Lecture Topics

- Today: Single-Cycle Processors
  (P&H 4.1-4.4)

- Next: continued

Announcements

- Milestone #2 (due 2/2)
- Milestone #3 (due 2/9)
- Milestone #4 (due 2/23)
ISA Implementations

- The ISA for a particular processor defines the features of that machine: instructions, registers, etc.

- The fetch-execute cycle governs the processor:
  - Fetch current instruction
  - Decode and execute it
  - Update PC to point to next instruction

ISA Implementations

- Three different strategies:
  - single-cycle implementation
  - multi-cycle implementation
  - pipelined implementation

- Many different implementations possible within a particular framework
Timing Considerations

Operations sequenced by clock

Example:

\[
\text{clock period} = 250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}
\]
\[
\text{clock frequency} = 4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}
\]

Timing Considerations

Combinational logic transforms data within one clock cycle (between clock edges):

- Input from state element, output to state element
- Longest delay determines clock period
Single-Cycle Implementation

- Each instruction completes in one clock cycle (fetch and execute in same cycle)
- Length of clock period determined by worst case (instruction category with longest path through circuits)

Multi-Cycle Implementation

- Each instruction broken down into a series of "phases", where each phase takes one clock cycle
- Each instruction in ISA completes in a specific number of clock cycles:
  - ADD – 3 cycles
  - LOAD – 5 cycles
  - STORE – 4 cycles
Pipelined Implementation

- Each instruction broken down into a series of "phases", where each phase takes one clock cycle
- Different phases of consecutive instructions overlapped
- All instructions within the same framework (ex: five stage pipeline)

MIPS ISA

- Machine language instructions (32 bits):
  - R-format
  - I-format
  - J-format
- Registers (32 bits):
  - 32 general-purpose (5-bit reg. numbers)
  - PC, IR
MIPS Fetch-Execute Cycle

Fetch Phase:

MEM[ PC ] ==> IR
PC + 4 ==> PC

Execute Phase:

decode IR
take appropriate action

Fetch Phase

- PC contains address of current instruction
- PC serves as input to Instruction Memory (Instruction Cache)
- Instruction Memory returns instruction
- Address of next sequential instruction is at PC+4
Review: Registers

Register used to hold data in a circuit

- Uses a clock signal to determine when to update the stored value
- Edge-triggered: update when Clk changes from 0 to 1

Review: Registers

Register with write control

- Only updates on clock edge when write control input is 1
- Allows selective updating
Execute Phase

- Decode instruction
  - extract operation code info
  - extract register numbers, immediate value

- Execute instruction
  - generate control signals
  - use circuits to take the correct actions for this particular instruction

MIPS Instruction Formats
Can’t just join wires together -- use multiplexers
Patterson and Hennessy describe a simplified version of the MIPS in chapter 4.

- Only a small subset of instructions:
  - ADD, SUB, AND, OR, SLT
  - LW, SW
  - BEQ, J

We need to consider a richer set of instructions.
MIPS Instruction Categories

- Data manipulation
  - R-format (both operands in registers)
  - I-format (one immediate value)

- Data movement

- Control transfer
  - R-format (jump register)
  - I-format (branch)
  - J-format (jump)

Data Manipulation (R-format)

- Operands in registers: R[rs] and R[rt]
- Result written to R[rd]
- Execution steps:
  \[
  \text{ALU( R[rs], R[rt], op ) } \Rightarrow \text{ R[rd]}
  \]
- ALU operation derived from bits in IR
Data Manipulation (I-format)

- Operands in R[rs] and IR[15:0]
- Result written to R[rt]
- Execution steps:
  
  extend immediate value to 32 bits
  
  ALU( R[rs], imm32, op ) ==> R[rt]

- ALU operation derived from bits in IR
Data Manipulation (I-format)

The 16-bit immediate value must be extended to 32 bits

- sign extended
  ADDI, ADDIU, SLTI, SLTIU
- zero extended
  ANDI, ORI, XORI
- moved left by 16 positions
  LUI
Data Manipulation (I-format)

- Sign extension – 16 copies of bit 15 of IR
- Examples:

```
addi $6, $5, 4       20A60004
                   00000004

addi $7, $6, -1      20C7FFFFFF
                   FFFFFFFF
```

Data Manipulation (I-format)

- Zero extension – 16 copies of zero
- Examples:

```
ori  $8, $0, 0xE000   3408E000
     0000E000

ori  $9, $0, 100      34090064
     00000064
```
Data Manipulation (I-format)

- LUI – move immediate bits 16 positions to the left, fill with zeroes on the right

- Examples:
  \[
  \begin{align*}
  \text{lui} & \quad \$4, \ 0xFF & \quad 3C0400FF \\
  & \quad \quad \quad \quad 00FF0000 \\
  \text{lui} & \quad \$5, \ 0x1234 & \quad 3C051234 \\
  & \quad \quad \quad \quad 12340000
  \end{align*}
  \]

Data Manipulation (all formats)

- Operand A is always R[rs]

- Operand B is one of four values:
  - R[rt]
  - IR[15:0], sign extended
  - IR[15:0], zero extended
  - IR[15:0], left shifted

- Result is either R[rd] or R[rt]
Operand B controlled by 4-to- MUX

- no need for "extend" circuit
- simply connect IR bits to MUX ports

Example: sign extension

IR: 00110100000100011110000000000000

MUX: 11111111111111111111000000000000
Data Movement (I-format)

- Operands in R[rs] and IR[15:0]
- Load: value copied into R[rt]
- Store: value copied from R[rt]
- Execution steps:
  - sign extend immediate value to 32 bits
  - ALU( R[rs], imm32, add ) ==> address
  - address used to access Data Memory

Data Movement (I-format)

- Load execution steps:
  - sign extend immediate value to 32 bits
  - ALU( R[rs], imm32, add ) ==> address
  - MEM[address] ==> R[rt]

- Store execution steps:
  - sign extend immediate value to 32 bits
  - ALU( R[rs], imm32, add ) ==> address
  - R[rt] ==> MEM[address]
Data Manipulation

- Data Memory (data cache) signals:
  - address (32 bits)
  - read enable (1 bit)
  - write enable (1 bit)
  - data input (8, 16 or 32 bits)
  - data output (8, 16 or 32 bits)

- Data input and output must support various sizes (byte, halfword, word)

Combined Datapath

- The data manipulation instructions and data movement instructions must be placed in the same framework

- Both use the register file and the ALU

- MUX used to select value to written into destination register:
  - data manipulation: output of ALU
  - Load instruction: output of Data Memory
P&H R-Type/Load/Store Datapath