**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.**

**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.

**What is Parallel Processing?**

Parallel processing is a method which used more than one processor to perform different parts of a task at the same time.

It breaks a big job into smaller tasks, with each processor handling a different task simultaneously.

This method speeds up large and complex tasks, by doing multiple calculations at once.

It’s like having a team of workers handling different parts of a big job simultaneously instead of one person doing everything step by step.

A diagram of a process

Description automatically generated

The diagram demonstrates how data from the processor registers can be sent to multiple functional units simultaneously.

**For example:**

* If arithmetic operations are needed, data goes to the Arithmetic Unit
* If logical operations are needed, data goes to the Logic Unit

This allows for parallel execution of different operations.

How Parallel Processing Works

* **Break down the task:** Divide a large task into smaller, manageable parts.
* **Assign tasks:** Assign each part to a different processor or core.
* **Work independently:** Each processor works on its assigned part.
* **Coordinate:** If needed, processors communicate to ensure they're working together correctly.
* **Combine results:** Once all parts are finished, the results are combined to create the final output.

**Types of Parallel processing**

* Single Instruction Multiple Data (SIMD)
* Multiple Instruction Multiple Data (MIMD)
* Multiple Instruction Single Data (MISD)
* Single Program Multiple Data (SPMD)

**Instruction Stream and Data Stream**

* **Instruction stream:** The sequence of instructions read from memory.
* **Data stream:** The operations performed on data in the processor.

**Application of Parallel Processing**

**Weather Forecasting:** Faster, more accurate predictions.

**Graphics:** Real-time visuals and faster animation.

**Data Analysis:** Faster analysis of large datasets.

**Medical Scans (like MRI or CT scans)**

**Financial Modeling:** Faster calculations and simulations.

**Cryptocurrency Mining:** Faster puzzle-solving.

Search Engines (like Google)

Self-Driving Cars and Robots

**Benefits of Parallel Processing**

**Faster tasks:** Work gets done quicker.

**Better efficiency:** Makes the most of your computer's resources.

**Real-time action:** Enables instant responses for things like games and self-driving cars.

**Pipeline**

1. Pipeline Structure
2. Pipeline Processing
3. Register Contents Over Time
4. Space-Time Execution

### Example: Pipeline Processing Diagram

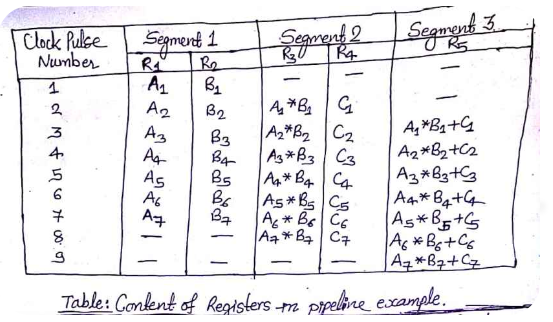
A screenshot of a diagram

Description automatically generated

**Pipeline Structure**

* Think of the pipeline as an assembly line in a factory, where tasks move from one station to the next.
* Data flows **from left to right**, meaning it starts in the first segment and moves to the last segment step by step.

**Register Contents (Table) Over Time**

****

The table has four columns:

1. **Clock Pulse Number:** Tracks the number of clock cycles.
2. **Segment 1:** Shows the task currently being processed in the first stage.
3. **Segment 2:** Shows the task currently being processed in the second stage.
4. **Segment 3:** Shows the task currently being processed in the third stage.

**Pipeline Operation:**

The table illustrates how data moves from one stage to the next.

1. **Initialization:**

* In the first clock cycle (Clock Pulse Number 1), the initial values A1 and B1 are loaded into the registers of Segment 1.

1. **Data Movement and Processing:**

* In the second clock cycle, A1 and B1 move to Segment 2, and new values A2 and B2 are loaded into Segment 1.
* In Segment 2, the operation A1 \* B1 is performed, and the result is stored in a register.
* This pattern continues, with data moving through the pipeline stages and undergoing processing at each stage.

1. **Final Output:**

* The final result of the computation is obtained in the last clock cycle. In this case, it is the value stored in the register of Segment 3.

**Key Points:**

* **Pipeline Stages:** The pipeline has three stages (segments) in this example.
* **Data Flow:** Data moves from one stage to the next in each clock cycle.
* **Parallel Processing:** The pipeline allows for parallel processing of different parts of the task.

**Register Content Table (Six Segment Pipeline with 8 Tasks)**

* **Fetch (F)**: Fetches the instruction.
* **Decode (D)**: Decodes the instruction.
* **Execute (E)**: Executes the instruction.
* **Memory Access (M)**: Reads or writes data.
* **Write Back (WB)**: Writes the result back to the register.
* **Idle (I)**: Indicates no task is being processed in that segment.

**A table with text and numbers

Description automatically generated**

**Explanation:**

* Each row represents a clock cycle.
* Each column represents a pipeline stage.
* The letters T1, T2, ..., T8 represent the eight tasks.
* As the clock cycles progress, tasks move through the pipeline stages, with each stage performing its specific operation.

**What do you mean by pipeline? Explain with space time diagram for a six segmented pipeline showing the time it takes to process eight tasks.**

A pipeline is a technique that works like a step-by-step process, where different parts of data are handled one after the other. Each step takes care of a specific part, allowing several tasks to be worked on simultaneously, but each task is at a different stage in the process.

A diagram of a flowchart

Description automatically generated

**Figure: Six segment pipeline**

* Six processing segments (S₁ to S₆)
* Six registers (R₁ to R₆)
* A single clock signal controlling all registers
* The clock ensures synchronized data movement between segments
* Each register captures data at the clock edge
* Data flows from left to right through the pipeline
* The clock coordinates the movement of data through all stages

A grid of numbers and letters

Description automatically generated

**Figure: space-time diagram for a six-segmented pipeline processing eight tasks**

This diagram illustrates how tasks are processed in a six-segmented pipeline over time. Each row represents a pipeline stage, and each column represents a clock cycle. The letters T1, T2, ... T8 represent the eight tasks being processed.

**How the Pipeline Works:**

1. **Initialization:**

* In the first clock cycle, Task 1 (T1) enters the first stage (S1).

1. **Task Movement:**

* In the second clock cycle, T1 moves to S2, and T2 enters S1.
* In the third clock cycle, T1 moves to S3, T2 moves to S2, and T3 enters S1.

This pattern continues, with each task progressing through the stages.

1. **Completion:**

* Once a task reaches the final stage (S6), it is considered complete. For instance, T1 completes in the seventh clock cycle.

Explain an instruction pipeline with an example.

Pipeline processing is a technique which is used to improve the performance of computer systems by breaking down the execution of instructions into multiple stages allowing different parts of multiple instructions to be processed simultaneously.

**Instruction Cycle**

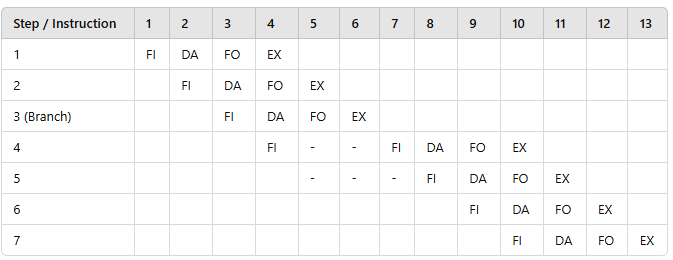
* 1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Calculate the memory address.
  4. Fetch the operands(data) from memory/register.
  5. Execute the instruction.
  6. Store the result.

**Four-Segment Pipeline Design**

A diagram of a computer program

Description automatically generated

**Figure:** Four Segment CPU Pipeline



**Figure:** Timing of Instruction Pipeline

**Explanation of Stages:**

* **IF** (Fetch Instruction) - Instruction Fetch - Retrieves the instruction from memory.
* **DA**: **(Decode/Address Calculation)**- Decodes the instruction and calculates any necessary memory addresses.
* **FO**: Fetch Operand - Fetches the operands needed for execution from memory or registers.
* **EX**: Execute - Carries out the instruction (arithmetic, logical operations, etc.).

**Stalls**: The **"-"** in the table represents a stall, indicating that an instruction has to wait for a resource or data to be available. This is common in pipelining, especially when a branch instruction is encountered.

**How is performance of computer increased using pipeline? Explain with practical example?**

Pipeline processing is a technique which is used to improve the performance of computer systems by breaking down the execution of instructions into multiple stages allowing different parts of multiple instructions to be processed simultaneously.

**How Pipelining Increases Performance**

1. Parallel Execution: (Reduces Wait Time)
2. Higher Throughput (Increases the Number of Instructions Processed)
3. Reduced Instruction Latency (Enables Faster Operation)
4. **Improved Clock Rate**
5. Supports Parallelism (Allows for Simultaneous Operations)
6. **Overlapping Execution**

### Pipelining allows different stages of multiple instructions to be processed at the same time, reducing the CPU waiting time

### Higher Throughput

By allowing new instructions to start before previous ones finish, pipelining enables the CPU to complete more instructions in a shorter period time.

1. **Reduced Instruction Latency (Enables Faster Operation)**:

Although the time to execute a single instruction might not change much, pipelining reduces the overall time needed to complete a sequence of instructions.

**Practical Example of Pipelining**

Imagine you’re in a car wash. The car wash has six steps: washing, soaping, scrubbing, rinsing, drying, and polishing. Each of these steps happens in a different section, and the car moves from one section to the next.

**How It Works:**

1. **First Car (Car 1)**: It starts in the first section (washing).
   * While Car 1 is being washed, the other sections are empty.
2. **Second Car (Car 2)**: After Car 1 moves to the next section (soaping), Car 2 enters the first section (washing).
   * Now, two cars are in the car wash, each in a different section.
3. **Third Car (Car 3)**: When Car 1 moves to the third section (scrubbing), Car 2 moves to the second section (soaping), and Car 3 starts in the first section (washing).
4. **Eventually, All Sections Are Busy**: After six cars have entered, every section will be busy with a different car. Now, a new car can enter the car wash every time a car finishes, keeping the whole process moving smoothly.

**Differentiate between instruction pipeline and an arithmetic pipeline.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Instruction Pipeline** | **Arithmetic Pipeline** |
| Focus | Aims to make the CPU execute instructions faster | Aims to make arithmetic calculations faster |
| Level of Operation | Works at the instruction level | Works at the operation level |
| Stages | Involves stages like fetching, decoding, executing, and writing back | Involves smaller stages for specific operations like addition and multiplication |
| Overlapping | Allows different instructions to be in various stages at the same time | Allows different arithmetic operations to be in various stages at the same time |
| Hazards | Stalls if instructions depend on each other | Stalls if one operation relies on another |
| Goal | Increases the performance of the system | Improve the performance of arithmetic operations |

What is cache memory? Explain the mapping process. Differentiate between direct mapping and associate mapping.

**Mapping => Cache Mapping**

Cache memory is a high-speed memory that acts as a buffer between the CPU and main memory (RAM). It stores copies of frequently accessed data from main memory, allowing the CPU to access it more quickly. Cache memory is typically smaller and faster than main memory, making it a valuable tool for improving system performance.

**Mapping Process in Cache Memory**

The **mapping process** refers to how the addresses from the main memory are translated into cache addresses. Since cache memory is smaller than main memory, so it cannot hold all the data. Mapping determines where data from main memory will be stored in the cache.

A diagram of a computer network

Description automatically generated

Figure: Process of Cache Mapping

**Types of Mapping**

* **Direct Mapping**
* **Associate Mapping**

**Direct Mapping**

1. **Mapping Method:** Each item from the main memory goes to one specific spot in the cache using a modulus operation.
2. **Mapping Formula:** Uses a formula to figure out the spot: **(Block number) mod (Number of cache spots)**.
3. **Example:** If there are 4 spots in the cache and the block number (item) is 6, then **Cache Line = 6 mod 4 = 2**.
4. **Tag Bits:** Needs fewer labels because the spot is always the same.
5. **Access Time:** Quicker because you know exactly where to look.
6. **Conflict Misses:** Higher chance of conflict misses, because different items (blocks) might need the same spot, causing them to replace each other.
7. **Advantages:** Simple implementation, fast access, and less expensive.
8. **Disadvantages:** Multiple items (blocks) may map to the same cache spot, causing replacement of data.

**Associative Mapping**

1. **Mapping Method:** An item can be stored in any available spot in the cache.
2. **Mapping Formula:** No formula is needed; the item can go anywhere there is space.
3. **Example:** If an item is brought into the cache, it can go to any empty spot.
4. **Tag Bits:** Needs more detailed labels since the item (blocks) can be anywhere, and all spots need to be checked.
5. **Access Time:** Slower because you have to check every spot to find the item.
6. **Conflict Misses:** Less likely to happen since items (blocks) can go to any spot, reducing the chance of replacing each other.
7. **Advantages:** Reduced conflict misses.
8. **Disadvantages:** Requires more time to search since the item (blocks) can be anywhere, and all spots need to be checked.

**Direct Mapping vs Associative Mapping**

A screenshot of a computer

Description automatically generated

**Explain common bus system for basic computer.**

A common bus system is a fundamental concept in computer architecture that allows various components to communicate and share data.

A bus is a collection of wires used to transmit data, addresses, and control signals from one system to another.

Types of Buses / Key Components of a Common Bus System

1. **Data Bus:** Carries data to and from memory and I/O devices.
2. **Address Bus:** Specifies the memory location or I/O device address.
3. **Control Bus:** Transmits control signals to coordinate the data transfer.

A diagram of a computer program

Description automatically generated

A white board with writing on it

Description automatically generated

Differentiate between restoring division and non-restoring division

A table with text on it

Description automatically generated