CS220 Assignment-8 Documentation/Report

Rahul Meena(230832), Rohit Vinod Atkurkar(230872)

PDS1: Registers and Usage Protocol

The IITK-Mini-MIPS processor implements an architecture based on MIPS design principles. As specified in the MIPS architecture, our implementation includes 32 general-purpose registers (r0-r31) and 32 floating-point registers.

General Purpose Registers:

- Register r0: Hardwired to constant zero (0x00000000)
- Registers r1-r31: Fully programmable 32-bit registers
- Special Purpose Registers:
    - r29: Stack pointer
    - r30: Frame pointer
    - r31: Return address register

Register Usage Convention:

Following MIPS calling conventions:

- r0: Zero register
- r1: Reserved for assembler use
- r2-r3: Function return values
- r4-r7: Function arguments
- r8-r15: Temporary registers
- r16-r23: Saved registers
- r24-r25: Temporary registers
- r26-r27: Reserved for OS kernel
- r28: Global pointer
- r29: Stack pointer
- r30: Frame pointer
- r31: Return address

Floating Point Registers:

- 32 floating-point registers (f0-f31) supporting IEEE 754 single-precision format
- Movement between general-purpose and floating-point register files is restricted to specific instructions (mfc1, mtc1)

PDS2: Memory Size and Organization

Instruction Memory:

- Size: 4KB (1024 words of 32 bits each)
- Organization: Word-addressable (addresses must be multiples of 4)
- Addressing: Uses the lower 12 bits of the address (bits [11:2]), ignoring bits [1:0] to ensure word alignment
- Access: Read-only

Data Memory:

- Size: 4KB (1024 words of 32 bits each)
- Organization: Word-addressable
- Addressing: Uses the lower 12 bits of the address (bits [11:2]) for word operations
- Access: Both read and write operations supported

Memory Access Protocol:

- Load/store architecture following MIPS conventions
- Memory transactions occur with explicit load and store instructions
- Word-aligned operations for maximum efficiency

PDS3: Instruction Layouts and Encoding

The IITK-Mini-MIPS processor implements three standard MIPS instruction formats: R-type, I-type, and J-type, each with a fixed 32-bit length.

R-Type Instruction Format:

Opcode:[31:26] rs:[25:21] rt:[20:16] rd:[15:11] shamt:[10:6] funct:[5:0]

- Used for: Register-to-register operations (add, sub, mul, etc.)
- Opcode: Always 000000 (0x00) for standard R-type instructions
- rs, rt: Source register numbers (5 bits each)
- rd: Destination register number (5 bits)
- shamt: Shift amount (5 bits), used for shift instructions
- funct: Function code that specifies the exact operation (6 bits)

I-Type Instruction Format:

Opcode:[31:26] rs: [25:21] rt:[20:16] immediate:[15:0]

- Used for: Immediate operations, memory access, and branches
- Opcode: Varies based on instruction type
- rs: First source register number (5 bits)
- rt: Destination or second source register (5 bits)
- immediate: 16-bit immediate value or memory offset

J-Type Instruction Format:

Opcode:[31:26] address:[25:0]

- Used for: Jump instructions
- Opcode: 000010 (0x02) for jump, 000011 (0x03) for jump-and-link
- address: 26-bit jump target address (shifted left 2 bits and combined with PC[31:28])

Floating-Point Instruction Format:

Opcode:[31:26] fmt:[25:21] ft:[20:16] fs:[15:11] fd: [10:6] funct: [5:0]

- Opcode: 010001 (0x11) for floating-point operations
- fmt: Format (0 for single-precision)
- ft, fs: Source register numbers
- fd: Destination register number
- funct: Function code for the operation

PDS4: Instruction Fetch Implementation

The instruction fetch phase retrieves the next instruction from instruction memory based on the current Program Counter (PC) value.

The PC module contains a 32-bit register that holds the address of the next instruction to be executed. On reset, the PC is initialized to 0x00000000. During normal operation, the PC is updated to the next sequential address (PC+4) unless modified by a branch or jump instruction.

The instruction memory module is a read-only memory that stores the program instructions. When addressed by the PC, it outputs the 32-bit instruction at that address. To ensure proper word alignment, the two least significant bits of the address are ignored.

The next PC value is calculated based on the current instruction:

- For sequential execution: PC + 4
- For branch instructions: PC + 4 + (branch_offset << 2) if the branch condition is true
- For jump instructions: {PC[31:28], target address, 00}
- PC provides the address to instruction memory
- Instruction memory returns the 32-bit instruction at that address
- PC is incremented by 4
- Branch and jump conditions are evaluated to determine the next PC value

PDS5: Instruction Decode Module

The instruction decode phase determines what operation to perform based on the opcode and function fields of the fetched instruction, generating control signals for subsequent pipeline stages.

The control unit decodes the instruction opcode and function fields to generate control signals that direct the datapath's operation. These signals include:

- Register destination selection
- ALU operation selection

- Memory read/write enables
- Register write enable
- Branch and jump control
- Floating-point operation control

During the decode stage, register operands are read from the register file based on the rs and rt fields of the instruction. For R-type instructions, the destination register is specified by the rd field, while for I-type instructions, the rt field specifies the destination.

For I-type instructions, the 16-bit immediate value is extended to 32 bits. Depending on the instruction, this may be sign-extended (for arithmetic operations) or zero-extended (for logical operations).

PDS6: Arithmetic Logic Unit (ALU) Implementation

The ALU performs various operations based on the control signals provided by the instruction decoder. It supports:

- Arithmetic operations: addition, subtraction, multiplication
- Logical operations: AND, OR, XOR, NOT
- Shift operations: logical and arithmetic shifts in both directions
- Comparison operations: set-less-than for signed and unsigned comparisons

A separate floating-point ALU handles IEEE 754 single-precision floating-point operations, including:

- Addition and subtraction
- Multiplication
- Comparison operations
- Movement operations between floating-point registers
- Adder circuit shared between addition, subtraction, and address calculations
- Barrel shifter for all shift operations
- Logic unit for AND, OR, XOR, and NOT operations
- Multiplier for mul, madd, and maddu instructions
- Comparator for set-less-than and branch condition evaluation

ALU Integration:

- ALU receives operands from register file or immediate extension
- Operation selection driven by control signals from instruction decoder
- Results written back to register file or used for memory addressing
- Flag outputs used for branch condition evaluation

PDS7: Branching Operations Implementation

The branching operations include all conditional and unconditional changes to the program flow, including branch instructions, jumps, and subroutine calls. The branch control module evaluates branch conditions by comparing register values according to the branch type. It generates a signal that determines whether the program counter should be updated with the branch target address. The branch address is calculated by adding the sign-extended, left-shifted branch offset to the PC+4 value. This follows the MIPS convention where branch displacements are relative to the instruction following the branch.

Jump addresses are calculated by concatenating the upper 4 bits of PC+4 with the 26-bit jump target from the instruction, shifted left by 2 bits. This forms a 32-bit absolute address.

Branching Operation Types

- Conditional branches: beq, bne, bgt, bgte, blt, ble
- Unconditional jumps: j, jal
- Register-based jumps: jr (jump register)
- Subroutine calls: jal (jump and link)
- Subroutine returns: jr $ra (jump to return address)

PDS8: Control Path Design and System Integration

The control path coordinates the operation of all processor components and ensures proper instruction execution through a finite state machine.

Single-Cycle Control Path

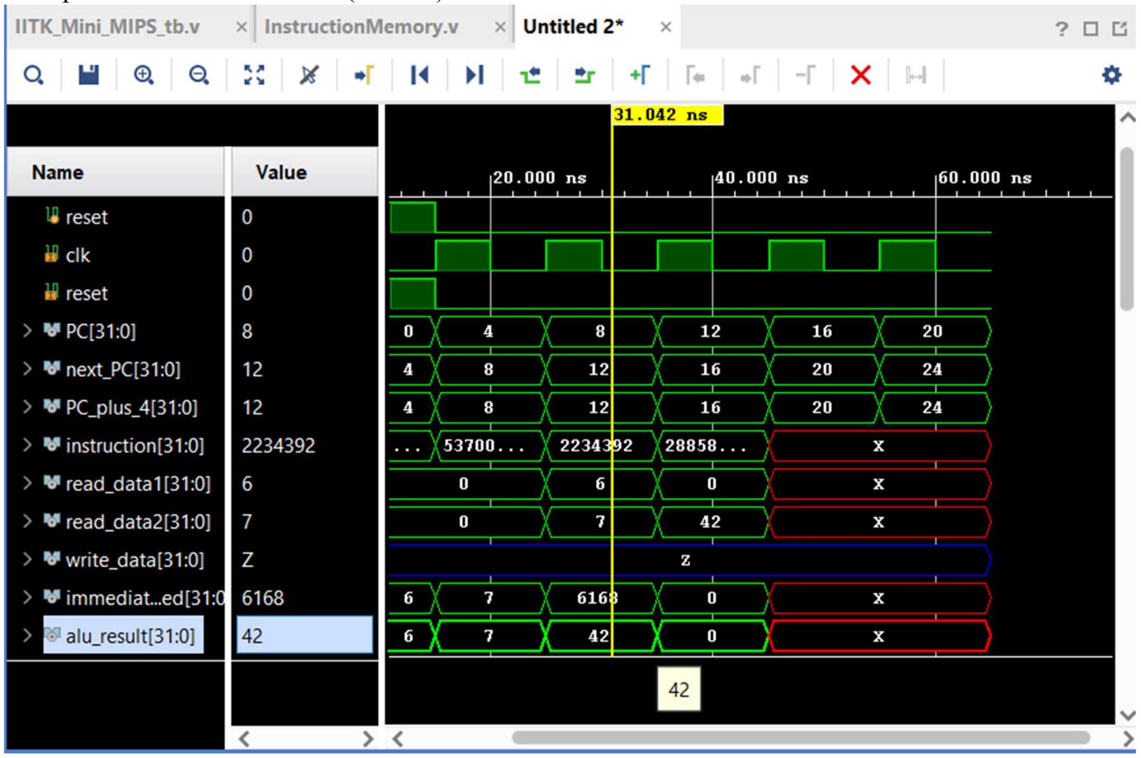
- All instructions execute in a single clock cycle
- Control signals generated based on instruction opcode and function fields
- Direct connection between control signals and datapath components

Control Signal Generation

1. Fetch Stage: PC update, instruction memory read
2. Decode Stage: Register file read, control signal generation
3. Execute Stage: ALU operation, branch evaluation
4. Memory Stage: Data memory read/write
5. Write-Back Stage: Register file write

PDS9 & 10: Testing Machine Code

MIPS instructions were converted to machine code using the provided Python assembler and were loaded into instruction memory in the test bench. The simulation results obtained for the storage and multiplication of two numbers (6 and 7) is as follows:



Name	Value
reset	0
clk	0
reset	0
PC[31:0]	8
next_PC[31:0]	12
PC_plus_4[31:0]	12
instruction[31:0]	2234392
read_data1[31:0]	6
read_data2[31:0]	7
write_data[31:0]	Z
immediat...ed[31:0	6168
alu_result[31:0]	42

```
PC[31:0]           0 | 4 | 8 | 12 | 16 | 20
next_PC[31:0]      4 | 8 | 12 | 16 | 20 | 24
PC_plus_4[31:0]    4 | 8 | 12 | 16 | 20 | 24
instruction[31:0]  ... | 53700... | 2234392 | 28858... | X
read_data1[31:0]   0 | 6 | 0 | X
read_data2[31:0]   0 | 7 | 42 | X
write_data[31:0]   Z
immediat...ed[31:0 6 | 7 | 6168 | 0 | X
alu_result[31:0]   6 | 7 | 42 | 0 | X
```