Chapter 13 – Reduced Instruction Set Computers

Major architectural advances since the stored program concept in 1950 include (but are not limited to) the following:

1. The Family of Computers - companies such as IBM, DEC, Sperry, and so on developed lines of computers around the same architecture with different price/performance ranges.
2. Microprogrammed Control - provided flexibility and ease in regard to developing the control unit.
3. Cache Memory - a great performance gain by not having to access main memory for everything.
4. Pipelining - introduced parallelism into the architecture.
5. Multi-processing - the use of multiple processors in various computer organizations.

We now add RISC to the above list with its characteristics:

1. Small/Simple Instructions Set
2. Large number of GPRs
3. Emphasis on instruction pipelining

Table 13.1 shows some evolutions of computers over the years. The characteristics of RISC vs CISC is evident.

Table 13.1 Characteristics of Some CISCs, RISCs, and Superscalar Processors

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

How did we get to this CISC state anyway?

As the cost of hardware has dropped, the cost of software has risen.

The response from researchers has been to develop more powerful and complicated HLLs.

This resulted in the *semantic gap*.

How did designers respond to the semantic gap?

1. Large instruction sets.
2. Dozens of addressing modes.
3. Various HLL statements implemented in hardware.

How do the above help?

1. Ease the task of the compiler writer.
2. Improve execution efficiency.
3. Provide support for even more complicated HLLs.

For architects to decide how to build a system, it is useful to study HLLs in detail. That way, architects can provide better support.

To understand the line of reasoning of RISC advocates, we're going to investigate:

1. Operations Performed.
2. Operands Used.
3. Execution Sequencing

Operations

	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%

Operands

	Pascal	C	Average
Integer Constant	16%	23%	20%
Scalar Variable	58%	53%	55%
Array/Structure	26%	24%	25%

Procedure Calls

1. Very time consuming
2. Depends on number of parameters passed
3. Depends on level of nesting
4. Most programs do not do a lot of calls followed by lots of returns
5. Most variables are local
6. (c.f. locality of reference)

Large Register File

1. Software solution
    a. Require compiler to allocate registers
    b. Allocate based on most used variables in a given time
    c. Requires sophisticated program analysis
2. Hardware solution
    a. Have more registers
    b. Thus more variables will be in registers

Registers for Local Variables

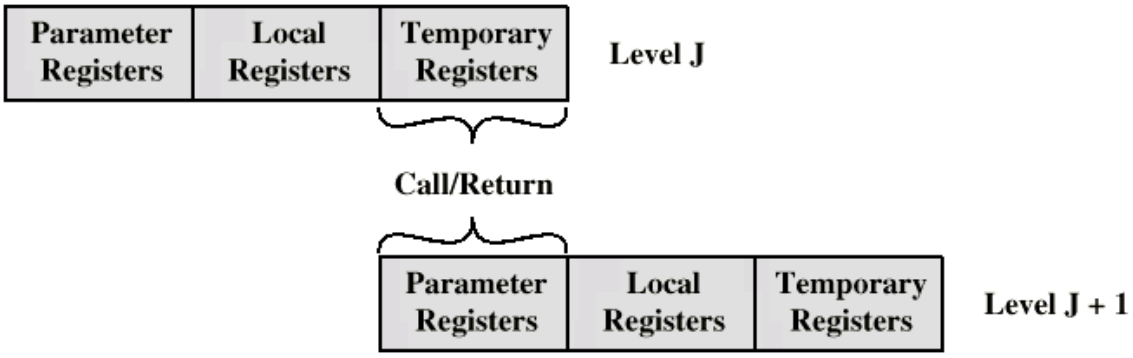
1. Store local scalar variables in registers
2. Reduces memory access
3. Every procedure (function) call changes locality
4. Parameters must be passed
5. Results must be returned
6. Variables from calling programs must be restored

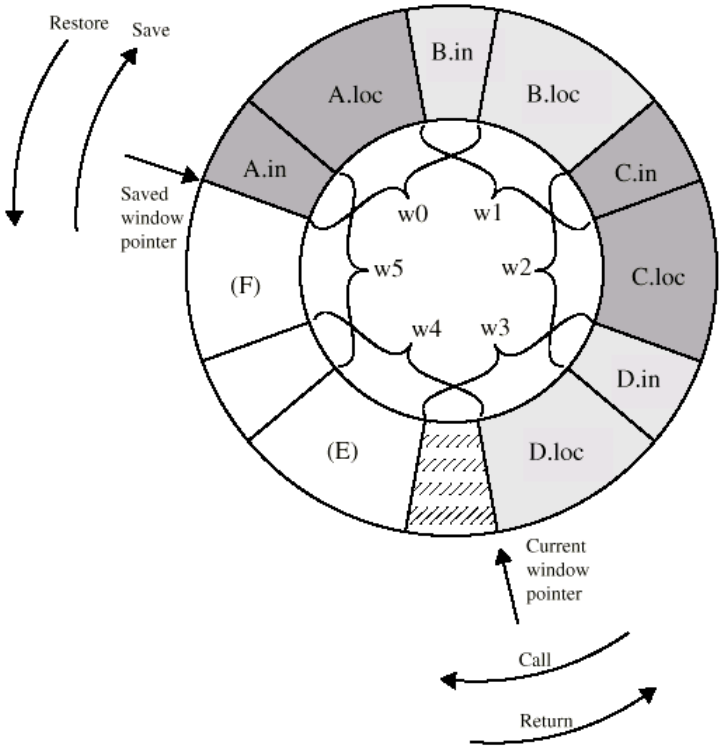
Register Windows

1. Only few parameters
2. Limited range of depth of call
3. Use multiple small sets of registers
4. Calls switch to a different set of registers
5. Returns switch back to a previously used set of registers

Three areas within a register set

1. Parameter registers
2. Local registers
3. Temporary registers
4. Temporary registers from one set overlap parameter registers from the next
5. This allows parameter passing without moving data



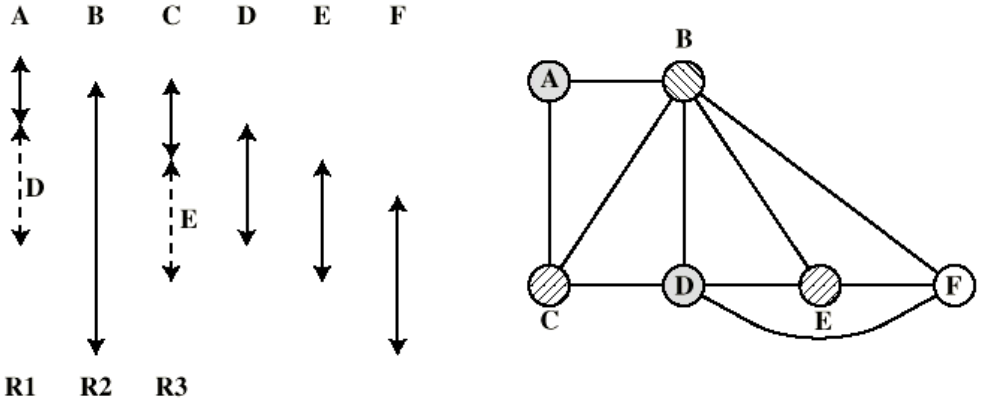


Compiler Based Register Optimization

1. Assume small number of registers (16-32)
2. Optimizing use is up to compiler
3. HLL programs have no explicit references to registers
4. Assign symbolic or virtual register to each candidate variable
5. Map (unlimited) symbolic registers to real registers
6. Symbolic registers that do not overlap can share real registers
7. If you run out of real registers some variables use memory

Graph Coloring

1. Given a graph of nodes and edges
2. Assign a color to each node
3. Adjacent nodes have different colors
4. Use minimum number of colors
5. Nodes are symbolic registers
6. Two registers that are live in the same program fragment are joined by an edge
7. Try to color the graph with n colors, where n is the number of real registers
8. Nodes that cannot be colored are placed in memory



(a) Time sequence of active use of registers

(b) Register interference graph

Why CISC

1. Compiler simplification?
    a. Disputed…
    b. Complex machine instructions harder to exploit
    c. Optimization more difficult
2. Smaller programs?
    a. Program takes up less memory but…

    b. Memory is now cheap
    c. May not occupy less bits, just look shorter in symbolic form
        i. More instructions require longer op-codes
        ii. Register references require fewer bits
3. Faster programs?
    a. Bias towards use of simpler instructions
    b. More complex control unit
    c. Microprogram control store larger
    d. thus simple instructions take longer to execute

It is far from clear that CISC is the appropriate solution

RISC Characteristics

1. One instruction per cycle
2. Register to register operations
3. Few, simple addressing modes
4. Few, simple instruction formats
5. Hardwired design (no microcode)
6. Fixed instruction format
7. More compile time/effort

RISC v CISC

1. Not clear cut
2. Many designs borrow from both philosophies
3. e.g. PowerPC and Pentium II

Controversy

1. Quantitative
    a. compare program sizes and execution speeds
2. Qualitative
    a. examine issues of high level language support and use of VLSI real estate
3. Problems
    a. No pair of RISC and CISC that are directly comparable
    b. No definitive set of test programs
    c. Difficult to separate hardware effects from complier effects
    d. Most comparisons done on “toy” rather than production machines
    e. Most commercial devices are a mixture