Chapter 10

Reduced Instruction Set Computers (RISCs)

10.1 RISC/CISC Evolution Cycle

- The term RISCs stands for **R**educed **I**nstruction **S**et **C**omputers.
- It was originally introduced as a notion to mean architectures that can execute as fast as one instruction per clock cycle.
- This paradigm promotes simplicity in computer architectures design.
- This paradigm shift relates to what is known as the *Semantic Gap*, a measure of the difference between the operations provided in the high level languages (HLLs) and those provided in computer architectures.

10.1 RISC/CISC Evolution Cycle

- It is recognized that the wider the semantic gap, the larger the number of undesirable consequences.
- These include
  - (a) execution inefficiency,
  - (b) excessive machine program size, and
  - (c) increased compiler complexity.
- Because of these expected consequences, the conventional response of computer architects has been to add layers of complexity to newer architectures.
- These include increasing the number and complexity of instructions together with increasing the number of addressing modes.

10.1 RISC/CISC Evolution Cycle

- The architectures resulting from the adoption of this “add more complexity” are now known as **C**omplex **I**nstruction **S**et **C**omputers (CISCs).
- However, it soon became apparent that a complex instruction set has a number of disadvantages.
- These include a complex instruction decoding scheme, an increased size of the control unit, and increased logic delays.

10.2 RISCs Design Principles

Operations	Estimated Percentage
Assignment Statements	35
Loops	5
Procedure Calls	15
Conditional Branches	40
Unconditional Branches	3
Others	2

10.2 RISCs Design Principles

- Simple movement of data (represented by assignment statements), rather than complex operations, are substantial and should be optimized.
- Conditional branches are predominant and therefore careful attention should be paid to the sequencing of instructions. This is particularly true when it is known that pipelining is indispensable to use.
- Procedure calls/return are the most time consuming operations and therefore a mechanism should be devised to make the communication of parameters among the calling and the called procedures cause the least number of instructions to execute.
- A prime candidate for optimization is the mechanism for storing and accessing local scalar variables.

10.2 RISCs Design Principles

- 2 principles:
  - Keeping the most frequently accessed operands in CPU registers.
  - Minimizing the register-to-memory operations.
- These principles can be achieved using the following mechanisms:
  - Use a large number of registers to optimize operand referencing and reduce the processor memory traffic.
  - Optimize the design of instruction pipelines such that minimum compiler code generation can be achieved.
  - Use a simplified instruction set and leave out those complex and unnecessary instructions.

10.2 RISCs Design Principles

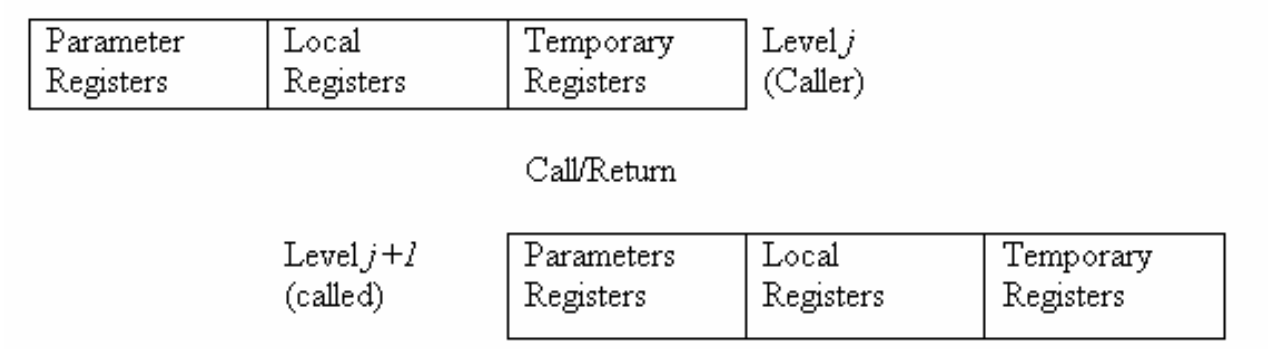
- The following two approaches were identified to implement the above three mechanisms.
  - Software Approach
    - Use the compiler to maximize register usage by allocating registers to those variables that are used the most in a given time period.
  - Hardware Approach
    - Use ample of CPU registers so that more variables can be held in registers for larger periods of time. The hardware approach necessitates the use of a new register organization, called *overlapped register window*.

10.3 Overlapped Register Windows

- The main idea behind the use of register windows is to minimize memory accesses.
- A procedure call will automatically switch the CPU to use a different fixed-size window of registers.
- In order to minimize the actual movement of parameters among the calling and the called procedures, each set of registers is divided into three subsets:
  - Parameter Registers,
  - Local Registers, and
  - Temporary Registers.

10.3 Overlapped Register Windows

- When a procedure call is made, a new overlapping window will be created such that the Temporary Registers of the caller are physically the same as the Parameter Registers of the called procedure.
- This overlap allows parameters to be passed among procedure without actual movement of data.



10.4 RISCs Versus CISCs

- RISC versus CISC Performance:

Application	MIPS CPI (RISC)	VAX CPI (CISC)	CPI Ratio	Instruction Ratio
Spice2G6	1.80	8.02	4.44	2.48
Matrix300	3.06	13.81	4.51	2.37
Nasa 7	3.01	14.95	4.97	2.10
Espresso	1.06	5.40	5.09	1.70

- RISC versus CISC Characteristics:

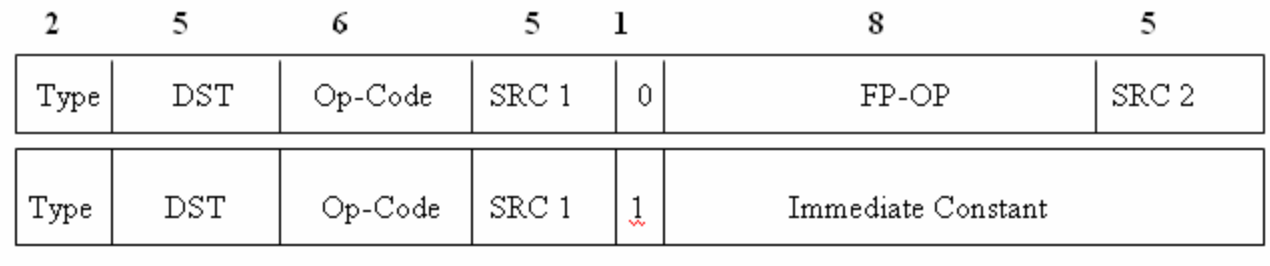
Characteristic	VAX-11 (CISC)	Berkeley RISC-1 (RISC)
Number of Instructions	303	31
Instruction size (bits)	16-456	32
Addressing Modes	22	3
No. General purpose registers	16	138

10.4 RISCs Versus CISCs

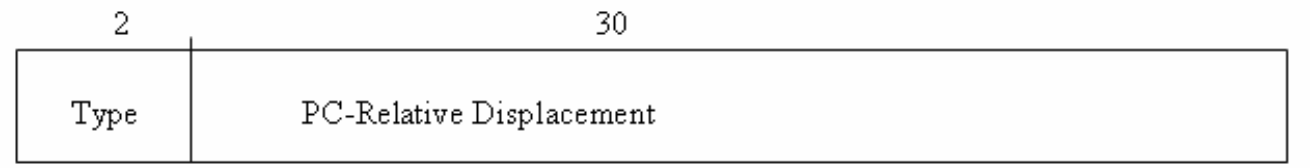
- The following set of common characteristics among RISC machines is observed:
  - Fixed-length instructions.
  - Limited number of instructions (128 or less).
  - Limited set of simple addressing modes (minimum of two: Indexed and PC-relative).
  - All operations are performed on registers; no memory operations.
  - Only two memory operations: Load and Store.
  - Pipelined instruction execution.
  - Large number of general purpose registers or the use of advanced compiler technology to optimize register usage
  - One instruction per clock cycle.
  - Hardwired Control unit design rather than microprogramming.

10.5 Pioneer (University) RISC Machines

- The Berkeley RISC
  - Three operand instructions formats used in RISC:

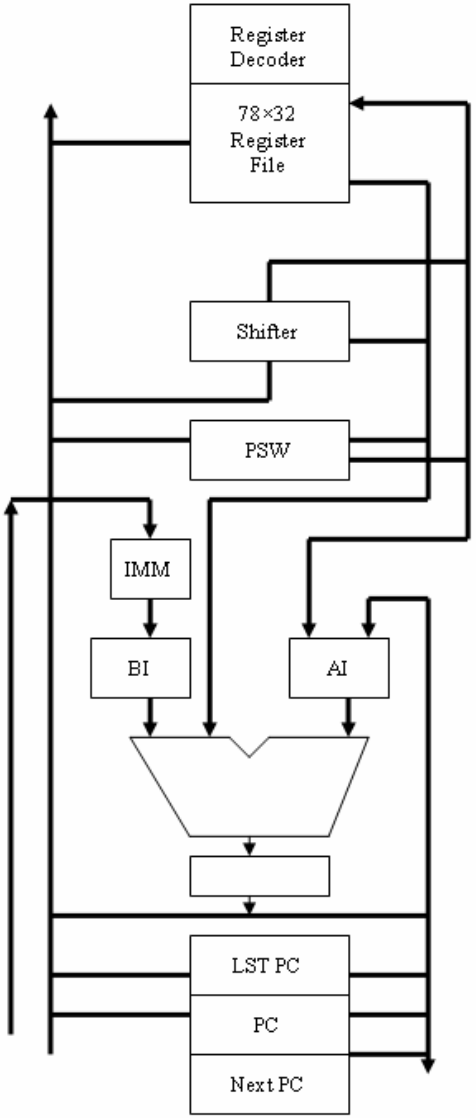


  - Procedure call instruction in RISC:



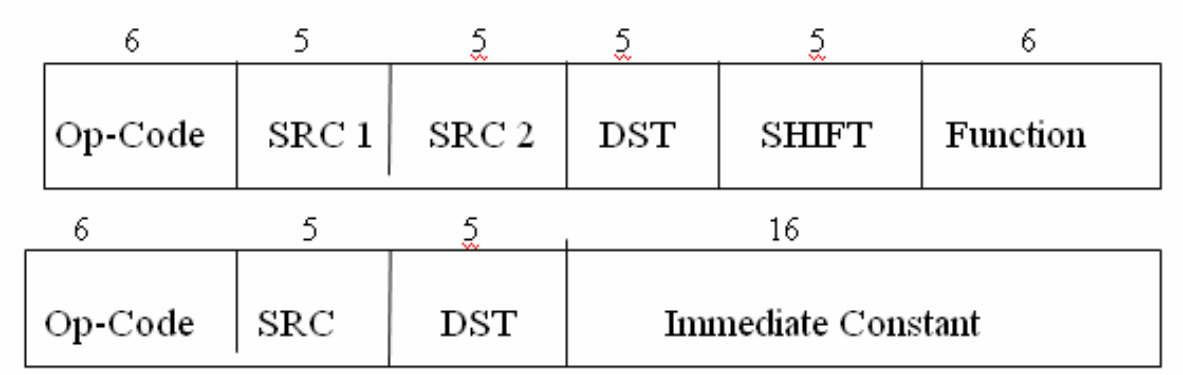
10.5 Pioneer (University) RISC Machines

- The Berkeley RISC
  - RISC 4-bus Organization:

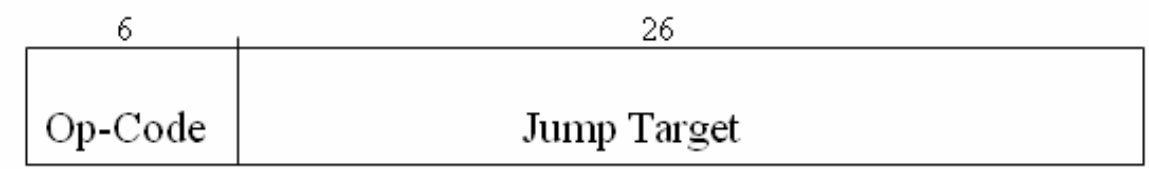


10.5 Pioneer (University) RISC Machines

- Stanford MIPS (Microprocessor without Interlock Pipe Stages)
  - Three-operand instructions used in MIPS:

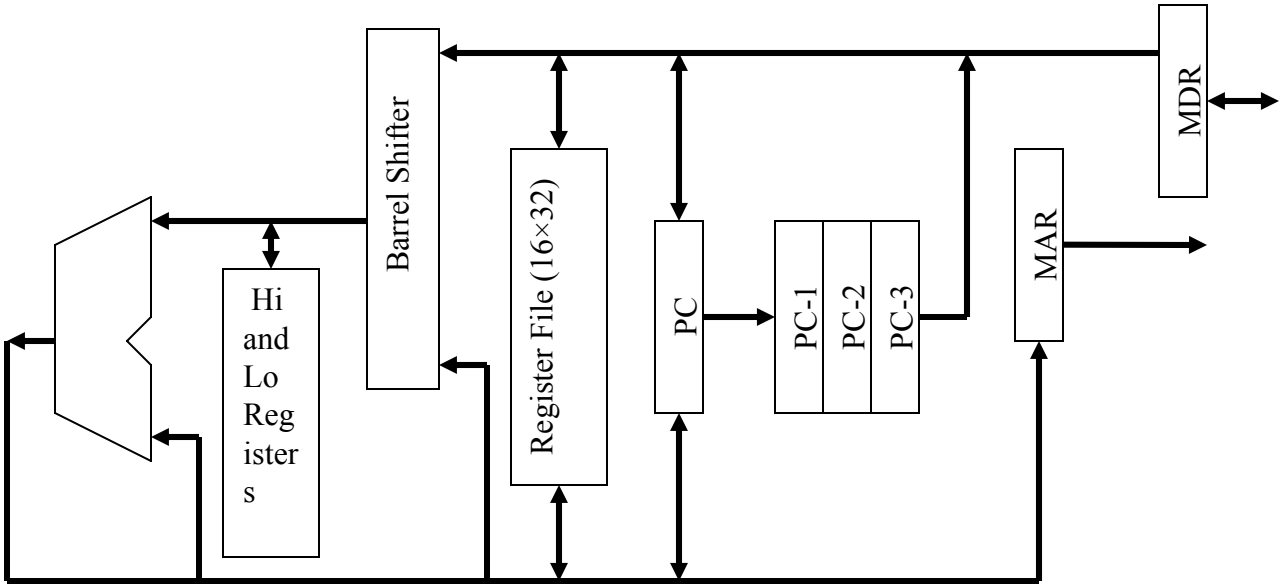


  - Jump instruction format used in MIPS:



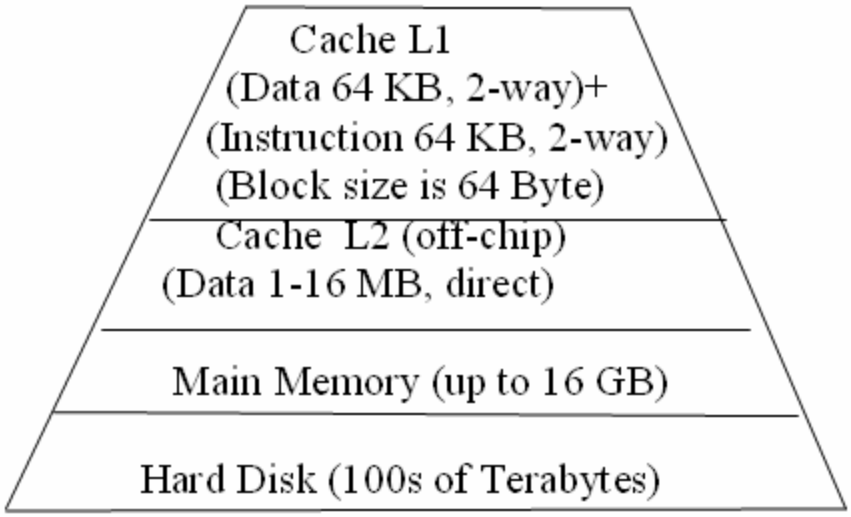
10.5 Pioneer (University) RISC Machines

- Stanford MIPS (Microprocessor without Interlock Pipe Stages)
  - MIPS Organization:

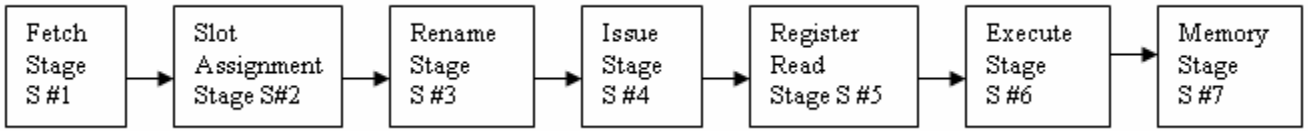


10.6 Example of Advanced RISC Machines

- Compaq (formerly DEC) Alpha 21264
  - Memory Hierarchy:

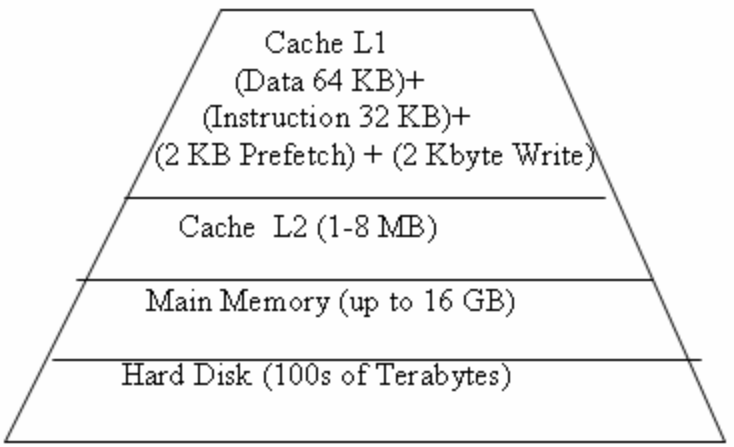


  - The Alpha 21264 Pipeline:

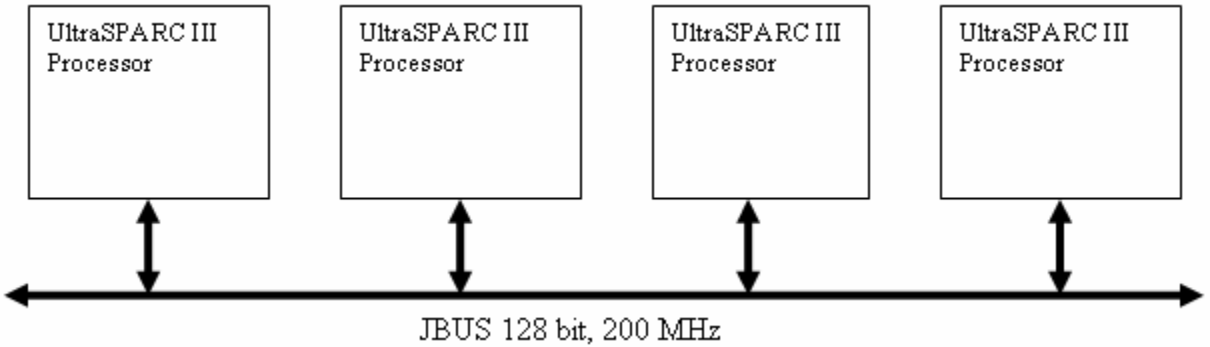


10.6 Example of Advanced RISC Machines

- SUN UltraSPARC III
  - UltraSPARC III memory hierarchy:



  - A four-way UltraSPARC III Multiprocessor configuration:



10.7 Summary

- A RISC architecture saves the extra chip area used by CISC architectures for decoding and executing complex instructions.
- The saved chip area is then used to provide on-chip instruction cache that can be used to reduce instruction traffic between the processor and the memory.
- Common characteristics shared by most RISC designs are:
  - Limited and simple instruction set, Large number of general purpose registers and/or the use of compiler technology to optimize register usage, and optimization of the instruction pipeline.
- An essential RISC philosophy is to keep the most frequently accessed operands in registers and minimize register-memory operations.
- This can be achieved using one of two approaches:
  - Software Approach
  - Hardware Approach

10.7 Summary

- It is worthwhile mentioning that the classification of processors as entirely pure RISC or entirely pure CISC is becoming more and more inappropriate and may be irrelevant.
- What actually counts is how much performance gain can be achieved by including an element of a given design style.
- Most of modern processors use a calculated combination of elements of both design styles.
- The decisive factor in which element(s) of each design style to include is made based on a tradeoff between the required improvement in performance and the expected added cost.
- A number of processors are classified as RISC while employing a number of CISC features, such as integer/floating-point division instructions.
- Similarly, there exist processors that are classified as CISC while employing a number of RISC features, such as pipelining.