DAR ES SALAAM INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER STUDIES

COU07302 MICROPROCESSOR AND COMPUTER ARCHITECTURE

Lecture 1 - Introduction to RISC

by

E. Kondela

Introduction

Since the development of the stored program computer around 1950, there are few innovations in the area of computer organization and architecture. Some of the major developments are:

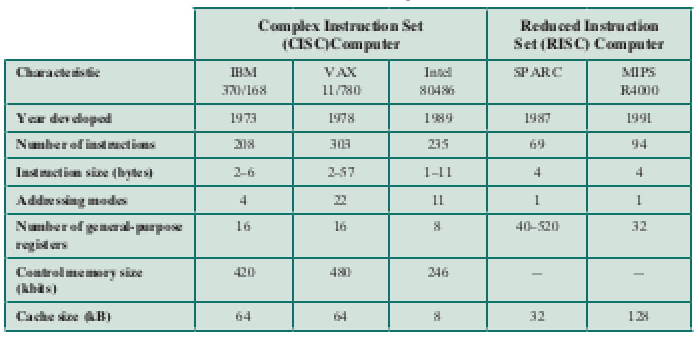
- **The Family Concept**: Introduced by IBM with its system/360 in 1964 followed by DEC, with its PDP-8. The family concept decouples the architecture of a machine from its implementation. A set of computers are offered, with different price/performance characteristics, that present the same architecture to the user.
- **Microprogrammed Control Unit**: Suggested by Wilkes in 1951, and introduced by IBM on the S/360 line in 1964. Microprogramming eases the task of designing and implementing the control unit and provide support for the family concept.
- **Cache Memory**: First introduced commercially on IBM S/360 Model 85 in 1968. The insertion of this element into the memory hierarchy dramatically improves performance.
- **Pipelining**: A means of introducing parallelism into the essentially sequential nature of a machine instruction program. Examples are instruction pipelining and vector processing.
- **Multiple Processor**: This category covers a number of different organizations and objectives.

When it appeared, RISC architecture was a dramatic departure from the historical trend in processor architecture. An analysis of the RISC architecture brings into focus many of the important issues in computer organization and architecture.

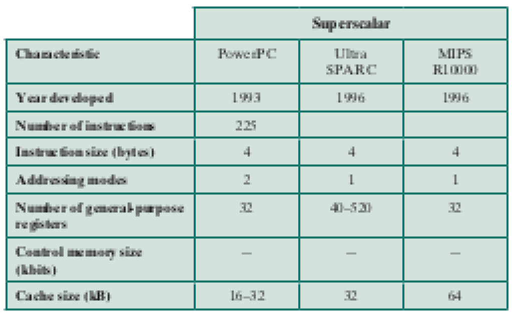
Although RISC architectures have been defined and designed in a variety of ways by different groups, the key elements shared by most designs are these:

- A large number of general-purpose registers, and/or the use of compiler technology to optimize register usage.
- A limited and simple instruction set.
- An emphasis on optimizing the instruction pipeline.

Table 1 compares several RISC and non-RISC systems. We begin this chapter with a brief survey of some results on instruction sets, and then examine each of the three topics just listed. This is followed by a description of two of the best-documented RISC designs



	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer	
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
Number of instructions	208	303	235	69	94
Instruction size (bytes)	2–6	2–57	1–11	4	4
Addressing modes	4	22	11	1	1
Number of general-purpose registers	16	16	8	40–520	32
Control memory size (kbits)	420	480	246	—	—
Cache size (kB)	64	64	8	32	128



	Superscalar		
Characteristic	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1993	1996	1996
Number of instructions	225		
Instruction size (bytes)	4	4	4
Addressing modes	2	1	1
Number of general-purpose registers	32	40–520	32
Control memory size (kbits)	—	—	—
Cache size (kB)	16–32	32	64

Instruction Execution Characteristics

One of the most visual forms of evolution associated with computers is that of programming languages. Even more powerful and complex high level programming languages has been developed by the researcher and industry people.

The development of powerful high level programming languages give rise to another problem known as the semantic gap, the difference between the operations provided in HLLs and those provided in computer architecture.

The computer designers intend to reduce this gap and include large instruction set, more addressing mode and various HLL statements implemented in hardware. As a result the instruction set becomes complex. Such complex instruction sets are intended to-

- Ease the task of the compiler writer.
- Improve execution efficiency, because complex sequences of operations can be implemented in microcode.
- Provide support for even more complex and sophisticated HLLs.

To reduce the gap between HLL and the instruction set of computer architecture, the system becomes more and more complex and the resulted system is termed as Complex Instruction Set Computer (CISC).

A number of studies have been done over the years to determine the characteristics and patterns of execution of machine instructions generated from HLL programs. The instruction execution characteristics involves the following aspects of computation:

- **Operation Performed**: These determine the functions to be performed by the processor and its interaction with memory.
- **Operand Used**: The types of operands and the frequency of their use determine the memory organization for storing them and the addressing modes for accessing them.
- **Execution sequencing**: This determines the control and pipeline organization.

These results are instructive to the machine instruction set designers, indicating which type of statements occur most often and therefore should be supported in an “optimal” fashion.

From these studies one can observe that though a complex and sophisticated instruction set is available in a machine architecture, common programmer may not use those instructions frequently.

Operands :

Researches also studied the dynamic frequency of occurrence of classes of variables. The results showed that majority of references are single scalar variables. In addition references to arrays/structures required a previous reference to their index or pointer, which again is usually a local scalar. Thus there is a predominance of references to scalars, and these are highly localized.

It is also observed that operation on local variables is performed frequently and it requires a fast accessing of these operands. So, it suggests that a prime candidate for optimization is the mechanism for storing and accessing local scalar variables.

Procedure Call :

The procedure calls and returns are an important aspects of HLL programs. Due to the concept of modular and functional programming, the call/return statements are becoming a predominate factor in HLL program.

It is known fact that call/return is a most time consuming and expensive statements. Because during call we have to restore the current state of the program which includes the contents of local variables that are present in general purpose registers. During return, we have to restore the original state of the program from where we start the procedure call.

Thus, it will be profitable to consider ways of implementing these operations efficiently. Two aspects are significant, the number of parameters and variables that a procedure deals with, and the depth of nesting.

Implications :

A number of groups have looked at these results and have concluded that the attempt to make the instruction set architecture close to HLL is not the most effective design strategy. Generalizing from the work of a number of researchers three element emerge in the computer architecture.

- First, use a large number of registers or use a compiler to optimize register usage. This is intended to optimize operand referencing.
- Second, careful attention needs to be paid to the design of instruction pipelines. Because of the high proportion of conditional branch and procedure call instructions, a straight forward instruction pipeline will be inefficient. This manifests itself as a high proportion of instructions that are prefetched but never executed.
- Third, a simplified (reduced) instruction set is indicated. It is observed that there is no point to design a complex instruction set which will lead to a complex architecture. Due to the fact, a most interesting and important processor architecture evolves which is termed as Reduced Instruction Set Computer (RISC) architecture.

Although RISC system have been defined and designed in a variety of ways by different groups, the key element shared by most design are these:

- A large number of general purpose registers, or the use of compiler technology to optimize register usage.
- A limited and simple instruction set.
- An emphasis on optimizing the instruction pipelin

An analysis of the RSIC architecture begins into focus many of the important issues in computer organization and architecture. The table in the next page compares several RISC and non-RISC system. . . . . .

Characteristics of Reduced Instruction Set Architecture :

Although a variety of different approaches to reduce Instruction set architecture have been taken, certain characteristics are common to all of them:

1. One instruction per cycle.
2. Register–to–register operations.
3. Simple addressing modes.
4. Simple instruction formats.

1. One machine instruction per machine cycle :

A machine cycle is defined to be the time it takes to fetch two operands from registers, perform an ALU operation, and store the result in a register.

With simple, one-cycle instructions there is little or no need of microcode, the machine instructions can be hardwired. Hardware implementation of control unit executes faster than the microprogrammed control, because it is not necessary to access a microprogram control store during instruction execution.

2. Register –to– register operations

With register–to–register operation, a simple LOAD and STORE operation is required to access the memory, because most of the operation are register–to-register. Generally we do not have memory–to–memory and mixed register/memory operation.

3. Simple Addressing Modes

Almost all RISC instructions use simple register addressing. For memory access only, we may include some other addressing, such as displacement and PC-relative. Once the data are fetched inside the CPU, all instruction can be performed with simple register addressing.

4. Simple Instruction Format

Generally in most of the RISC machine, only one or few formats are used. Instruction length is fixed and aligned on word boundaries. Field locations, especially the opcode, are fixed.
With fixed fields, opcode decoding and register operand accessing can occur simultaneously. Simplified formats simplify the control unit.

The use of a large register file:

The reason that register storage is indicated is that it is the fastest available storage device, faster than both main memory and cache. The register file will allow the most frequently accessed operands to be kept in registers and to minimize register-memory operations.
Two basic approaches are possible, one based on software and the other on hardware.
The software approach is to rely on the compiler to maximize register usage. The compiler will attempt to allocate registers to those variables that will be used the most in a given time period. This approach requires the use of sophisticated program-analysis algorithms.
The hardware approach is simply to use more registers so that more variables can be held in registers for longer periods of time. Here we will discuss the hardware approach.

For fast execution of instructions, it is desirable of quick access to operands.
There is large proportion of assignment statements in HLL programs, and many of these are of the simple form $A \leftarrow B$. Also there are significant number of operand accesses per HLL Statement.
Also it is observed that most of the accesses are local scalars. To get a fast response, we must have an easy excess to these local scalars, and so the use of register storage is suggested. Since registers are the fastest available storage devices, faster than both main memory and cache, so the uses of registers are preferable. The register file is physically small, and on the same chip as the ALU and Control Unit. A strategy is needed that will allow the most frequently accessed operands to be kept in registers and to minimize register-memory operations.
Two basic approaches are possible, one is based on software and the other on hardware.

- The software approach is to rely on the compiler to maximize register uses. The compiler will attempt to allocate registers to those variables that will be used the most in a given time period.
- The hardware approach is simply to use more registers so that more variables can be held in registers for longer period of time.

Register Window :

The use of a large set of registers should decrease the need to access memory. The design task is to organize the registers in such a way that this goal is realized.
Due to the use of the concept of modular programming, the present day programs are dominated by call/return statements. There are some local variables present in each function or procedure.

1. On every call, local variables must be saved from the registers into memory, so that the registers can be reused by the called program. Furthermore, the parameters must be passed.
2. On return, the variables of the parent program must be restored (loaded back into registers) and results must be passed back to the parent program.
3. There are also some global variables which are used by the module or procedure.

Thus the variables that are used in a program can be categorized as follows :

- Global variables : which is visible to all the procedures.

- Local variables : which is local to a procedure and it can be accessed inside the procedure only.
- Passed parameters : which are passed to a subroutine from the calling program. So, these are visible to both called and calling program.
- Returned variable : variable to transfer the results from called program to the calling program. These are also visible to both called and calling program.

Why CISC

CISC has richer instruction sets, which include a larger number of instructions and more complex instructions. Two principal reasons have motivated this trend: a desire to simplify compilers and a desire to improve performance.

The first of the reasons cited, compiler simplification, seems obvious. The task of the compiler writer is to generate a sequence of machine instructions for each HLL statement. If there are machine instructions that resemble HLL statements, this task is simplified.

This reasoning has been disputed by the RISC researchers. They have found that complex machine instructions are often hard to exploit because the compiler must find those cases that exactly fit the construct. The task of optimizing the generated code to minimize code size, reduce instruction execution count, and enhance pipelining is much more difficult with a complex instruction set.

The other major reason cited is the expectation that a CISC will yield smaller, faster programs. Let us examine both aspects of this assertion: that program will be smaller and that they will execute faster. There are two advantages to smaller programs.

- First, because the program takes up less memory, there is a savings in that resource.
- Second, in a paging environment, smaller programs occupy fewer pages, reducing page faults.

The problem with this line of reasoning is that it is far from certain that a CISC program will be smaller than a corresponding RISC program. Thus it is far from clear that a trend to increasingly complex instruction sets is appropriate. This has led a number of groups to pursue the opposite path.

CISC versus RISC Characteristics

After the initial enthusiasm for RISC machines, there has been a growing realization that (1) RISC designs may benefit from the inclusion of some CISC features and that (2) CISC designs may benefit from the inclusion of some RISC features. The result is that the more recent RISC designs, notably the PowerPC, are no longer “pure” RISC and the more recent CISC designs, notably the Pentium II and later Pentium models, do incorporate some RISC characteristics.

For purposes of this comparison, the following are considered typical of a classic RISC:

1. A single instruction size.
2. That size is typically 4 bytes.
3. A small number of data addressing modes,typically less than five.This parameter is difficult to pin down. In the table, register and literal modes are not counted and different formats with different offset sizes are counted separately.
4. No indirect addressing that requires you to make one memory access to get the address of another operand in memory.
5. No operations that combine load/store with arithmetic (e.g., add from memory, add to memory).
6. No more than one memory-addressed operand per instruction.
7. Does not support arbitrary alignment of data for load/store operations.
8. Maximum number of uses of the memory management unit (MMU) for a data address in an instruction.
9. Number of bits for integer register specifier equal to five or more. This means that at least 32 integer registers can be explicitly referenced at a time.

10. Number of bits for floating-point register specifier equal to four or more.This means that at least 16 floating-point registers can be explicitly referenced at a time.

Items 1 through 3 are an indication of instruction decode complexity. Items 4 through 8 suggest the ease or difficulty of pipelining, especially in the presence of virtual memory requirements. Items 9 and 10 are related to the ability to take good advantage of compilers.

In the table, the first eight processors are clearly RISC architectures, the next five are clearly CISC, and the last two are processors often thought of as RISC that in fact have many CISC characteristics.