Topics

- Construct the “Datapath” & “Control unit” for MIPS instruction set
- Design implementation for core MIPS
  - Memory reference
    - Load ($lw$) and store ($sw$) instructions
  - Arithmetic-logical instructions
    - $add$, $sub$, $and$, $or$, & $slt$
  - Branch instructions
    - $beq$, $j$
Introduction

- The 5 classic components of a computer

- The following chapters discuss those components
- Chapter 5 discusses the processor
  - Part-a: Datapath
  - Part-b: Control
Introduction

- Key design principles:
  - Make common case faster
  - Simplicity favors regularity

- Most concepts used to implement MIPS subset are the same basic ideas used to implement many other computers, from high-performance servers to general-purpose microprocessors to embedded processors
The Processor

- Consists of:
  - Datapath
  - Control

- Responsible for:
  - Use the program counter (PC) to supply instruction address
  - Get the instruction from memory
  - Read registers
  - Use the instruction to decide exactly what to do
The Processor

- Actions needed to complete the different instructions depend on the instruction class
  - Memory reference:
    - Access memory to write / read data
  - Arithmetic:
    - Write data from ALU back into registers
  - Branch:
    - Change next instruction address
For all instruction types the first 2 steps are identical

1. Fetch memory contents to which PC points
   - Send PC to memory containing the instruction
2. Read (1 or 2) registers whose number(s) is/are given in the instruction fields (*rs*, *rt*, or *rd*)
The Processor

- Even across different instruction classes there are some similarities
- All instructions (except jump) use the ALU after reading the registers
  - Memory-reference instructions
    - For address calculation
  - Arithmetic instructions
    - For operation execution
  - Control flow (branch) instructions
    - For comparison
The Processor

- After using the ALU, the actions required to complete each instruction class can differ
  - Memory-reference instructions
    - Access the memory to read or write data
  - Arithmetic-logical instructions
    - Write data from ALU back into registers
  - Branch instructions
    - Change the next instruction address based on the comparison; otherwise the PC is incremented by 4 to get address of next instruction
The Processor

- PC supplies instructions to “Instruction Memory”
- Registers hold operands
- ALU computes results
- Results stored in data memory or values fetched from data memory into registers
- Data going to a particular unit might come from different sources
  - A MUX is required
The Processor

- Example: PC
  - Result comes from one of 2 adders
- Example: Register file
  - Data written into a register can come from ALU or Data memory
The Processor

- Several units must be controlled depending on the type of instruction
  - Control Lines are needed
- Example:
  - Data memory on load and write on store
  - Register file must be written on load & arithmetic-logical instructions]
The Processor

- Modified Processor
  - 3 MUX’s and Control lines are added
The Datapath

- Datapath contains the component of the CPU that performs arithmetic operations

Main components
- Instruction & data memories
- Register files
- Program Counter (PC)
- ALU
- Adders
- Others . . .
The Control Unit

- Has the instruction as input
- A decoder unit can be used to find how to set the control lines
- Selects the operations
- Controls data flow
- Synchronized by the clock
- Consists mainly of combinational circuits
  - Gates
  - MUX’s
Performance Perspective

- Two choices for processor’s clock
  - Single clock cycle:
    - Every instruction begins execution on one clock edge and completes on the next clock edge
    - Advantage: One clock cycle per instruction
    - Disadvantage: Long cycle time makes it slower, since some instructions might need shorter time
  - Multiple clock cycle:
    - Use multiple clock cycles for each instruction
    - Advantage: Faster, each instruction will take the number of cycles needed
    - Disadvantage: Multiple clock cycles per instruction

- We will start with a single clock cycle design
Datapath Design

What will we include in the design?
- Subset of core MIPS instructions
  - Memory reference (R-Format)
    - Load/Store instructions \textit{lw, sw}
    - Examples:
      \begin{align*}
      \text{lw} \ rt, \ rs, \ \text{imm16} \\
      \text{sw} \ rt, \ rs, \ \text{imm16}
      \end{align*}
  - Arithmetic/logical (R-Format)
    - \textit{add, sub, and, or, slt, ori}
    - Examples:
      \begin{align*}
      \text{add} \ rd, \ rs, \ rt \\
      \text{sub} \ rd, \ rs, \ rt
      \end{align*}
  - Branch (I- & J-Formats)
    - \textit{beq, j}
    - Examples:
      \begin{align*}
      \text{beq \ rs, \ rt, \ imm16}
      \end{align*}
Datapath Design

- What will not be included?
  - Multiplication & division instructions
  - Floating-point instructions

- Design guidelines:
  - Make common case faster
  - Simplicity favors regularity
Datapath Design

- Functional units required:
  - Combinational:
    - Elements that operate on data values
    - Output depends on current input
    - ALU
    - Other components
  - Sequential:
    - Elements that contain state
    - Instruction memory
    - Data memory
    - Registers
Review: State Element

- Element has internal storage (Memory)
  - D-Flip-flop
  - Memories
  - Registers

- Element has at least
  - Two inputs
    - Data value to be written into the element
    - Clock to determine when data is written
  - One output
    - Data value written in earlier clock cycle
    - Can be read at any time during the clock period

- Output depends on:
  - Input
  - Internal state
  - Can be read at any time
Review: Clocking Methodology

- Defines when signals can be read / written
- Prevents unpredictability in reading or writing values
- Un-clocked:
  - Used in asynchronous logic
  - Values updated as soon as input arrives
- Clocked:
  - Used in synchronous logic
  - Values updated with clock input, anytime during which the clock is asserted
- Edge-triggered:
  - Values updated only on clock edge (rising or falling edge, depending on the logic used)
- We will assume an edge-triggered methodology
Review: Edge-triggered Methodology

- **Input:**
  - Values written in a previous clock cycle

- **Output:**
  - Values to be used in the following clock cycle

- Prevents reading the signal in the same time it is written

- More than one action can take place in the same clock cycle

- Logically true clock could mean electrically low

![Clock period diagram with Rising and Falling edges]
Review: Timing Methodology

- **Clock cycle (Tick/Period):**
  - Time for one clock period, usually of the processor, which runs at a constant rate

- **Access time:**
  - Time between the initiation of a read request and when the desired word arrives

- **Cycle time:**
  - Minimum time between requests to memory
  - Should be greater than access time to keep address line stable between accesses
Review: Timing Methodology

- **Typical execution cycle:**
  - Read contents of some state elements,
  - Send values through some combinational logic
  - Write results to one or more state elements
  - Clock period should cover all these activities

- All signals must propagate from state element1 to state element2 in the time of one clock cycle

- If the state element is not updated on every clock, an explicit write control signal is required, in which case the state element is changed only when the control signal is asserted and the clock edge occurs
Review: Timing Methodology

- Edge triggered methodology allows a state element to be read and written in the same clock cycle.
- The clock must be long enough to allow the stability of input value before the active edge occurs.

![Diagram of state element and combinational logic](image-url)
Processor Design Steps

1. Analyze instruction set & determine datapath requirements
2. Select set of datapath components & establish clocking methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that affects the register transfer
5. Assemble the control logic
Building the Datapath

What are the major components required to execute each class of MIPS instructions?

- A memory unit to store the instructions
- PC to store the address of the current instruction
- Adder to increment PC to address of next instruction
Building the Datapath

How are these components connected?
Building the Datapath

What are the major components required to execute R-Format instructions?
- add, sub, and, or, slt
- Three register operands
  - Two read ports to read two registers from register file
  - One write port to write into one register
- Elements needed
  - Register file
  - ALU
- Control
  - Register write
  - ALU operation
- Data lines 1re 32-bit
- Register lines 5-bit
- ALUOp 4-bits
Building the Datapath

- The branch address calculation is the address of the following instruction (PC + 4)
- Offset field is shifted 2-bits so that it is word offset
  - Increases range by factor of 4
  - Need to be shifted back when calculating the actual address in bytes

![Diagram of the Datapath](image)
Building the Datapath

- For branch instructions, two operations are needed
  - Compare register contents using ALU
  - ALU Zero signal returns the result of comparison
  - Compute branch target address
    - A sign-extend unit is required
    - If branch is taken
      - Branch target address becomes the new PC contents
    - If branch is not taken
      - PC+4 is the new value for PC
Building the Datapath

- For jump instruction
  - Replace the lower 28 bits of PC with the 26 bits of the instruction shifted 2 bits to the left
- Delayed branch
  - The instruction immediately following the branch is always executed, independent of whether the branch condition is true or false
  - More efficient when dealing with pipelining
Review: Register File

- Contains the processor’s 32 registers
- Can be read / written by supplying a register number to be accessed
- Implemented with a decoder for each read/write port
- The array of registers is implemented using D-FFs
- Reading doesn’t change the state (Non-destructive)
- Writing changes the state (Destructive)
Review: Register File

- A set of 32 registers
  - 5-inputs
    - 2 read-ports to supply source register numbers
    - 1 write-port to supply destination register number
    - 1 Data bus
    - 1 Register write enable control signal
  - 2-outputs
    - Data from register 1
    - Data from register 2
  - 1-Control signal
    - RegWrite
    - No need to have a control signal for reading, data is always available on the data bus.
Review: Reading from Register File

- Need to submit:
  - Register #
- For N-registers we need:
  - \( n \times 1 \) MUX
Review: Writing to Register File

- Need to submit:
  - Register #
  - Data
  - Write control signal
- To choose a register, use
  - Decoder
- To determine when to write, use the clock
- Note:
  - C means control signal
  - D means data lines
Creating a Single Cycle Datapath

- Combine the previous datapath components into one
- Execute all instructions in one cycle
- No datapath resource can be more than once per cycle/instruction
- Many elements need duplicates
  - One memory for instruction & one for data
- To share datapath element between two different instruction classes, we allow multiple connections to input of an element, using multiplexors & control signals
Creating a Single Cycle Datapath

- **Example:**
  - Combine R-Type and memory instructions
    - ALU input can have input from a register or from the sign-extend unit
    - Value stored in destination register comes from ALU or from memory
Creating a Single Cycle Datapath

- Example:
  - Combine R-Type, memory, and branch instructions
    - Extra adder for computing branch target address
    - Extra multiplexor to select the branch target
5 steps to design a processor

1. Analyze instruction set => datapath requirements
2. Select set of datapath components & establish clock methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic
Summary

- MIPS makes it easier
  - Instructions are all the same size
  - Source registers are always in same place in the instruction
  - Immediate operands are the same size & in the same location
  - Operations are always on either register contents or immediate operands

- Single cycle datapath
  => CPI=1