State Elements**: These elements are not combinational, but instead contain *state*. An element contains state if it has some internal storage. We call these elements **state elements** because, if we pulled the power plug on the computer, we could restart it accurately by loading the state elements with the values they contained before we pulled the plug. Examples Instruction and data storage, registers.

The MIPS 'fdxmw' Pipeline Boxes

- The next slides will describe the 5 MIPS boxes.

The MIPS f-box Fetch

- Reads the memory byte address of the next instruction from the PC register and fetches it from memory. It then updates PC by adding 4.
- The I-cache intercepts the memory request and, if it has a copy of the instruction, delivers the instruction to the f-box. 32 bits of address travel up to the I-cache, and 32 bits of instruction travel down to the f-box.
- Before the cycle completes, the f-box latches the fetched instruction in the f/d latch.

The MIPS d-box Decode

The d-box has multiple tasks.

- It decodes the instruction, noting the operands (register and immediate) and the destination register (if any).
- It also localizes any register operands in the instruction from the register file.
- The register file can be viewed as an array of registers. No instruction has room to store register values; the register fields in an instruction contain register designators (IDs).
- Register names travel up to the register file, and register values go down to the d-box to be stored in the d/x latch.

The MIPS d-box Decode

Handles conditional branches: 'beq' or 'bne'.

- Suppose the branch is 'bne r1,r2,loop'. The d-box tests inequality of the two registers it has previously localized.
- If they are not equal, the branch is taken, and the d-box adds the offset to the base register PC, thus causing the branch to take place. With 16-bit immediates, we can jump 2^15 instructions up or down.
- Recall: The branch-target address is computed by shifting a 16-bit signed number (multiply X 4) and adding it to PC.

The MIPS x-box Execute

The x-box is actually a case statement:

- In a **memory reference**, it adds the base register and the offset to compute the memory address.
- In a **register-register** instruction, it performs the operation specified by the opcode.
- In a **register-immediate** instruction, it performs the operation specified by the opcode, using the literal as an operand value.

The MIPS x-box Execute

The x-box is really a collection of various functional units. E.g. an integer adder, an integer multiplier, a floating-point adder, a floating-point multiplier, a shifter, a logic-operation unit, etc.

Consider the 3 instructions: 'l.d f6,-24(r2)', 'add r1,r2,r3', and 'addi r1,r1,#8'.

As far as the x-box is concerned, these are all ***one case***. The x-box's job here is simply to perform an integer addition; where these integers may have come from is irrelevant.

In contrast, 'mul.d f0,f2,f4' is a different case, since now the x-box needs to perform a floating-point multiplication

The MIPS m-box Memory

The m-box is another case statement:

- If the instruction is a load, the m-box reads from memory and latches the result in the m/w latch.
- If the instruction is a store, the m-box writes to memory taking the value to be stored from some pipeline latch.
- Otherwise, the m-box does nothing.
- Of course, data and control must sometimes flow from the x-box to the w-box even if the m-box itself does nothing. This is called "bypassing".

The MIPS w-box Write-Back

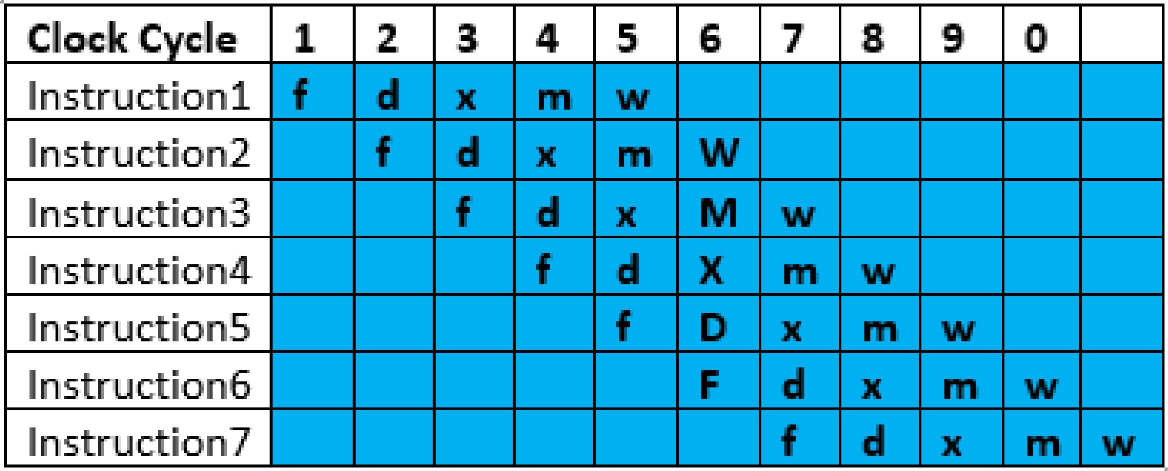
- The w-box does the same thing if the instruction is a load or an ALU instruction, which produces a result.
- Namely, it takes the loaded value, or the ALU result, which must both be in some pipeline register, and writes it in the destination register in the register file.
- **Note**: Not all instructions use all five boxes. A conditional branch uses only the f-box and the d-box. A store uses f, d, x, and m. An ALU instruction uses f, d, x, and w. Only a load uses f, d, x, m, and w.

The MIPS Pipeline Example

Assumptions:

- Task T is the execution of some machine instruction
- Break 'T' into a **sequence** of non-overlapping subtasks: 'T = t1; t2; ...; tn'
- A specialized workstation is provided for each subtask (these are the 5 boxes described earlier designated f, d, x, m, and w).
- The processor clock rings a bell at the start of each processor cycle.
- When the bell rings, each workstation passes the partially executed instruction to the workstation on its right.

The MIPS Pipeline Example



Clock Cycle	1	2	3	4	5	6	7	8	9	0	
Instruction1	f	d	x	m	w						
Instruction2		f	d	x	m	W					
Instruction3			f	d	x	M	w				
Instruction4				f	d	X	m	w			
Instruction5					f	D	x	m	w		
Instruction6						F	d	x	m	w	
Instruction7							f	d	x	m	w

Clock cycles 5, 6, 7 have five boxes working on five different instructions.

Note that an instruction pipeline can have any number of stages. We use 5.