4. Draw a single datapath representation for R-type instruction

Introduction

A **datapath** is the hardware structure within a CPU that performs operations like arithmetic, logic, and data transfer. In modern processors, datapath elements are connected together by buses and controlled by the **control unit** to execute instructions.

An **R-type instruction** is a format used in RISC architectures (like MIPS) where the operation is performed between registers — for example:

sql Copy code

```
ADD R1, R2, R3  →  R1 = R2 + R3
```

This instruction performs addition of contents of registers R2 and R3, and stores the result in R1.

R-Type Instruction Format

Field	Bits	Description
opcode	6	Operation code (always 000000 for R-type)
rs	5	First source register
rt	5	econd source register

opcode	6	Operation code (always 000000 for R-type)
rs	5	First source register
rt	5	Second source register
rd	5	Destination register
shamt	5	Shift amount (used in shift operations)
funct	6	Function code (specifies exact ALU operation)

Datapath Components Used

1. **Program Counter (PC):** Holds address of the current instruction.
2. **Instruction Memory:** Fetches the instruction using PC.
3. **Register File:** Contains 32 general-purpose registers. Two registers (rs, rt) are read; rd is written.
4. **ALU (Arithmetic Logic Unit):** Performs the operation specified by the funct field (ADD, SUB, AND, OR, etc.).
5. **Control Unit:** Decodes the instruction and generates control signals (like ALUOp, RegDst, RegWrite).
6. **Multiplexers (MUX):** Select between inputs such as immediate value or register data.
7. **Sign Extend:** Extends 16-bit immediate values to 32-bit (not used in R-type).

Step-by-Step Operation

1. **Instruction Fetch:**
   The instruction is fetched from instruction memory using the address from the PC.
   `Instruction ← Memory[PC]`
2. **Instruction Decode:**
   The control unit decodes the opcode and identifies it as R-type.
   Register file reads `rs` and `rt` registers.
3. **Execution:**
   ALU performs operation based on funct field, e.g., ADD, SUB, AND, OR.
   `Result = Register[rs] op Register[rt]`
4. **Write Back:**
   The ALU result is written into destination register `rd`.

Control Signals Used

- **RegDst** = 1 → Destination is `rd`
- **ALUSrc** = 0 → Second ALU input from register
- **RegWrite** = 1 → Write result to register file
- **MemRead / MemWrite** = 0 → No memory operation
- **ALUOp** = 10 → Determine operation using funct field

Datapath Diagram Description

mathematica Copy code

```
PC → Instruction Memory → Instruction [31:0]
        ↓
   ┌──────────────┐
   | Control Unit |
   └──────────────┘
      ↓       ↓
Register File → ALU → Result → Write Back
   ↑ rs, rt          ↑ funct field (control)
```

This single-cycle datapath executes the R-type instruction in one clock cycle — ideal for RISC design.

Conclusion

R-type instructions use only registers for computation. The datapath performs fetching, decoding, execution, and writing back using components like ALU and register file under control of the control unit.

5. With a neat diagram, explain the operation of DMA controllers in a computer system

and writing back using components like ALU and register file under control of the control unit.

---

5. With a neat diagram, explain the operation of DMA controllers in a computer system

Introduction

Direct Memory Access (DMA) is a technique that allows peripherals to transfer data directly to/from the main memory without continuous CPU involvement.
It increases system performance by freeing the CPU during data transfer.

Example: Transferring a block of data from disk to memory.

Need for DMA

Without DMA, every I/O operation requires CPU to:

1. Read data from device into a register.
2. Write data from register into memory.

This causes heavy CPU load. DMA removes this by allowing direct device–memory data transfers.

Components of DMA Controller

1. **DMA Controller:** Manages all DMA transfers.

New chat
Search chats
Library
Projects

Chats
DFCA assignment answers
Gaming vs professional laptop
Create Python class
CSS style changes
Fix JavaScript syntax error
Fix git configuration error
HTML frames code

This causes heavy CPU load. DMA removes this by allowing direct device–memory data transfers.

Components of DMA Controller

1. **DMA Controller:** Manages all DMA transfers.
2. **Address Register:** Stores the memory address for transfer.
3. **Count Register:** Stores number of bytes/words to transfer.
4. **Control Logic:** Determines direction (read/write) and issues control signals.
5. **Data Bus Interface:** Connects to system bus for data transfer.
6. **Interrupt Line:** Signals CPU when transfer is completed.

DMA Transfer Process

1. **CPU Initialization:**
   CPU programs DMA controller by providing:
   - Starting memory address
   - Transfer count
   - I/O device address
   - Direction (read/write)
2. **DMA Request:**
   The I/O device sends a **DMA request (DREQ)** signal to DMA controller.
3. **Bus Request and Grant:**
   DMA controller requests control of the system bus using **Bus Request (BR)**.

- I/O device address
- Direction (read/write)

2. **DMA Request:**
   The I/O device sends a **DMA request (DREQ)** signal to DMA controller.
3. **Bus Request and Grant:**
   DMA controller requests control of the system bus using **Bus Request (BR)**.
   CPU acknowledges with **Bus Grant (BG)**.
4. **Data Transfer:**
   DMA controller transfers data directly between I/O and memory using address and data buses.
5. **Transfer Completion:**
   After the transfer, DMA controller sends an **Interrupt** to CPU indicating completion.

Modes of DMA Operation

1. **Burst Mode:**
   Entire block of data transferred in one go. CPU remains idle.
2. **Cycle Stealing Mode:**
   DMA transfers one word per cycle, temporarily "stealing" bus cycles from CPU.
3. **Transparent Mode:**
   DMA transfers occur only when CPU is not using the system bus.

DMA Diagram Description

New chat
Search chats
Library
Projects

Chats
DFCA assignment answers
Gaming vs professional laptop
Create Python class
CSS style changes
Fix JavaScript syntax error
Fix git configuration error
HTML frames code

DMA Diagram Description

mathematica

Copy code

```
I/O Device ↔ DMA Controller ↔ Memory
                ↑       ↑
          Bus Req   Bus Grant
                ↓
               CPU
```

Advantages

- High-speed data transfer.
- Reduces CPU overhead.
- Improves overall system efficiency.

Disadvantages

- Hardware complexity.
- CPU cannot access memory during DMA transfer.

Conclusion

DMA controllers enhance system performance by handling bulk data transfers between I/O and memory efficiently, allowing the CPU to perform other tasks s..ltaneously.

efficiently, allowing the CPU to perform other tasks simultaneously.

---

6. How should two or more simultaneous interrupt requests be handled? Explain with figure.

Introduction

In a computer system, multiple devices (keyboard, disk, printer, etc.) can request CPU attention simultaneously through **interrupts.**
When this occurs, the CPU must decide **which interrupt to handle first** — this is called **Interrupt Priority Handling.**

Types of Interrupts

1. **Maskable Interrupt:** Can be enabled or disabled by software.
2. **Non-Maskable Interrupt:** Always recognized by CPU; cannot be ignored.

Problem of Simultaneous Interrupts

If two devices raise interrupt signals at the same time, CPU must use a mechanism to:

- Identify which device requested the interrupt.
- Determine the priority among them.
- Service them in correct order.

New chat
Search chats
Library
Projects
Chats
DFCA assignment answers
Gaming vs professional laptop
Create Python class
CSS style changes
Fix JavaScript syntax error
Fix git configuration error
HTML frames code

- Service them in correct order.

Methods to Handle Simultaneous Interrupts

1. Software Polling

- CPU checks each device's interrupt flag sequentially.
- The first device found with an active flag is serviced.
- Simple but **slow**, as each device must be checked manually.

Example:

scss Copy code

```
if (Device1 interrupt) service Device1
else if (Device2 interrupt) service Device2
else if (Device3 interrupt) service Device3
```

---

2. Hardware Priority (Vectored Interrupts)

- Each device has its own interrupt line and a priority level.
- When an interrupt occurs, the device sends an **interrupt vector** (address of its ISR) directly to CPU.
- Highest priority interrupt is serviced first.

Advantages:

- Faster than polling.
- Automatic vector identification

- Faster than polling.
- Automatic vector identification.

---

3. Daisy Chaining (Hardware Priority)

- Devices connected serially in a chain.
- Highest priority device placed closest to CPU.
- CPU sends **Interrupt Acknowledge (INTA)** signal through the chain.
- First device with an interrupt captures the signal and blocks others below it.

Diagram Description:

scss

Copy code

```
CPU ← Device 1 ← Device 2 ← Device 3
(Highest)        (Medium)    (Lowest)
```

Operation:

1. All devices send interrupt requests.
2. CPU sends acknowledgment.
3. The highest-priority device captures the acknowledgment and blocks it from passing further.
4. CPU executes that device's ISR.
5. After completion, CPU checks again for pending interrupts.

---

4. Priority Encoder Method

4. Priority Encoder Method

A **priority encoder** circuit receives multiple interrupt requests and encodes the highest-priority one.
Example:
If input D3 has higher priority than D2, D1, D0 → output corresponds to D3 even if all are active.

Conclusion

When two or more interrupt requests occur simultaneously, they are handled using **priority mechanisms** such as:

- Software polling (simple)
- Daisy chaining (hardware priority)
- Vectored interrupts (fastest)

These methods ensure that critical devices receive CPU attention first, maintaining system efficiency and reliability.

Would you like me to generate a **Word or PDF file (ready to print)** version of this assignment with formatted titles, tables, and labeled diagrams?
I can create that instantly for submission.