**Q) In the RISC architecture, what is meant by over lapping register window? Explain the relationship among register windows with over lapping register windows.**

**Overlapping register windows** are a feature found in some Reduced Instruction Set Computer (RISC) architectures. They provide a mechanism for efficient function calling and returning. This mechanism is especially useful in programming languages like C and Pascal, which frequently use function calls.

**How Overlapping Register Windows Work:**

1. Multiple Register Windows
2. Overlap
3. Window Pointer
4. Function Calls
5. Function Returns

**Multiple Register Windows:** The register file is divided into multiple register windows, each containing a set of registers.

**Overlap:** Adjacent register windows share some portion of their register. This means they have overlapping regions. This overlapping region (shared register) are used to pass arguments to functions and to return values.

**Window Pointer:** A special register called the "window pointer" keeps track of which register window is currently in use. This pointer is used for navigating between the overlapping register windows.

**Function Calls:** When a function is called:

* The window pointer is incremented to point to the next window.
* Arguments are passed into the overlapping region of the new window.
* The function's code executes in the new window.

**Function Returns:** When a function return:

* The window pointer is decremented to point to the previous window.
* The return value is placed in the overlapping region of the previous window.

**Benefits of Overlapping Register Windows:**

1. **Efficient Function Calls**
2. Reduced Memory Access
3. Simplified Compiler Implementation

**Efficient Function Calls:** By using the overlapping region, functions can directly access and share data without needing to save and restore it from registers, because data are kept in overlapping region, improving performance.

**Reduced Memory Access:** By keeping frequently used data in registers, memory accesses are reduced.

**Simplified Compiler Implementation:** The compiler can optimize function calls by placing arguments and return values in the overlapping region. This avoids data to save in register and restore from register. So, compilers can generate more optimized code, leading to faster execution.

**Relationship Among Register Windows:**

1. Sequential Register Windows
2. Circular Register Windows
3. SharingRegister Windows

**Sequential Register Windows**

Imagine a stack of plates. Each plate represents a register window. When you add a new plate to the stack, it overlaps with the top plate. This is similar to how sequential register windows work.

* **Each window** overlaps with its predecessor and successor.
* **When a function is called**, the window pointer moves to the next window, which overlaps with the current window. This overlap allows data to be passed between the two windows efficiently.
* **When a function returns**, the window pointer moves back to the previous window, again benefiting from the overlap.

**Circular Register Windows**

Think of a carousel with several seats. As the carousel spins, each seat passes by the same point. This is similar to how **circle** register windows work.

* **The windows are arranged in a circle**.
* **The window pointer can move in either direction**.
* **This allows for a larger number of windows** without increasing the total number of registers. For example, if you have 10 registers, you can create 10 circular windows, while with sequential windows you might only be able to create 5.

**Sharing Register Windows**

The overlapping region is shared between adjacent windows, enabling efficient data transfer during function calls and returns.

**Q) Explain the Flynn’s classification of computer architectures with diagrams.**

**Flynn's classification** is a way to categorize computer architectures based on how they handle instructions and data. It looks at two main factors:

1. **Instruction streams:** These are the sequences of instructions that the computer executes.
2. **Data streams:** These are the data that the computer processes.

Categories

1. SISD (Single Instruction, Single Data)
2. SIMD (Single Instruction, Multiple Data)
3. MISD (Multiple Instruction, Single Data)
4. MIMD (Multiple Instruction, Multiple Data)

**1. Single Instruction, Single Data (SISD)**

* **Instructions:** Only one instruction is executed at a time.
* **Data:** Only one data item is processed at a time.
* **Example:** A traditional desktop computer with a single processor.

**2. Single Instruction, Multiple Data (SIMD)**

* **Instructions:** Only one instruction is executed at a time, but it operates on multiple data items simultaneously.
* **Data:** Multiple data items are processed in parallel.
* **Example:** A graphics processing unit (GPU) that can perform the same operation on many pixels at once.

**3. Multiple Instruction, Single Data (MISD)**

* **Instructions:** Multiple instructions are executed simultaneously, but they operate on the same data item.
* **Data:** Only one data item is processed at a time.
* **Example:** Not widely used in modern systems

**4. Multiple Instruction, Multiple Data (MIMD)**

* **Instructions:** Multiple instructions are executed simultaneously on multiple data items.
* **Data:** Multiple data items are processed in parallel.
* **Example:** Multiprocessor systems, clusters.

**Q) What are the different types of pipeline hazards? Explain each pipeline hazard with example.**

Pipeline hazards, also known as pipeline conflicts, occur when instructions cannot execute in sequence like one after another because of problems like hardware limitations, needing the result of a previous instruction, or not knowing the next step (like in a branch). These hazards can significantly impact the performance of a processor.

Types of pipeline hazards:

1. Structural Hazards
2. Data Hazards
3. Control Hazards

**Structural Hazards**

* Occur when two or more instructions need the same hardware simultaneously and hardware resources are not available
* **Example:** A processor with a single ALU (Arithmetic Logic Unit) cannot execute two arithmetic operations in the same cycle.

**Data Hazards**

* Occur when an instruction depends on the result of a previous instruction that is still in the pipeline, leading to a stall
* **Example:** Load R1, A

Add R2, R1, B

* The **Add** instruction depends on the result of the **Load** instruction. If the Load is still in the memory access stage, the **Add** will have to stall until the data is available.

**Control Hazards**

* Occur when the pipeline makes decisions based on branch instructions and decide whether to follow one set of instructions or another set of instructions is not known.
* Example:

A computer code with black text

Description automatically generated

When the processor comes across the "**if**" statement, it needs to check the condition to see whether to go to "**label1**" or "**label2**." If it can't decide until later in the process, processor clear out the instructions known as **flushing the pipeline**. The processor has to start fetching and executing the correct instructions from scratch. This slows down the overall execution

**Types of Data Hazards**

Data hazards can be further classified into three types:

* **Read After Write (RAW):** An instruction tries to read a register before a previous instruction writes to it.
* **Write After Read (WAR):** An instruction tries to write to a register before a previous instruction reads from it.
* **Write After Write (WAW):** Two instructions try to write to the same register.

**Mitigation Techniques**

To mitigate pipeline hazards and improve performance, various techniques are employed:

* **Forwarding:** Bypassing the data from one stage to another to avoid stalls.
* **Branch Prediction:** Predicting the outcome of a branch to reduce the impact of control hazards.
* **Delayed Branches:** By inserting a no-operation (nop) instruction after a branch instruction. This delay allows the computer to calculate the correct target address for the branch, reducing the errors.
* **Pipeline Scheduling:** Rearranging instructions to minimize hazards.

Explain the computer components.

**Q) Explain the store-program concept with example.**

The store-program concept is a fundamental concept in computer architecture where both instructions and data are stored in the computer's memory. It allows the computer to fetch, decode, and execute instructions sequentially from memory. This concept was first introduced by John von Neumann.

**How it works:**

1. **Instruction Storage**
2. Program Counter
3. Fetch-Execute Cycle
4. Increment PC

**Instruction Storage**

* Instructions are stored in memory.
* Memory can be volatile (RAM) or non-volatile (ROM).

**Program Counter:** PC holds/track the address of the next instruction to be executed.

**Fetch-Execute Cycle**

1. **Fetch:** PC is used to get the instruction from memory.
2. **Decode:** Understand what the instruction means and what needs to be done.
3. **Execute:** Perform the operation specified by the instruction (like calculations).

**Increment PC:** PC is increased to point to the next instruction.

**Example**

Consider a simple program that adds two numbers, 5 and 3.

1. **Instructions:**

* Load 5 into register A
* Load 3 into register B
* Add the contents of register B to register A
* Store the result in memory location C

1. **Execution:**

* The PC starts at the address of the first instruction.
* The CPU fetches the instruction "Load 5 into register A" and decodes it.
* The CPU loads the value 5 into register A.
* The PC is incremented to the next instruction.
* The process continues for the remaining instructions.

**Q) Define associative memory. Explain with block diagram how it can be implemented**

**Associative memory**, also known as content-addressable memory (CAM), where data is accessed based on its content rather than its address. Instead of looking for data at a specific location, you can search data by providing a key that matches the data. It is often used in applications like cache memory, database searching, where finding data by its content is more practical than searching by its exact location.

Block Diagram Representation

A diagram of a computer program

Description automatically generated

**Explanation of the Block Diagram**

**Argument Register**: This is where the search query (or argument) is entered.

**Key Register:** The key register allows you to specify which parts of the argument to search for.

**Associative Memory Array:** The database containing all stored information.

**Matching Process: The** system compares the argument register's contents with each record in the associative memory, using the filter from the key register. If a record matches, then corresponding bit in the match register will be set to 1.

**Match Register:** Tracks which records (in associative memory) matched the search criteria, with bits set to 1 for matching entries.

**Results:** Displays the matched data after the search is complete.

### Advantages of Associative Memory

1. **Fast Search:** Quickly finds data without scanning the entire memory.
2. **Content-Based Retrieval:** Retrieves data based on its content instead of its memory location.
3. **Database Speedup:** Often used to enhance database performance.
4. **Parallel Processing:** Supports simultaneous searching across multiple entries. This means it can search all relevant data at once instead of checking each record one by one.
5. **Application in Virtual Memory:** Utilized in page tables and neural networks.

### Disadvantages of Associative Memory

1. **Higher Cost:** More expensive than traditional RAM.
2. **Complex Design:** Each cell requires storage and logic circuits for content matching.

### Applications of Associative Memory

1. Database Management Systems: fast data access
2. Image Processing: Used to search for specific patterns within images
3. Artificial Intelligence
4. Networking
5. Memory Allocation

**How does DMA controller work? Give an example of DMA data transfer**

DMA (Direct Memory Access) transfer is to a method of transferring data between **peripherals** devices and memory without involving the **CPU** in the actual data movement. This allows devices to communicate directly with the system memory, enabling faster data transfers and freeing up CPU resources for other tasks. This is done using a dedicated hardware component called the DMA controller.

A diagram of a computer system

Description automatically generated

How It Works / Process of DMA Data Transfer

**DMA Request:** The I/O device sends a request to the DMA controller.

**DMA Grant:** The DMA controller grants permission and takes control of the system bus.

**Data Transfer:** The device directly transfers data between its memory and main memory.

**DMA Acknowledge:** The DMA controller signals completion to the device.

**Interrupt:** The DMA controller may send an interrupt to the CPU to notify it of the completed transfer.

**Example of DMA Data Transfer**

**Scenario**: Copying a large file from a USB flash drive to the computer's memory (RAM).

**Process**:

1. **Initiation**: You insert a USB flash drive into your computer and select a large file to copy.
2. **Request**: The operating system (OS) sends a request to the DMA controller to begin the transfer, providing details about the source (USB drive), destination (RAM), and the size of the file.
3. **Grant**: The CPU acknowledges this request and gives permission for the DMA controller to access the system bus.
4. **Transfer**:
   * The DMA controller takes control and instructs the USB controller to send a block of data directly to RAM.
   * Instead of moving data byte by byte, the DMA controller transfers large blocks of data at once, significantly speeding up the process.
5. **Notification**: Once the transfer is complete, the DMA controller sends an interrupt signal to the CPU, notifying it that the operation is finished. The CPU can then check for errors and proceed with other tasks.

**Differentiate between hardwired control unit and a micro programmed control unit.**

|  |  |  |
| --- | --- | --- |
| Feature | Hardwired Control Unit | Microprogrammed Control Unit |
| Definition | Use Logic gates and flip-flops to generate control signals and execute instructions | uses a set of instructions (microinstructions) stored in memory (ROM) to generate control signals and execute instructions |
| Functionality | Fixed, specific tasks | Flexible, can be updated easily |
| Modification Requirements | Difficult to modify | Easier to modify |
| Speed | Faster due to direct hardware operations | Slightly Slower due to fetch decode execute microinstructions |
| Design Complexity | Complex | Simpler |
| Cost | High, since changing functionality requires new hardware | Lower, as updates are done through changes in microinstructions |
| Example Processor | RISC Processors | CISC Processors |