Stallings 2012 edition

CS 320
Ch. 15 RISC

NOTE: Skip R4000 discussion near the end of the chapter. We will do the SUN Sparc only.

The author lists 5 major advances in computer architecture since 1950:
The family concept **IBM 360 in 1964**
Microprogrammed Control Unit **Wilkes in 1950 and IBM in 1964**
Cache memory **IBM 360 in 1968**
Pipelining **1956-1961 IBM Stretch**
Multiple processors
RISC. **IBM in 1980**

The key elements of a RISC machine are:
Large number of gp registers
Use of compiler technology to optimize the use of registers
A limited and simple instruction set
An emphasis on optimizing the instruction pipeline.

Over the history of computing **Hardware cost drops dramatically and software cost goes up. Software cost is due to increasing complexity and a heavy reliance on people.**

The difference between the operations provided in HLL's and those provided by computer architecture is called the *Semantic GAP*. In other words, HLL's demand more and more complex operations not normally provided in a CPU.

Hardware designers have attempted to close the semantic gap by a**dding more complex addressing modes and more complex instructions.**

To determine what type of operations are needed by software, researchers do both static and dynamic studies.
Static is done by looking at software listing.
Dynamic is done by running a program and examining what instructions actually get executed and in what order.

With regard to operations performed the studies indicate that **Assignment and if statements predominate.**

Stallings 2012 edition

7. With regard to Table 15.2 p. 540:

Table 13.2 Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Dynamic occurrence is the p**ercent of time these instructions show up in a running program.**

The term *Machine-Instruction weighted* means **the product of the dynamic occurrence times the number of machine instruction each HLL instruction generates. Note that even though there are lots of assignment statements these generate relatively few machine instructions.**

The term *Memory Reference weighted* means **the product of the dynamic occurrence times the number of memory references generated. Note that there are relatively few calls but each generates a number of memory references.**

With regard to operands in typical HLL's references to simple scalar variables are most often done. Most are local.

The most time consuming operations performed in typical HLL's are procedure calls.

We conclude that that new architectures should have lots of registers. **This optimizes the operand referencing.**

Not that it is also important to make the instruction pipeline rather simple and optimized because a l**arge number of procedure calls cause pipeline disruption which is worse if pipe is complex.**

A large number of registers is needed to optimize operand referencing but a large number of registers would make the operand size and hence the instruction size large. On a RISC machine this is handled with **Register windows which are a large number of overlapping sets of registers.**

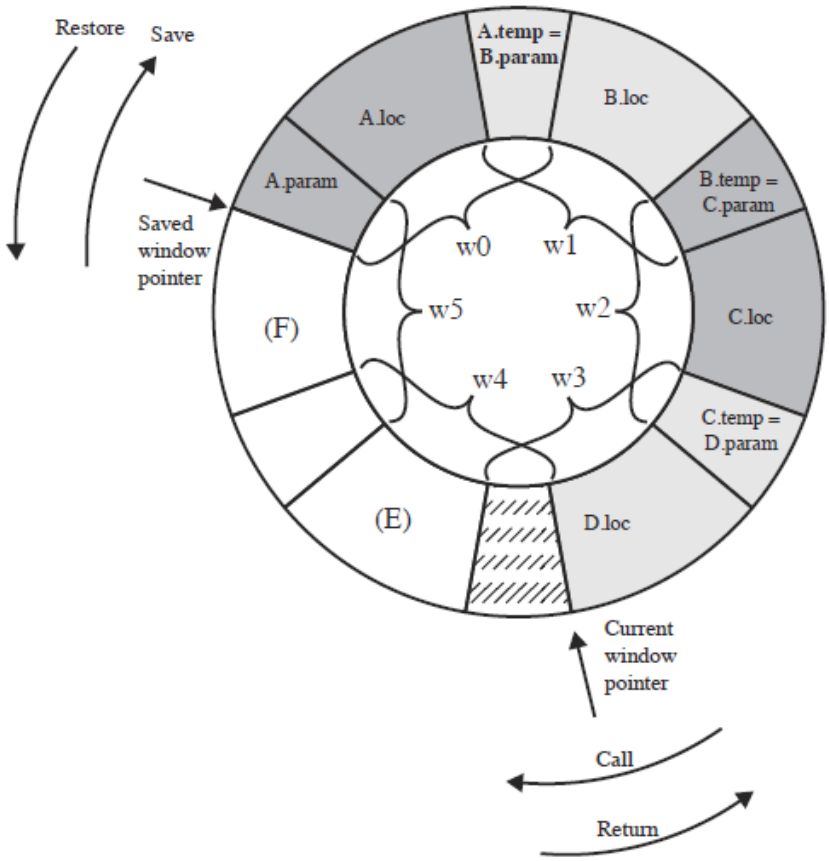

Figure 13.2 Circular-Buffer Organization of Overlapped Windows

Each window has three parts, parameter registers, local registers, and temporary registers. The temporary registers of one window overlap the parameter registers of the next window. Hence no stack is needed to pass parameters.

You need one window for each nested procedure call. When you run out of windows you must resort to stack.

Global variables are handled differently. They h**ave either their own area of memory or their own register set on chip. These are shared by all and hence can't be in local registers.**

Note that a windows based register file is superior to an on board cache because t**he addressing is much simpler since no tag or other cache hardware is needed.**

Compilers can do register optimization. This is t**he choice by the compiler of which variable get to reside in registers.**

The figure below is an example of compiler based register optimization. **6 symbolic register to be mapped to three actual registers. Symbolic registers are joined by a line if they are live during the same program fragment. Color the graph with three colors, one for each actual register. Two symbolic registers joined by a line cannot have the same color. In this example register F cannot be colored so it must go into memory.**

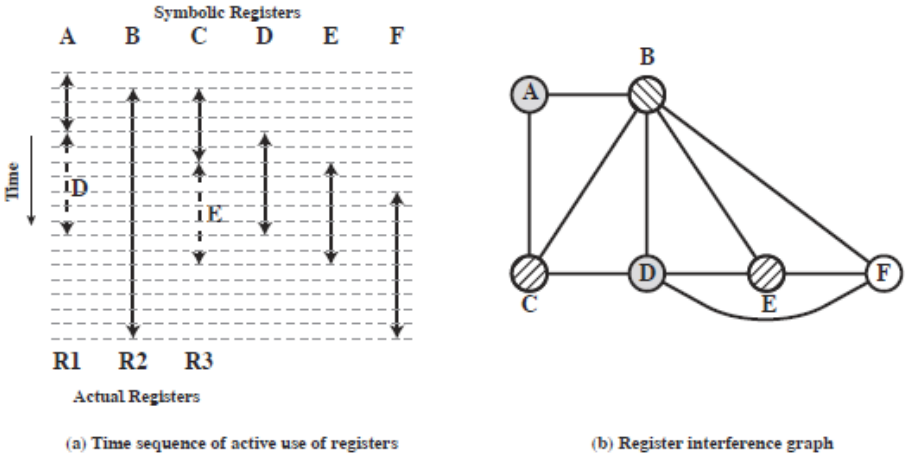

Figure 13.4 Graph Coloring Approach

The reasons for building a CISC machine are **to simplify compiler and need to improve performance.**

But complex machine instructions do not necessarily simplify the compiler. **They are hard to exploit and therefore are not often used. Complex instructions tend to require peculiar circumstances.**

CISC programs may require fewer HLL instructions but they typically require more machine instruction space simply because they are larger.

The author lists four characteristics of RISC machines.
One instruction per cycle
register to register operations
simple addressing modes
simple instruction formats.

With compiler optimization most programs do not use more than 32 registers and 16 is enough in most cases.

RISC instructions typically operate at 1 instruction per cycle b**ecause they are so simple. They do not require microcode and can be hardwired.**

Register to register operations result in a *load and store* architecture. T**he only memory operations are load from memory to a register and store a register in memory.**

The advantage of having simple addressing modes is that it s**implifies controller and pipeline design and saves room on chip for other things such as cache or more registers.**

The benefit of having a simple instruction format is that it s**implifies control unit, no wasted space because of differing instruction lengths and straddling word boundaries in memory, microcode not necessary, and a simplified pipe.**

****

RISC PIPELINING

A typical RISC pipeline can be 2, 3, or 4 stages:

```
I  Fetch       I Fetch          I Fetch
E  Execute     E Execute        E1 Register read
               D memory op      E2 Register write
                                D memory op
```

Consider the following sequence.

```
load a←m
load b←m
add c←a+b
store m←c
branch x
```

Here is how it passes through an IED pipe.

```
load a←m  I E D
load b←m    I E D
nop           I E _
add c←a+b       I E _
store m←c         I E D
branch x            I E _
nop                   I E _
```

The nop has been added between the load and add due to a d**ata dependency. We can not add b until it is fetched.**

There is also a nop after the branch because we m**ust fetch something that is not executed.**

Note that the instruction after a branch is always fetched but may not be executed if the branch is taken. This instruction is replaced with an instruction that needs to be executed in any case. This is called a *Delayed Branch*.

Here is how a delayed branch might work.

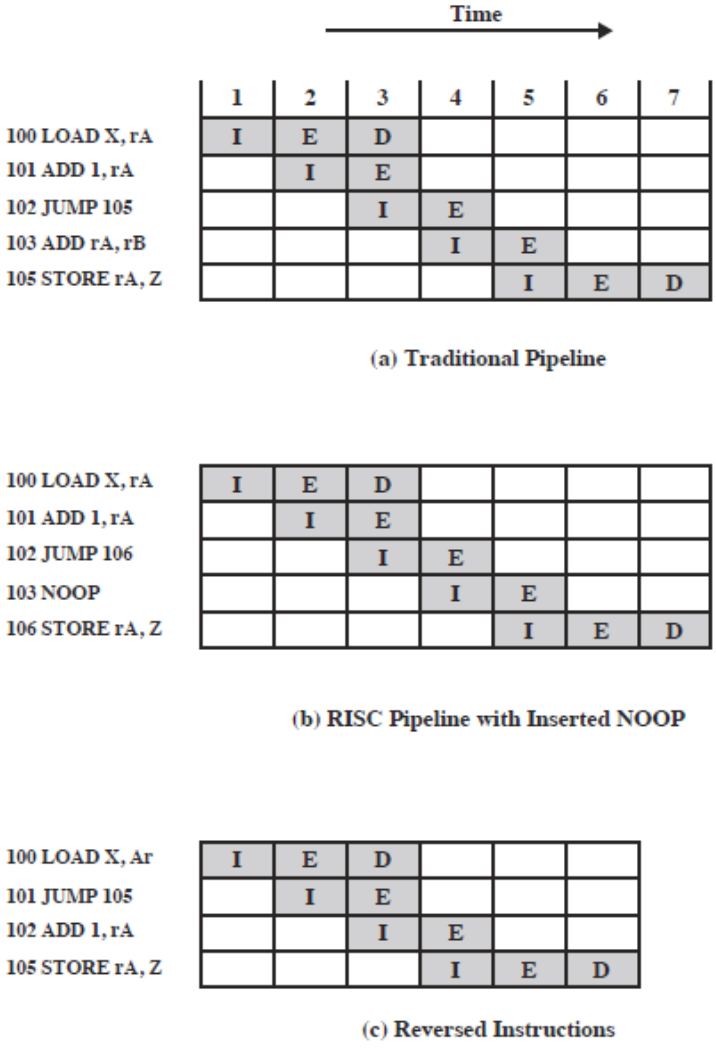

Figure 13.7 Use of the Delayed Branch

A delayed load is s**imilar to a delayed branch. Instruction following a load is fetched but its execution is delayed since its data is not loaded. Use a different but useful instruction following a load.**

The instructions for a delayed branch or a delayed load are rearranged by the **compiler or assy language programmer.**

Ultimately the limit on the number of stages in a pipeline is determined by h**ow many small pieces an instruction can be broken into.**

Stallings 2012 edition

SUN SPARC

The Sun SPARC has **136 physical registers broken into 8 windows. Each window has 8 ins, 8 locals, and 8 outs. All processes see 32 registers which include a 24 bit window and 8 global registers. Note that the 8 outs of procedure n overlap the 8 ins of procedure n+1.**

Windowing speed up processing because it is n**ot necessary to use a stack to pass parameters.**

The instructions on the SUN Sparc use:
Variable length op code.
Instruction fields generally line up for easy decoding
5 bit operands
13 bit immediate constants

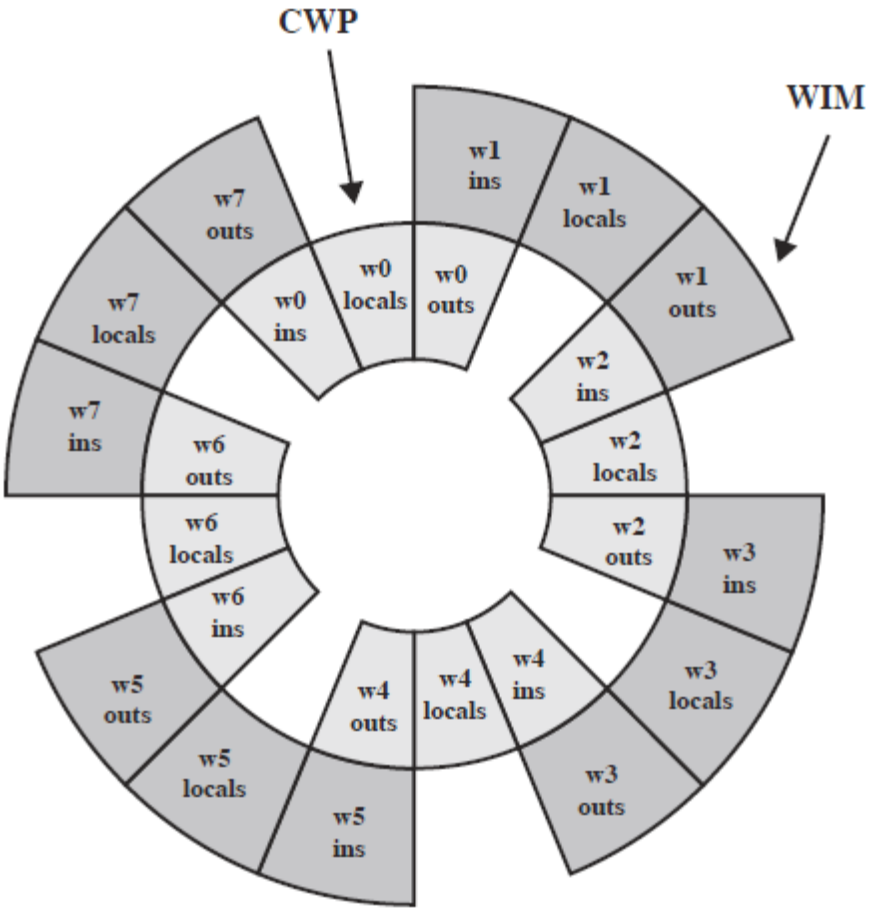

Figure 13.13 Eight Register Windows Forming a Circular Stack in SPARC

Note that it is very difficult to compare RISC and CISC machines. **There are no RISC and CISC machines of comparable vintage and gate complexity, there is no benchmark program to use as a basis for comparison, some effects are the result of hardware and not architecture and it is difficult to sort out which is which.**

The RISC/CISC controversy was resolved in the 1990s. **Machines now have a mix of both.**