COMPUTER ORGANIZATION AND ARCHITECTURE (COA)

EET 2211

4TH SEMESTER – CSE & CSIT

CHAPTER 15, LECTURE 32

By Ms. Arya Tripathy

REDUCED INSTRUCTION SET COMPUTERS

TOPICS TO BE COVERED

- ✓ Instruction Execution Characteristics
- ✓ The Use of a Large Register File
- ✓ Compiler-Based Register Optimization
- ✓ Reduced Instruction Set Architecture
- ✓ RISC Pipelining
- ✓ MIPS R4000
- ✓ SPARC
- ✓ RISC versus CISC Controversy

Contd.

- **15.1 Instruction Execution Characteristics**

Operations

Operands

Procedure Calls

Implications

- **15.2 The Use of a Large Register File**

Register Windows

Global Variables

Large Register File versus Cache

- **15.3 Compiler- Based Register Optimization**

LEARNING OBJECTIVES

✓Provide an overview research results on instruction execution characteristics that motivated the development of the RISC approach.

✓Summarize the key characteristics of RISC machines.

✓Understand the design and performance implications of using a large register file.

✓Understand the use of compiler-based register optimization to improve performance.

✓Discuss the implication of a RISC architecture for pipeline design and performance.

✓List and explain key approaches to pipeline optimization on a RISC machine.

INTRODUCTION

Major Advances in computer

- **The family concept** (a set of computers offered with different price/performance characteristics, that presents the same architecture to the user)
- **Microprogrammed control unit** (eases the task of designing and implementing the control unit and provides support for the family concept)
- **Cache memory** (improves performance)
- **Pipelining** (a means of introducing parallelism into the essentially sequential nature of a machine-instruction program)
- **Multiple processors** (many processors covers a number of different organizations and objectives)
- Reduced instruction set computer (**RISC**) architecture

Contd.

Key features

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

- 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.

Contd.

The table below compares several RISC and non-RISC systems.

	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

DRIVING FORCE FOR COMPLEX INSTRUCTION SETS

- Software costs far exceed hardware costs in the life-cycle of a system (due to chronic shortage of programmers).
- Increasingly powerful and complex high level languages (HLLs).
- Semantic gap-Difference between the operations provided in HLL and computer architecture (symptoms of the gap are execution inefficiency, excessive machine program size and compiler complexity).

Designers and architectures intended to reduce the gap by :

1. Large instruction sets
2. More addressing modes
3. Hardware implementation of HLL statements

e.g CASE machine instruction on the VAX

CHARACTERISTICS OF HIGH LEVEL PROGRAMMING LANGUAGE(HLLS)

- Allow the programmer to express algorithms more concisely
- Allow the compiler to take care of details that are not important in the programmer's expression of algorithms
- Often support naturally the use of structured programming and/or object-oriented design.

INTENTION OF COMPLEX INSTRUCTION SETS

- 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.

INSTRUCTION EXECUTION CHARACTERISTICS

- **Operations performed:** These determine the functions to be performed by the processor and its interaction with memory.
- **Operands 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.

OPERATIONS

- Assignment statements

—Simple Movement of data

- Conditional statements (IF, LOOP)

—Sequence control

Observation:

- Procedure call-return is very time consuming
- representative for contemporary complex instruction set computer (CISC) architectures which can provide guidance to those looking for more efficient ways to support HLLs.

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%

OPERANDS

Predominately local scalar variables

Optimization should concentrate on accessing and storing local scalar variables .

Dynamic Percentage of Operands

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

PROCEDURE CALLS

- Procedure calls and returns are an important aspect of HLL programs.
- Very time- consuming
- Depends on number of parameters and variable s
- Depends on depth of nesting
- A high proportion of operand references is to local scalar variables
- A program remains confined to a rather narrow window of procedure-invocation depth.

Contd.

Procedure Arguments and Local Scalar Variables

Percentage of Executed Procedure Calls With	Compiler, Interpreter, and Typesetter	Small Nonnumeric Programs
> 3 arguments	0–7%	0–5%
> 5 arguments	0–3%	0%
> 8 words of arguments and local scalars	1–20%	0–6%
> 12 words of arguments and local scalars	1–6%	0–3%

IMPLICATIONS

- Best support is provided by optimizing:

-most utilized features

-most time consuming

Three elements to Characterize RISC

- use a large number of registers or use a compiler to optimize register usage.
- the design of instruction pipelines
- an instruction set consisting of high-performance

primitives is indicated

THE USE OF A LARGE REGISTER FILE

LARGE REGISTER FILE

- It is the fastest available storage device, faster than both main memory and cache.
- The register file is physically small, on the same chip as the ALU and control unit, and employs much shorter addresses than addresses for cache and memory.
- Most frequently accessed operands to be kept in registers to minimize register-memory operations.

- Software solution

-Require compiler to allocate registers

-Allocate based on most used variables in a given time

-Requires sophisticated program analysis

- Hardware solution

-Have more registers

-More variables will be in registers

REGISTER WINDOWS

- A large set of registers should decrease the need to access memory due to operands of local scalar
- only a few passed parameters and local variables
- Limited range of depth of call
- Use multiple small sets of registers
- Calls switch to a different fixed size window of registers
- Windows for adjacent procedures are overlapped

  to allow parameter passing.
- register windows can be used to hold the few most recent procedure activations.
- Older activations must be saved in memory and later restored when the nesting depth decreases.

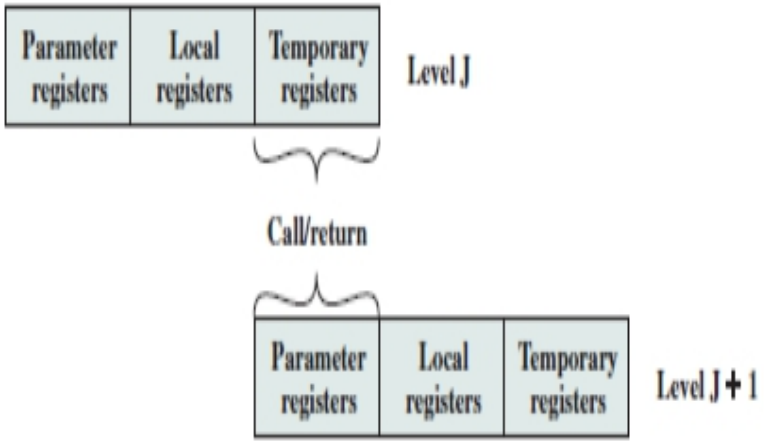
Contd.

The window is divided into three fixed-size areas.

- Parameter registers hold parameters passed down from the procedure that called the current procedure and hold results to be passed back up.
- Local registers are used for local variables, as assigned by the compiler.
- Temporary registers are used to exchange parameters and results with the next lower level (procedure called by current procedure).
- The temporary registers at one level are physically the same as the parameter registers at the next lower level.
- the parameter and local registers at level *J are disjoint from the local and temporary registers at level J + 1*.

OVERLAPPING REGISTER WINDOWS

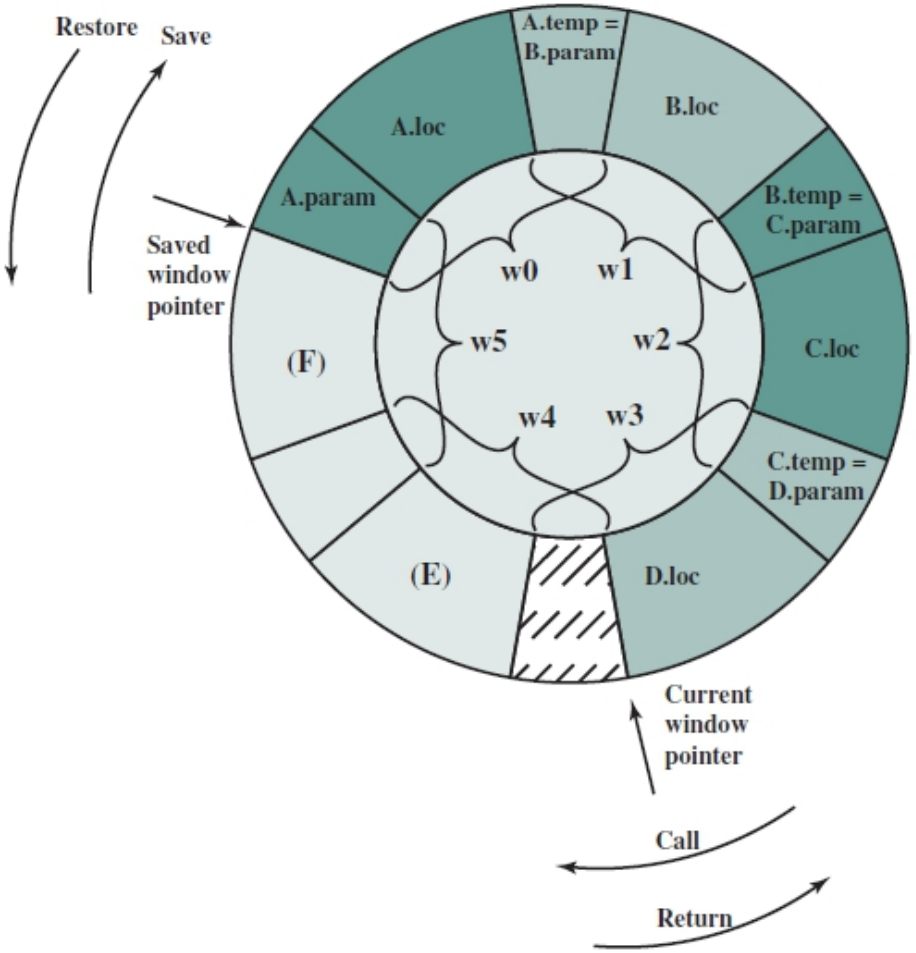
- At any time, only one window of registers is visible and it is addressable as if it were the only set of registers.
- The window is divided into 3 fixed areas.
- Parameter registers hold parameters passed down from the procedure that called the current procedure and holds results to be passed back up.
- Local registers are used for local variables, assigned by the compiler.
- Temporary registers are used to exchange parameters and results with the next lower level.



- Temporary registers at one level are physically the same as parameter registers at the next lower level.
- This overlap permits parameters to be passed without the actual movement of data.

CIRCULAR- BUFFER ORGANIZATION OF OVERLAPPED WINDOWS

- It shows a circular buffer of six windows.
- The buffer is filled to a depth of 4 (A called B; B called C; C called D) with procedure D active.



OPEARTION OF CIRCULAR BUFFER

- The current-window pointer (CWP) points to the window of the currently active procedure.
- Register references by a machine instruction are offset by this pointer to determine the actual physical address.
- The saved window pointer (SWP) identifies the window most recently saved in memory.
- If all windows are in use, an interrupt is generated and the oldest widow is saved to memory

GLOBAL VARIABLES

1ST METHOD: Variables declared as global in an HLL can be assigned memory locations by the compiler, and all machine instructions that reference these variables will use memory-reference operands.

2ND METHOD: To incorporate a set of global registers in the processor. These registers are fixed in numbers and available to all procedures.

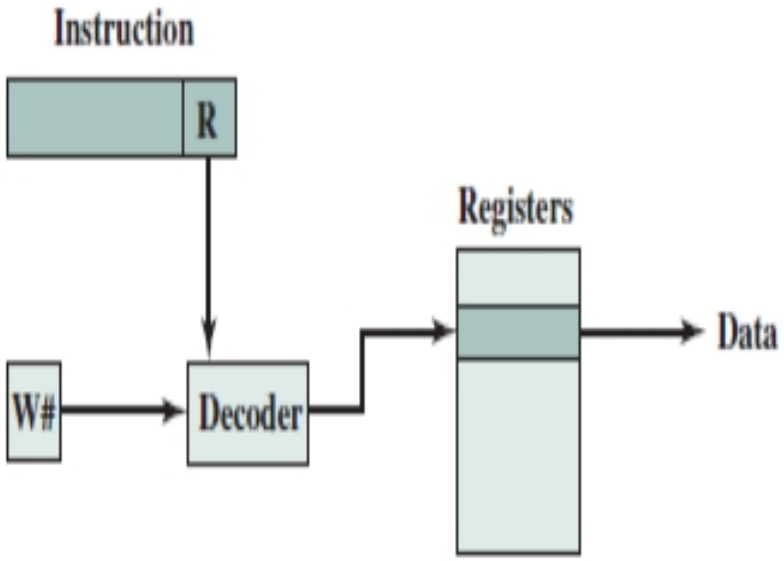
LARGE REGISTER FILE VERSUS CACHE

- The register file organized into windows acts as small, fast buffer for holding a subset of all variables that are likely to be used the most heavily.
- Therefore the register file acts much like a cache memory, although much faster memory.
- Characteristics of Large-Register-File and Cache Organizations.

Large Register File	Cache
All local scalars	Recently-used local scalars
Individual variables	Blocks of memory
Compiler-assigned global variables	Recently-used global variables
Save/Restore based on procedure nesting depth	Save/Restore based on cache replacement algorithm
Register addressing	Memory addressing
Multiple operands addressed and accessed in one cycle	One operand addressed and accessed per cycle

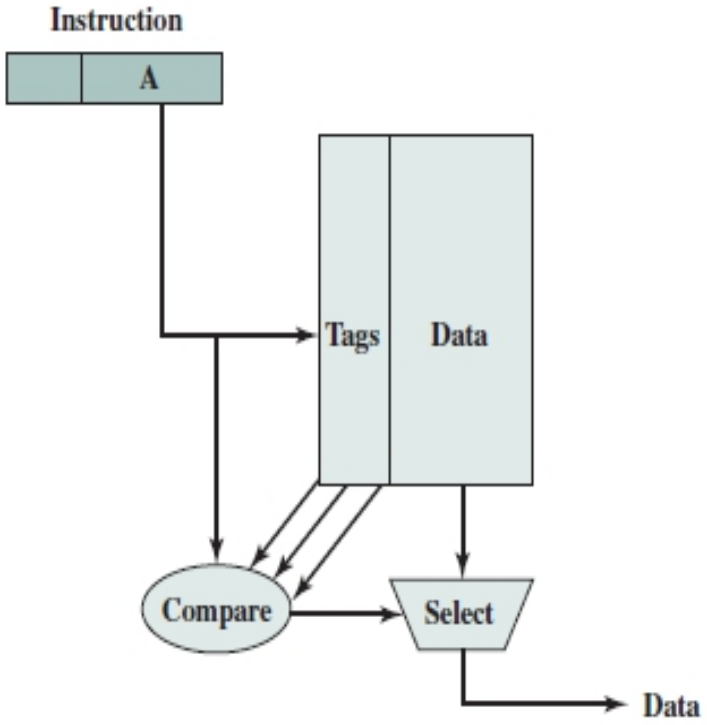
Referencing a scalar- (a)Window based register File

- To reference a local scalar in a window-based register file, a virtual register number and a window number are used .
- These can pass through a relatively simple decoder to select one of the physical registers.
- From performance point of view, the window-based register file is superior for local scalars.



Referencing a scalar-(a)Cache

- To reference a memory location in cache, a full-width memory address must be generated.
- The complexity of this operation depends on the addressing mode.
- In a set-associative cache, a portion of the address is used to read a number of tags and one of the words.
- Even if the cache is as fast as the register file, the access time will be larger.



COMPILER-BASED REGISTER OPTIMIZATION

Contd.

- Assume only a small number of registers (16-32) is available on the target RISC machine.
- Optimizing use is the responsibility of the compiler.
- HLL programs have no explicit references to registers.
- Program quantities are referred to symbolically.
- The objective of the compiler is to keep the operands for as many computations as possible in registers rather than in main memory and to minimize load-and-store operations.

Contd.

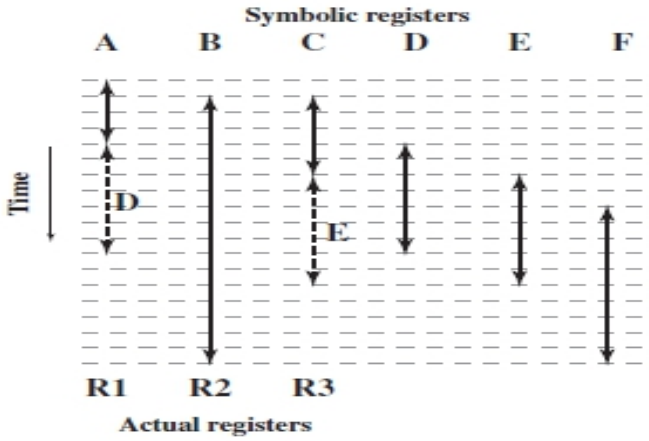
APPROACH

- Each program quantity that is a candidate for residing in a register, is assigned to a symbolic or virtual register.
- The compiler then maps the unlimited number of symbolic registers to real registers.
- Symbolic registers that do not overlap can share real registers
- With a small number of registers (e.g., 16), a machine with a shared register organization executes faster than one with a split organization.

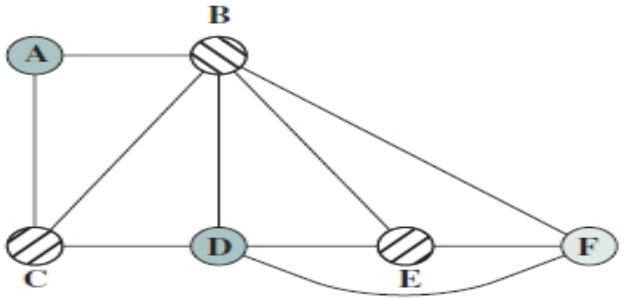
GRAPH COLORING

- Given a graph consisting of nodes and edges, assign colors to nodes such that adjacent nodes have different colors and minimize the number of different colors.
- Compiler first analyzes the program to build a register interference graph (nodes of the graphs are symbolic registers).
- If two symbolic registers are LIVE during the same program fragment, then they are joined by an edge to depict interference.
- Then an attempt is made to color the graph with n colors, where n is the number of registers.
- Nodes that share the same color can be assigned the same register.
- If this process does not fully succeed, then those nodes that can not be colored must be placed in memory, and loads and stores must be used to make space for the affected quantities when they are needed.

Graph coloring Approach



(a) Time sequence of active use of registers



(b) Register interference graph

a) Shows the time sequence of active use of each symbolic register. The dashed horizontal lines indicate successive instruction executions.

b) Shows the register interference graph (shading and stripes are used instead of colors). A possible coloring with 3 colors is indicated.

THANK YOU