

Computer Architecture
(CSC-3501)
Lecture 18
(01 April 2008)

Seung-Jong Park (Jay)
<http://www.csc.lsu.edu/~sjpark>

1

CSC3501 - S.J. Park

Announcement

2

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- Some CPUs divide the fetch-decode-execute cycle into smaller steps.
- These smaller steps can often be executed in parallel to increase throughput.
- Such parallel execution is called *instruction-level pipelining*.
- This term is sometimes abbreviated *ILP* in the literature.

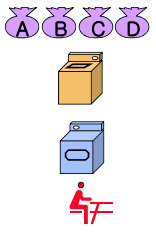
The next slide shows an example of instruction-level pipelining.

3

CSC3501 - S.J. Park

Pipelining: Its Natural!

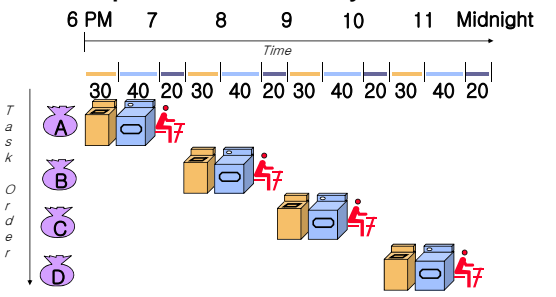
- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



4

CSC3501 - S.J. Park

Sequential Laundry

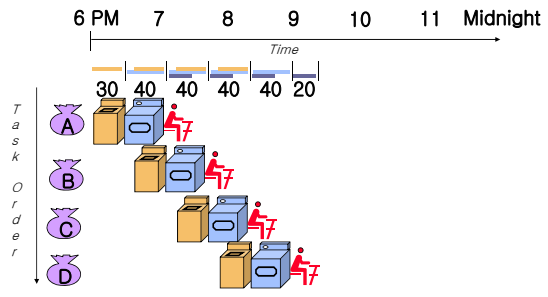


- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

5

CSC3501 - S.J. Park

Pipelined Laundry Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

6

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- Suppose a fetch-decode-execute cycle were broken into the following smaller steps:
 - Fetch instruction.
 - Decode opcode.
 - Calculate effective address of operands.
 - Fetch operands.
 - Execute instruction.
 - Store result.
- Suppose we have a six-stage pipeline. S1 fetches the instruction, S2 decodes it, S3 determines the address of the operands, S4 fetches them, S5 executes the instruction, and S6 stores the result.

7

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- For every clock cycle, one small step is carried out, and the stages are overlapped.

- S1. Fetch instruction.
- S2. Decode opcode.
- S3. Calculate effective address of operands.
- S4. Fetch operands.
- S5. Execute.
- S6. Store result.

8

CSC3501 - S.J. Park

Single cycle vs. Multiple cycle & Pipelining

9

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- The theoretical speedup offered by a pipeline can be determined as follows:
 - Let t_p be the time per stage. Each instruction represents a task, T , in the pipeline.
 - The first task (instruction) requires $k \times t_p$ time to complete in a k -stage pipeline. The remaining $(n - 1)$ tasks emerge from the pipeline one per cycle. So the total time to complete the remaining tasks is $(n - 1)t_p$.
 - Thus, to complete n tasks using a k -stage pipeline requires:

$$(k \times t_p) + (n - 1)t_p = (k + n - 1)t_p.$$

10

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- If we take the time required to complete n tasks without a pipeline and divide it by the time it takes to complete n tasks using a pipeline, we find:

$$\text{Speedup } S = \frac{nt_n}{(k + n - 1)t_p}$$
- If we take the limit as n approaches infinity, $(k + n - 1)$ approaches n , which results in a theoretical speedup of:

$$\text{Speedup } S = \frac{kt_p}{t_p} = k$$

11

CSC3501 - S.J. Park

5.5 Instruction-Level Pipelining

- Our neat equations take a number of things for granted.
 - First, we have to assume that the architecture supports fetching instructions and data in parallel.
 - Second, we assume that the pipeline can be kept filled at all times. This is not always the case. Pipeline *hazards* arise that cause pipeline conflicts and stalls.

12

CSC3501 – S.J. Park

5.5 Instruction-Level Pipelining

- An instruction pipeline may stall, or be flushed for any of the following reasons:
 - Resource conflicts.
 - Data dependencies.
 - Conditional branching.
- Measures can be taken at the software level as well as at the hardware level to reduce the effects of these hazards, but they cannot be totally eliminated.

13

CSC3501 – S.J. Park

5.6 Real-World Examples of ISAs

- We return briefly to the Intel and MIPS architectures from the last chapter, using some of the ideas introduced in this chapter.
- Intel introduced pipelining to their processor line with its Pentium chip.
- The first Pentium had two five-stage pipelines. Each subsequent Pentium processor had a longer pipeline than its predecessor with the Pentium IV having a 24-stage pipeline.
- The Itanium (IA-64) has only a 10-stage pipeline.

14

CSC3501 – S.J. Park

5.6 Real-World Examples of ISAs

- Intel processors support a wide array of addressing modes.
- The original 8086 provided 17 ways to address memory, most of them variants on the methods presented in this chapter.
- Owing to their need for backward compatibility, the Pentium chips also support these 17 addressing modes.
- The Itanium, having a RISC core, supports only one: register indirect addressing with optional post increment.

15

CSC3501 – S.J. Park

5.6 Real-World Examples of ISAs

- MIPS was an acronym for *Microprocessor Without Interlocked Pipeline Stages*.
- The architecture is little endian and word-addressable with three-address, fixed-length instructions.
- Like Intel, the pipeline size of the MIPS processors has grown: The R2000 and R3000 have five-stage pipelines.; the R4000 and R4400 have 8-stage pipelines.

16

CSC3501 – S.J. Park

5.6 Real-World Examples of ISAs

- The R10000 has three pipelines: A five-stage pipeline for integer instructions, a seven-stage pipeline for floating-point instructions, and a six-stage pipeline for **LOAD/STORE** instructions.
- In all MIPS ISAs, only the **LOAD** and **STORE** instructions can access memory.
- The ISA uses only base addressing mode.
- The assembler accommodates programmers who need to use immediate, register, direct, indirect register, base, or indexed addressing modes.

17

CSC3501 – S.J. Park

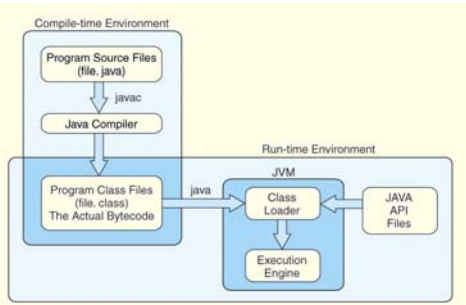
5.6 Real-World Examples of ISAs

- The Java programming language is an interpreted language that runs in a software machine called the *Java Virtual Machine (JVM)*.
- A JVM is written in a native language for a wide array of processors, including MIPS and Intel.
- Like a real machine, the JVM has an ISA all of its own, called *bytecode*. This ISA was designed to be compatible with the architecture of any machine on which the JVM is running.

The next slide shows how the pieces fit together.

18

5.6 Real-World Examples of ISAs



19

5.6 Real-World Examples of ISAs

- Java bytecode is a stack-based language.
- Most instructions are zero address instructions.
- The JVM has four registers that provide access to five regions of main memory.
- All references to memory are offsets from these registers. Java uses no pointers or absolute memory references.
- Java was designed for platform interoperability, not performance!

20

Chapter 5 Conclusion

- ISAs are distinguished according to their bits per instruction, number of operands per instruction, operand location and types and sizes of operands.
- Endianness as another major architectural consideration.
- CPU can store store data based on
 1. A stack architecture
 2. An accumulator architecture
 3. A general purpose register architecture.

21

Chapter 5 Conclusion

- Instructions can be fixed length or variable length.
- To enrich the instruction set for a fixed length instruction set, expanding opcodes can be used.
- The addressing mode of an ISA is also another important factor. We looked at:
 - Immediate – Direct
 - Register – Register Indirect
 - Indirect – Indexed
 - Based – Stack

22

Chapter 5 Conclusion

- A k -stage pipeline can theoretically produce execution speedup of k as compared to a non-pipelined machine.
- Pipeline hazards such as resource conflicts and conditional branching prevents this speedup from being achieved in practice.
- The Intel, MIPS, and JVM architectures provide good examples of the concepts presented in this chapter.

23