Q1. **State and Explain Circular Buffer with Diagram**

A circular buffer in register window organization is used in RISC architectures to efficiently manage procedure calls and returns without frequent memory access. It supports overlapping register windows, where each procedure gets a fixed-size window of registers.

Key Concepts:

* A register file is divided into windows.
* When a procedure calls another, the Current Window Pointer (CWP) shifts to the next window.
* If all windows are full, the oldest window is saved to memory (based on SWP - Saved Window Pointer).
* If procedure nesting increases and CWP = SWP, further calls cause interrupts and trigger save/restore cycles.

Diagram Summary:  
A diagram shows 6 windows in a circular layout. When procedure D is active, it uses W3. If it calls E, window shifts to W4, overlapping W3’s temporary registers. If E calls F, and buffer is full, W0 (A’s window) is saved.

Advantages:

* Enables fast parameter passing.
* Reduces memory traffic.
* Circular mechanism handles nesting efficiently.

Q2. **State and Explain Register Window with Diagram**

The register window mechanism uses a large register file split into overlapping fixed-size windows for procedure activation.

Structure of Each Window:

* Parameter Registers: For passing parameters.
* Local Registers: For local variables.
* Temporary Registers: Used to exchange parameters/results between caller and callee.

Key Features:

* Overlapping: Temporary registers in one window are shared with parameter registers of the next.
* On procedure call: Switches to a new window.
* On return: Switches back.
* Only active window’s registers are accessed, reducing load/store operations.

Hardware Example:

* Global Registers: 10
* Local: 10 per window
* Common (Overlapping): 6
* Windows: 4
* Total Registers: (10+6)\*4 + 10 = 74

Diagram Summary:  
An overlapping window diagram shows layers with parameter, local, and temporary registers shared between procedures.

Q3. **RISC and CISC Analysis and Keypoints**

RISC (Reduced Instruction Set Computer):

* Simpler instruction set.
* One instruction per cycle.
* Emphasis on compiler optimization.
* Hardwired control (no microcode).
* Register-to-register operations.
* Few addressing modes.

CISC (Complex Instruction Set Computer):

* Rich, complex instructions.
* Fewer lines of code per operation.
* Supports many addressing modes.
* Microprogrammed control unit.
* Large instruction sets.

Comparison Table:

| **Feature** | **RISC** | **CISC** |
| --- | --- | --- |
| Instructions | Few & simple | Many & complex |
| Execution | One per cycle | Variable |
| Control unit | Hardwired | Microprogrammed |
| Registers | Many | Few |
| Compiler role | Crucial | Less important |

Modern View:  
Most commercial processors (like PowerPC, Pentium) adopt hybrid approaches.

Q4. Why We Transfer from CISC to RISC

Historical Reasons:

* HLLs became dominant; procedure calls and branching increased.
* CISC couldn't efficiently handle this due to complex hardware.
* RISC design better matched software behavior.

Key Points:

* Simpler instructions → easier pipelining.
* More registers → fewer memory accesses.
* Focus on instruction-level parallelism.
* Smaller, faster control logic.
* Procedure calls optimized via register windows.

Study Findings:

* Most operations in HLLs are simple (assignment, loop, call).
* 98% of procedures use <6 arguments and <6 locals.
* Register-based operand referencing boosts performance.

Conclusion:  
RISC aligns better with HLLs and real-world usage, achieving faster, more efficient execution by simplifying the instruction set and using a register-rich design.

QUESTION: **Draw the diagram of circular-buffer organization of overlapped windows and define its working? How is it used to handle local variables for nested procedures? Three computers use register windows with the following characteristics. Determine the window size and the total number of registers in each computer: Computer 1 Computer 2 Computer 3 Global Registers 10 8 14 Local Registers 10 8 14 Common Registers 6 8 14 Number of Windows 8 4 14**  
  
  
Circular-Buffer Organization of Overlapped Windows (Register Windows)

In RISC architectures, procedure calls and returns are very frequent in high-level language programs. To handle this efficiently, overlapping register windows organized in a circular buffer are used.

🧩 Definition and Working:

A circular buffer of register windows enables fast context switching between procedure calls without needing to store and reload registers to/from memory every time.

Key Terms:

* Register Window: A subset of the full register file allocated to a specific procedure.
* Current Window Pointer (CWP): Points to the active procedure’s register window.
* Saved Window Pointer (SWP): Points to the window that was most recently saved to memory due to overflow.

When a procedure is called:

* CWP advances to the next window in the circular buffer.
* Parameters are passed using overlapping registers between windows (temporary of caller = parameter of callee).
* If buffer is full (i.e., CWP = SWP), the oldest window is saved to memory (partial save of parameter and local registers), and SWP is incremented.

When a procedure returns:

* CWP is decremented to the previous window.
* If needed, a saved window is restored from memory to the register file.

⚙️ Use for Handling Local Variables:

* Each procedure’s local variables are stored in its own local registers.
* These registers are quickly accessed and avoid costly memory access.
* Temporary registers overlap between adjacent procedure calls, helping in parameter passing.
* Since most procedures use limited arguments and locals, only a few register windows are sufficient (e.g., 8 windows can handle 99% of cases without memory save/restore).

📊 Advantages:

* Reduces overhead of saving/restoring registers to memory.
* Optimizes frequent nested calls in programs.
* Ensures fast context switch with minimal memory traffic.

|  | **Computer 1** | **Computer 2** | **Computer 3** |
| --- | --- | --- | --- |
| Global Registers | 10 | 8 | 14 |
| Local Registers | 10 | 8 | 14 |
| Common Registers | 6 | 8 | 14 |
| Number of Windows | 8 | 4 | 14 |

📗 Answer:

To calculate the required values, we use the following formulas:

* Window Size (S):  
  S = L + 2 × C + G  
  (This includes local registers, two overlapping register sets for parameter exchange, and global registers)
* Total Register File Size (F):  
  F = (L + C) × W + G  
  (Each window contributes L + C unique registers; the common part overlaps)

Now calculating for each computer:

🔹 Computer 1

* G = 10, L = 10, C = 6, W = 8
* Window Size:  
  S = 10 + 2×6 + 10 = 32
* Total Registers:  
  F = (10 + 6) × 8 + 10 = 138

🔹 Computer 2

* G = 8, L = 8, C = 8, W = 4
* Window Size:  
  S = 8 + 2×8 + 8 = 32
* Total Registers:  
  F = (8 + 8) × 4 + 8 = 72

🔹 Computer 3

* G = 14, L = 14, C = 14, W = 14
* Window Size:  
  S = 14 + 2×14 + 14 = 56
* Total Registers:  
  F = (14 + 14) × 14 + 14 = 406

📊 Summary Table:

| **Computer** | **Window Size** | **Total Registers** |
| --- | --- | --- |
| Comp 1 | 32 | 138 |
| Comp 2 | 32 | 72 |
| Comp 3 | 56 | 406 |

(EXPLAINED WAY):  
  
  
Certainly! Here's the complete, expanded explanation of the calculations, including formulas, definitions, and step-by-step solutions. This will be suitable for a full 10-mark answer.

📘 Question:  
Three computers use register windows with the following characteristics. Determine the window size and the total number of registers in each computer:

|  | **Computer 1** | **Computer 2** | **Computer 3** |
| --- | --- | --- | --- |
| Global Registers | 10 | 8 | 14 |
| Local Registers | 10 | 8 | 14 |
| Common Registers | 6 | 8 | 14 |
| Number of Windows | 8 | 4 | 14 |

📗 Full Answer:

In RISC architectures using register windows, each procedure is assigned a set of registers comprising:

* Local Registers (L): For storing local variables.
* Common Registers (C): Also called overlapping registers, used for passing parameters/results between caller and callee. Each window shares C registers with the previous and next window.
* Global Registers (G): Fixed registers available to all procedures regardless of which window is active.
* Windows (W): The total number of register windows supported by the architecture.

To calculate the window size and total register file size, we use the following formulas:

🔷 1. Formula for Window Size (S):

S = L + 2 × C + G

* This includes:
  + L local registers (used for local variables),
  + 2 × C for overlapping registers (C from previous and C for next),
  + G global registers (shared by all windows).

This gives us the number of registers that appear accessible during any single procedure's execution.

🔷 2. Formula for Total Register File Size (F):

F = (L + C) × W + G

* (L + C): Each window needs L local registers and C registers for overlap.
* W: Number of such windows.
* G: Global registers only exist once and are not duplicated.

Let’s apply the formulas step-by-step for each computer.

──────────────────────────────  
🔹 Computer 1:

* Local Registers (L) = 10
* Common Registers (C) = 6
* Global Registers (G) = 10
* Number of Windows (W) = 8

✅ Step 1: Window Size (S)

S = L + 2 × C + G  
S = 10 + 2 × 6 + 10  
S = 10 + 12 + 10 = 32 registers

✅ Step 2: Total Register File Size (F)

F = (L + C) × W + G  
F = (10 + 6) × 8 + 10  
F = 16 × 8 + 10 = 128 + 10 = 138 registers

──────────────────────────────  
🔹 Computer 2:

* L = 8, C = 8, G = 8, W = 4

✅ Step 1: Window Size (S)

S = 8 + 2 × 8 + 8  
S = 8 + 16 + 8 = 32 registers

✅ Step 2: Total Registers (F)

F = (8 + 8) × 4 + 8  
F = 16 × 4 + 8 = 64 + 8 = 72 registers

──────────────────────────────  
🔹 Computer 3:

* L = 14, C = 14, G = 14, W = 14

✅ Step 1: Window Size (S)

S = 14 + 2 × 14 + 14  
S = 14 + 28 + 14 = 56 registers

✅ Step 2: Total Registers (F)

F = (14 + 14) × 14 + 14  
F = 28 × 14 + 14 = 392 + 14 = 406 registers

📊 Final Results Summary:

| **Computer** | **Window Size (S)** | **Total Registers (F)** |
| --- | --- | --- |
| Computer 1 | 32 | 138 |
| Computer 2 | 32 | 72 |
| Computer 3 | 56 | 406 |

✅ Interpretation:

* The window size represents how many registers are available to a procedure at runtime (including global, local, and overlapping).
* The total register file size tells us how many actual hardware registers are needed in the CPU to implement this register windowing mechanism.