Computer Architecture
(CSC-3501)
Lecture 25
(24 April 2008)

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

Announcement

Chapter 9 Objectives

- Learn the properties that often distinguish RISC from CISC architectures.
- Understand how multiprocessor architectures are classified.
- Appreciate the factors that create complexity in multiprocessor systems.
- Become familiar with the ways in which some architectures transcend the traditional von Neumann paradigm.

9.1 Introduction

- We have so far studied only the simplest models of computer systems; classical single-processor von Neumann systems.
- This chapter presents a number of different approaches to computer organization and architecture.
- Some of these approaches are in place in today’s commercial systems. Others may form the basis for the computers of tomorrow.

9.2 RISC Machines

- The underlying philosophy of RISC machines is that a system is better able to manage program execution when the program consists of only a few different instructions that are the same length and require the same number of clock cycles to decode and execute.
- RISC systems access memory only with explicit load and store instructions.
- In CISC systems, many different kinds of instructions access memory, making instruction length variable and fetch-decode-execute time unpredictable.

9.2 RISC Machines

- The difference between CISC and RISC becomes evident through the basic computer performance equation:

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- RISC systems shorten execution time by reducing the clock cycles per instruction.
- CISC systems improve performance by reducing the number of instructions per program.

9.2 RISC Machines

- The simple instruction set of RISC machines enables control units to be hardwired for maximum speed.
- The more complex-- and variable-- instruction set of CISC machines requires microcode-based control units that interpret instructions as they are fetched from memory. This translation takes time.
- With fixed-length instructions, RISC lends itself to pipelining and speculative execution.

9.2 RISC Machines

- Consider the the program fragments:

CISC

```
mov ax, 10
mov bx, 5
mul bx, ax
```

RISC

```
       mov ax, 0
       mov bx, 10
       mov cx, 5
Begin  add ax, bx
       loop Begin
```

- The total clock cycles for the CISC version might be:
  `(2 movs × 1 cycle) + (1 mul × 30 cycles) = 32 cycles`
- While the clock cycles for the RISC version is:
  `(3 movs × 1 cycle) + (5 adds × 1 cycle) + (5 loops × 1 cycle) = 13 cycles`
- With RISC clock cycle being shorter, RISC gives us much faster execution speeds.

9.2 RISC Machines

- Because of their load-store ISAs, RISC architectures require a large number of CPU registers.
- These register provide fast access to data during sequential program execution.
- They can also be employed to reduce the overhead typically caused by passing parameters to subprograms.
- Instead of pulling parameters off of a stack, the subprogram is directed to use a subset of registers.

9.2 RISC Machines

- It is becoming increasingly difficult to distinguish RISC architectures from CISC architectures.
- Some RISC systems provide more extravagant instruction sets than some CISC systems.
- Some systems combine both approaches.
- The following two slides summarize the characteristics that traditionally typify the differences between these two architectures.

9.2 RISC Machines

- **RISC**
  - Multiple register sets.
  - Three operands per instruction.
  - Parameter passing through register windows.
  - Single-cycle instructions.
  - Hardwired control.
  - Highly pipelined.
- **CISC**
  - Single register set.
  - One or two register operands per instruction.
  - Parameter passing through memory.
  - Multiple cycle instructions.
  - Microprogrammed control.
  - Less pipelined.

Continued….

9.2 RISC Machines

- **RISC**
  - Simple instructions, few in number.
  - Fixed length instructions.
  - Complexity in compiler.
  - Only `LOAD/STORE` instructions access memory.
  - Few addressing modes.
- **CISC**
  - Many complex instructions.
  - Variable length instructions.
  - Complexity in microcode.
  - Many instructions can access memory.
  - Many addressing modes.

9.3 Flynn's Taxonomy

- Many attempts have been made to come up with a way to categorize computer architectures.
- *Flynn's Taxonomy* has been the most enduring of these, despite having some limitations.
- Flynn's Taxonomy takes into consideration the number of processors and the number of data paths incorporated into an architecture.
- A machine can have one or many processors that operate on one or many data streams.

9.3 Flynn's Taxonomy

- The four combinations of multiple processors and multiple data paths are described by Flynn as:

Instruction streams	Data streams	Name	Examples
1	1	SISD	Classical Von Neumann machine
1	Multiple	SIMD	Vector supercomputer, array processor
Multiple	1	MISD	Arguably none
Multiple	Multiple	MIMD	Multiprocessor, multicomputer

9.3 Flynn's Taxonomy

- Flynn's Taxonomy falls short in a number of ways:
- First, there appears to be no need for MISD machines.
- Second, parallelism is not homogeneous. This assumption ignores the contribution of specialized processors.
- Third, it provides no straightforward way to distinguish architectures of the MIMD category.
  - One idea is to divide these systems into those that share memory, and those that don't, as well as whether the interconnections are bus-based or switch-based.

9.3 Flynn's Taxonomy

- Symmetric multiprocessors (SMP) and massively parallel processors (MPP) are MIMD architectures that differ in how they use memory.
- SMP systems share the same memory and MPP do not.
- An easy way to distinguish SMP from MPP is:

MPP $\Rightarrow$ many processors + distributed memory + communication via network

SMP $\Rightarrow$ fewer processors + shared memory + communication via memory

9.3 Flynn's Taxonomyh

- Other examples of MIMD architectures are found in distributed computing, where processing takes place collaboratively among networked computers.
  - A **network of workstations** (NOW) uses otherwise idle systems to solve a problem.
  - A **collection of workstations** (COW) is a NOW where one workstation coordinates the actions of the others.
  - A **dedicated cluster parallel computer** (DCPC) is a group of workstations brought together to solve a specific problem.
  - A **pile of PCs** (POPC) is a cluster of (usually) heterogeneous systems that form a dedicated parallel system.

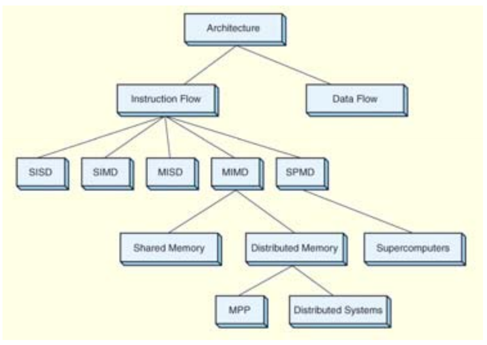
9.3 Flynn's Taxonomy

- Flynn's Taxonomy has been expanded to include SPMD (single program, multiple data) architectures.
- Each SPMD processor has its own data set and program memory. Different nodes can execute different instructions within the same program using instructions similar to:

  If myNodeNum = 1 do this, else do that
- Yet another idea missing from Flynn's is whether the architecture is instruction driven or data driven.

The next slide provides a revised taxono my.

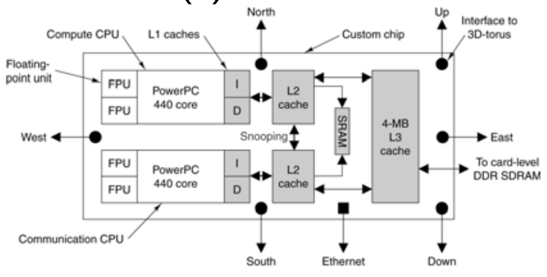
9.3 Flynn’s Taxonomy



Current Supercomputer 500

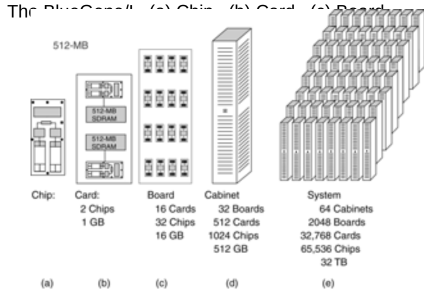
TOP5
PERFORMANCE DEVELOPMENT
PROJECTED PERFORMANCE DEVELOPMENT
ARCHITECTURES
CHIP TECHNOLOGY

BlueGene (1)

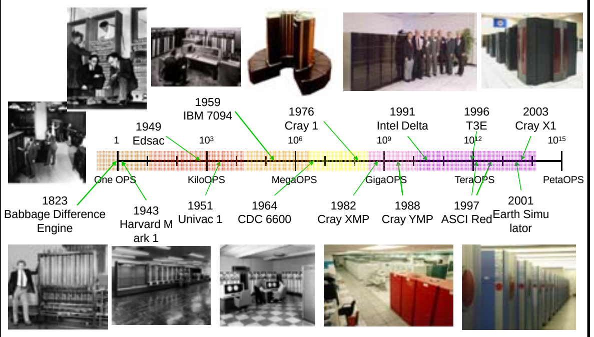


BlueGene (2)

The BlueGene/L (a) Chip (b) Card (c) Board


512-MB
512-MB SDRAM
512-MB SDRAM
Chip:
Card: 2 Chips 1 GB
Board 16 Cards 32 Chips 16 GB
Cabinet 32 Boards 512 Cards 1024 Chips 512 GB
System 64 Cabinets 2048 Boards 32,768 Cards 65,536 Chips 32 TB
(a)
(b)
(c)
(d)
(e)

A Growth of High Performance Computing



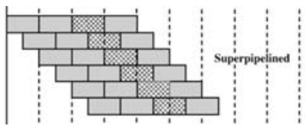
By Thomas Sterling – Caltech & JPL

9.4 Parallel and Multiprocessor Architectures

- Parallel processing is capable of economically increasing system throughput while providing better fault tolerance.
- The limiting factor is that no matter how well an algorithm is parallelized, there is always some portion that must be done sequentially.
  - Additional processors sit idle while the sequential work is performed.
- Thus, it is important to keep in mind that an *n* -fold increase in processing power does not necessarily result in an *n* -fold increase in throughput.

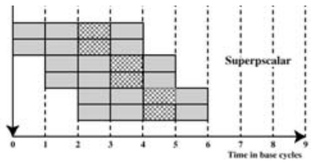
9.4 Superpipelining

- Recall that pipelining divides the fetch-decode-execute cycle into stages that each carry out a small part of the process on a set of instructions.
- Ideally, an instruction exits the pipeline during each tick of the clock.
- *Superpipelining* occurs when a pipeline has stages that require less than half a clock cycle to complete.
  - The pipeline is equipped with a separate clock running at a frequency that is at least double that of the main system clock.
- Superpipelining is only one aspect of superscalar design.

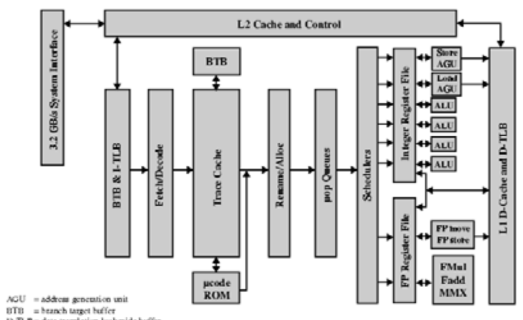


9.4 Superscalar (Dynamic multiple-issue processors)

- Superscalar architectures include multiple execution units such as specialized integer and floating-point adders and multipliers.
- A critical component of this architecture is the *instruction fetch unit*, which can simultaneously retrieve several instructions from memory.
- A *decoding unit* determines which of these instructions can be executed in parallel and combines them accordingly.
- This architecture also requires compilers that make optimum use of the hardware.



Pentium 4 Block Diagram

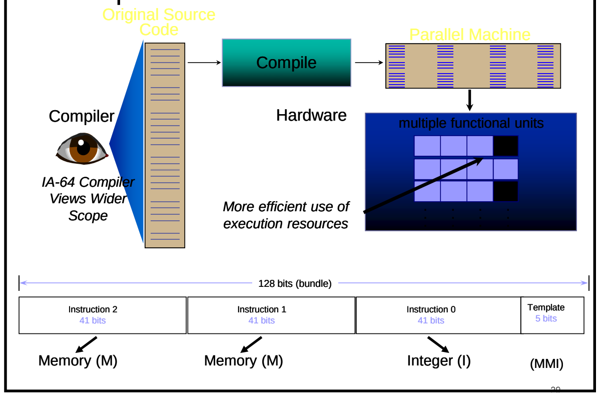


AGU = address generation unit
BTB = branch target buffer
D-TLB = data translation lookaside buffer
I-TLB = instruction translation lookaside buffer

9.4 VLIW (Static multiple-issue processors)

- Very long instruction word (VLIW) architectures differ from superscalar architectures because the VLIW compiler, instead of a hardware decoding unit, packs independent instructions into one long instruction that is sent down the pipeline to the execution units.
  - Compiler takes greater responsibility for exploiting parallelism
- One could argue that this is the best approach because the compiler can better identify instruction dependencies.
- However, compilers tend to be conservative and cannot have a view of the run time code.
  - E.g., Intel Itanium and Itanium 2 for the IA-64 ISA – EPIC (Explicit Parallel Instruction Computer)

Example of VLIW - IA-64



CISC vs RISC vs SS vs VLIW

	CISC	RISC	Superscalar	VLIW
Instr size	variable size	fixed size	fixed size	fixed size (but large)
Instr format	variable format	fixed format	fixed format	fixed format
Registers	few, some special	many GP	GP and rename (RUU)	many, many GP
Memory reference	embedded in many instr's	load/store	load/store	load/store
Key Issues	decode complexity	data forwarding, hazards	hardware dependency resolution	(compiler) code scheduling
Instruction flow		IF ID EX M WB IF ID EX M WB IF ID EX M WB	IF ID EX M WB IF ID EX M WB IF ID EX M WB IF ID EX M WB	IF ID EX M WB EX M WB IF ID EX M WB EX M WB