Advanced Computer Architecture
(0630561)

Lecture 3

RISC Performance

Prof. Kasim M. Al-Aubidy

Computer Eng. Dept.

RISC Performance:

- A large number of registers is useful for storing intermediate results and for optimizing operand references.
- The advantage of register storage as opposed to memory storage is that registers can transfer information to other registers much faster than the transfer of information to and from memory.
- Register-to-register operations can be minimized by keeping the most frequent accessed operands in registers. For this reason, a large number of registers in the processing unit are associated with RISC processors.

Example:

An illustrative example with the following assumption:

- A program with 80% of executed instructions being simple and 20% complex.
- CISC: simple instructions take 4 cycles, complex instructions take 8 cycles; cycle time is 100 ns.
- RISC: simple instructions are executed in one cycle; complex operations are implemented as a sequence of instructions (14 instructions on average); cycle time is 75 ns.

How much time takes a program of 1 000 000 instructions?

- CISC: $(10^6 \times 0.80 \times 4 + 10^6 \times 0.20 \times 8) \times 10^{-7} = 0.48$ s
- RISC: $(10^6 \times 0.80 \times 1 + 10^6 \times 0.20 \times 14) \times 0.75 \times 10^{-7} = 0.27$ s

Register to Register Operation

- **Load-and-store architecture**
  - Only LOAD and STORE instructions reference data in memory.
  - All other instructions operate only with registers
- This characteristic simplifies the instruction set and therefore the control unit.
  - A RISC instruction set may include only 1 or 2 ADD instructions (e.g. integer add, add with carry)
  - VAX has 25 different ADD instructions
- This architecture encourages the optimization of register use, so that frequently accessed operands remain in high-speed storage.

Register to Register Operation

A large number of registers is available.

- Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory.
- All local variables of procedures and the passed parameters can be stored in registers.
- The large number of registers is typical for RISC, because the reduced complexity of the processor means that we have silicon space on the processor chip to implement them. This is usually not the case with CISC machines.

R to R and M to M Approaches:

8	16	16	16
Add	B	C	A

Memory to memory

I = 56, D= 96, M = 152

8	4	16	
Load	RB	B	
Load	RC	C	
Add	RA	RB	RC
Store	RA	A	

Register to memory

I = 104, D = 96, M = 200

I= Size of executed instructions.
D= Size of executed data.
M=I+D= Total Memort traffic

(a) A ← B + C

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memory to memory

I = 168, D= 288, M = 456

8	4	4	4
Add	RA	RB	RC
Add	RB	RA	RC
Sub	RD	RD	RB

Register to Register

I = 60, D = 0, M = 60

(b) A ← B + C; B ← A + C; D ← D – B

Register Windows:

- Procedure CALL and RETURN occurs in HL programming languages. When translated into machine language, a procedure **CALL** produces a sequence of instructions that:
  - Save register values.
  - Pass parameters needed for the procedure.
  - Call a subroutine to execute the body of the procedure.
- After a procedure **RETURN**, the program will:
  - Restore the old register values,
  - Pass results to the calling program,
  - Return from the subroutine.

Register Windows:

- Saving and restoring registers and passing parameters and results involve time-consuming operations. To overcome this, one of the following techniques must be used:

1. Using multiple-register banks: each procedure is allocated its own bank of registers. This will eliminate the need for saving and storing register values.
2. Using the memory stack to store the parameters that are needed by the procedure, but this requires a memory access every time the stack is accessed.
3. Using overlapped register windows (RISC processors) to provide the passing of parameters and avoid the need for saving and restoring register values.

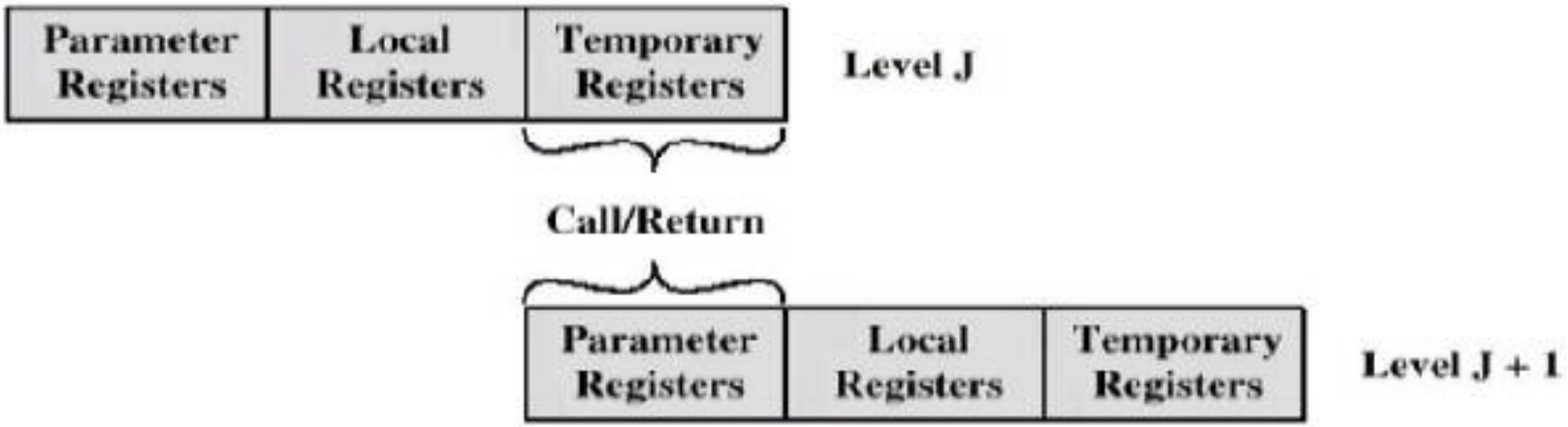
Register Windows:

For overlapped register windows:

- Each procedure CALL results in the allocation of a new window consisting of a set from the register file for use by the new procedure.
- Each procedure CALL activates a new register window by incrementing a pointer, while the RETURN statement decrements the pointer and causes the activation of the previous window.
- Windows for adjacent procedures have overlapping registers that are shared to provide the passing of parameters and results.

Register Windows:

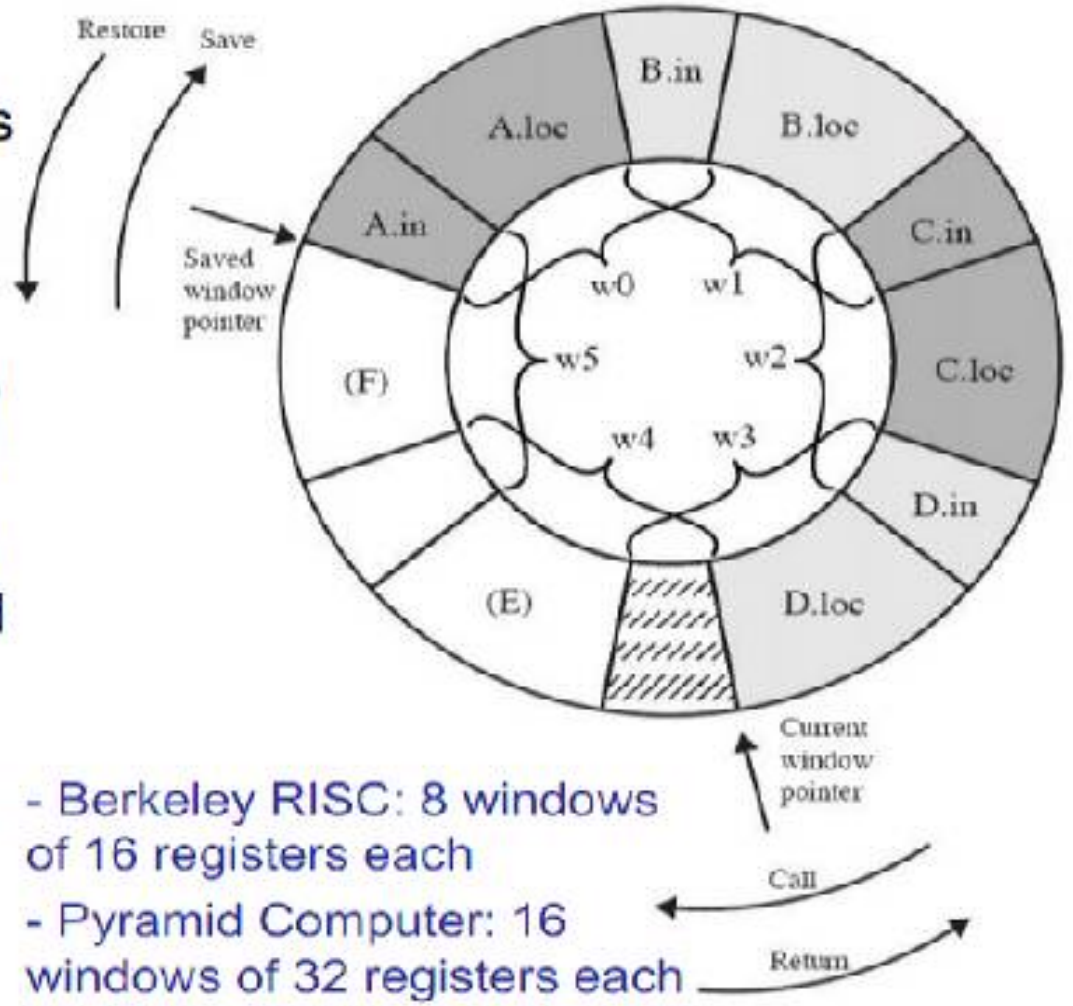
- A large number of registers is usually very useful.
- If contents of all registers must be saved at every procedure call, however, more registers mean longer delay.
- A solution to this problem is to divide the register file into a set of fixed-size windows.
  - Each register window is assigned to a procedure.
  - Windows for adjacent procedures are overlapped to allow parameter passing.



- At any time only one window of registers is visible and is addressable.
- The window is divided into three fixed-size areas:
  1. Parameter registers: hold parameters passed down from the procedure that called the current procedure and hold results to be passed back up.
  2. Local registers: used for local variables.
  3. Temporary registers: used to exchange parameters and results with the next lower level.
- The temporary registers at one level are physically the same as the parameter registers at the next lower level.
- The 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. Thus, the actual organization of the register file is as a circular buffer of overlapping windows.

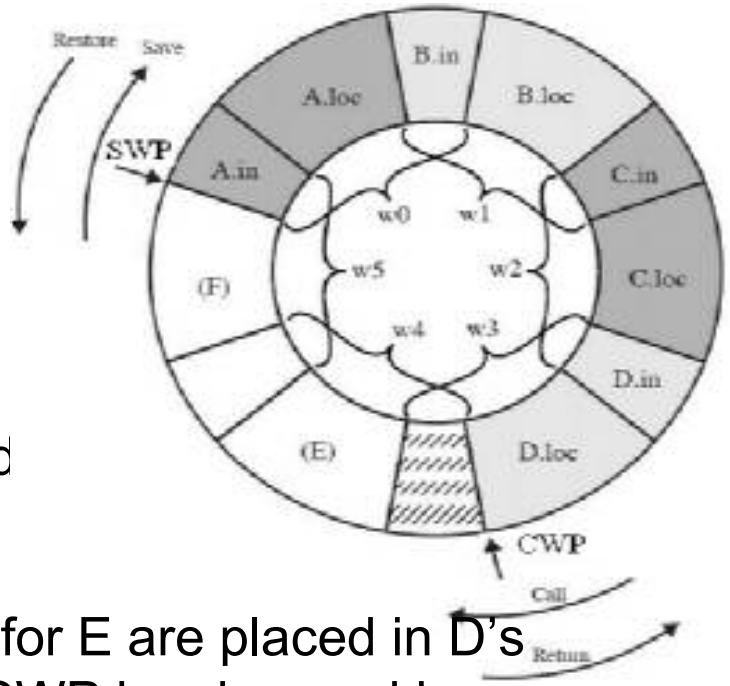
Circular Buffer Organization of Overlapped Windows:

- When a call is made, a current window pointer is moved to show the currently active register window
- If all windows are in use, an interrupt is generated and the oldest window (the one furthest back in the call nesting) is saved to memory
- A saved window pointer indicates where the next saved windows should restore to



- Berkeley RISC: 8 windows of 16 registers each
- Pyramid Computer: 16 windows of 32 registers each

- A circular buffer of 6 windows. The buffer is filled to a depth of 4 (A called B, B called C, C called D) with procedure D active.
- CWP points to the window of the currently active procedure. Reg references are offset by CWP to determine the actual physical reg.
- SWP identifies the window most recently saved in memory.



If procedure D now calls procedure E, arguments for E are placed in D's temporary regs (overlap between w3 & w4), and CWP is advanced by one window.
If procedure E then calls procedure F, the call cannot be made with the current status of the buffer. This is because window F overlaps window A.
If F begins to load its temporary registers, it will overwrite the parameter registers of A (A-in). Thus, when CWP is incremented (modulo 6) so that it becomes equal to SWP, an interrupt occurs, and window A is saved. Only the first two portions (A-in & A-loc) need to saved. Then, the SWP is incremented and the call to F proceeds.

Example:

The system has **74** registers:

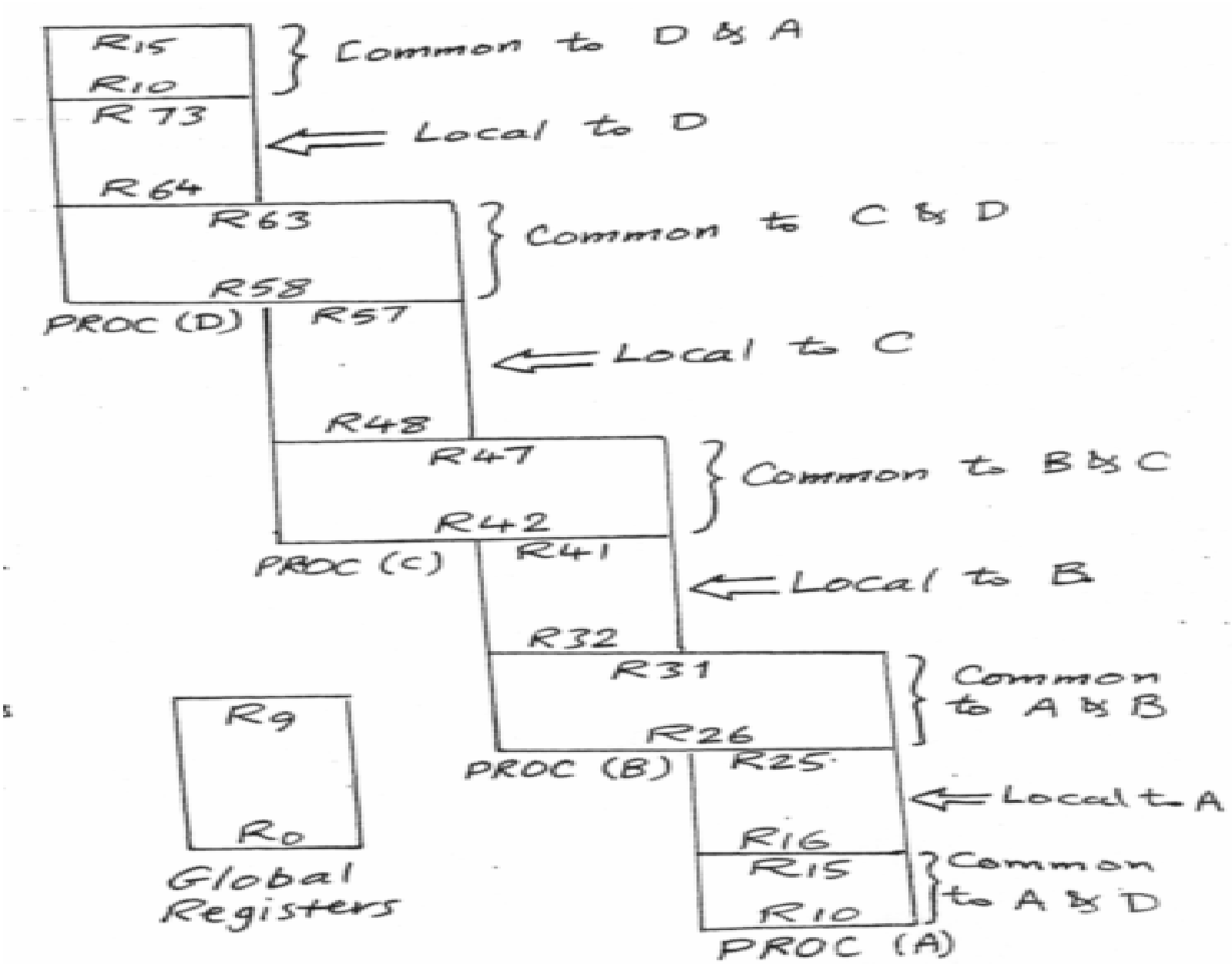
R_0-R_9: Ten hold parameters shared by all procedures.

R_{10}-R_{73}: Sixty four registers are divided into FOUR windows to accommodate procedures A, B, C & D.

- Each register window consists of 10 local registers and **TWO sets** of **6 registers common** to adjacent windows.
- **Local Registers:** used for local variables.
- **Common Registers:** used for exchange of parameters and results between adjacent procedures.
- Each procedure has 32 registers when it is active, these are;

global regs + local regs + overlapping regs

10+10+6+6= 32 Registers


R15
R10
Common to D & A
R73
Local to D
R64
R63
R58
Common to C & D
PROC (D)
R57
Local to C
R48
R47
R42
Common to B & C
PROC (C)
R41
Local to B
R32
R31
R26
Common to A & B
PROC (B)
R25
Local to A
R16
R15
R10
Common to A & D
PROC (A)
R9
R0
Global Registers

Register Window Size:

G: No. of global registers.

L: No. of local registers in each window.

C: No. of registers common to two windows

W: No. of windows.

- **Window size (S):** No. of registers available for each window.

Window Size= S= L+2C+G

- **Register file (F):** Total no. of registers needed in the processor.

Register File= F= (L+C)W+G

For the given example:

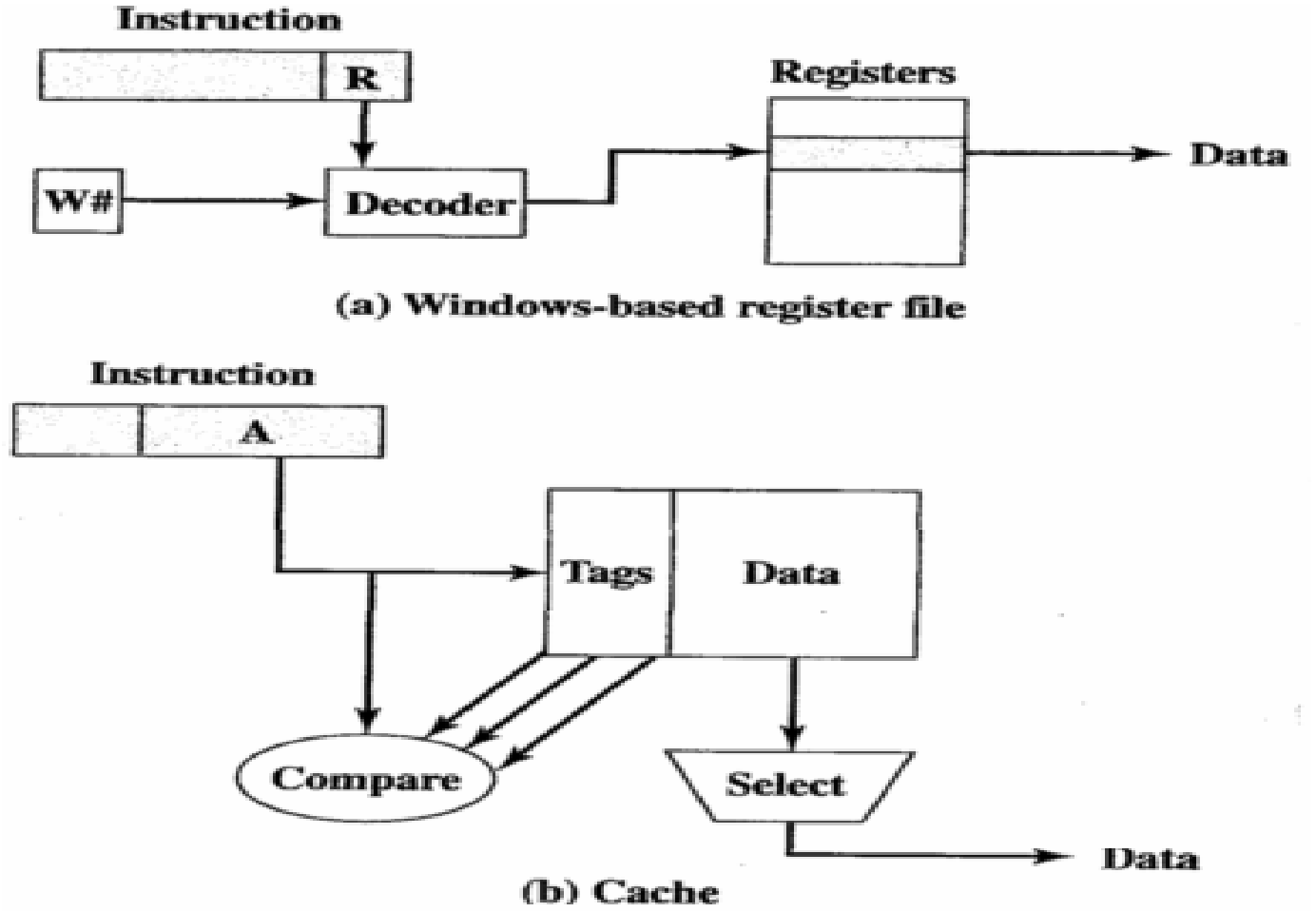
G=10, L=10, C=6 and W=4, then

S= L+2C+G= 32 regs, and F= (L+C)W+G= 74 regs

Large Register File versus Cache:

- The register file, organized into windows, acts as a small, fast buffer for holding a subset of all variables.
- The register file acts much like a cache memory.
- The window-based register file holds all the local scalar variables of most recent procedure activations. The cache holds a selection of recently used scalar variables. The following table compares characteristics of two approaches:

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



It should be clear that the even the cache memory is as fast as the register file, the access time will be considerably longer.