CS 61C: Great Ideas in Computer Architecture (Machine Structures)

*Single-Cycle Processor Design*

Instructors:
Randy H. Katz
David A. Patterson

http://inst.eecs.Berkeley.edu/~cs61c/fa10

---

**Agenda**

- MIPS-lite Datapath
- Administrivia
- Technology Break
- CPU Timing
Agenda

- MIPS-lite Datapath
- Administrivia
- Technology Break
- CPU Timing

The MIPS-lite Subset

- **ADDU and SUBU**
  - addu rd, rs, rt
  - subu rd, rs, rt

- **OR Immediate:**
  - ori rt, rs, imm16

- **LOAD and STORE Word**
  - lw rt, rs, imm16
  - sw rt, rs, imm16

- **BRANCH:**
  - beq rs, rt, imm16
Register Transfer Language (RTL)

- RTL gives the **meaning** of the instructions
  
  \[
  \{\text{op} , \text{rs} , \text{rt} , \text{rd} , \text{shamt} , \text{funct}\} \leftarrow \text{MEM}[\text{PC}]
  \]
  
  \[
  \{\text{op} , \text{rs} , \text{rt} , \text{Imm16}\} \leftarrow \text{MEM}[\text{PC}]
  \]

- All start by fetching the instruction

  \begin{align*}
  \text{Inst} & \quad \text{Register Transfers} \\
  \text{ADDU} & \quad \text{R[rd]} \leftarrow \text{R[rs]} + \text{R[rt]} ; \text{PC} \leftarrow \text{PC} + 4 \\
  \text{SUBU} & \quad \text{R[rd]} \leftarrow \text{R[rs]} - \text{R[rt]} ; \text{PC} \leftarrow \text{PC} + 4 \\
  \text{ORI} & \quad \text{R[rt]} \leftarrow \text{R[rs]} | \text{zero_ext(Imm16)} ; \text{PC} \leftarrow \text{PC} + 4 \\
  \text{LOAD} & \quad \text{R[rt]} \leftarrow \text{MEM}[\text{R[rs]} + \text{sign_ext(Imm16)}] ; \text{PC} \leftarrow \text{PC} + 4 \\
  \text{STORE} & \quad \text{MEM}[\text{R[rs]} + \text{sign_ext(Imm16)}] \leftarrow \text{R[rt]} ; \text{PC} \leftarrow \text{PC} + 4 \\
  \text{BEQ} & \quad \text{if ( R[rs] == R[rt] )} \\
  & \quad \text{then PC} \leftarrow \text{PC} + 4 + (\text{sign_ext(Imm16)} || 00) \\
  & \quad \text{else PC} \leftarrow \text{PC} + 4
  \end{align*}

Processor Design Process

- Five steps to design a processor:
  
  - **Step 1**: Analyze instruction set to determine datapath requirements
  - **Step 2**: Select set of datapath components & establish clock methodology
  - **Step 3**: Assemble datapath components that meet the requirements
  - **Step 4**: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer
  - **Step 5**: Assemble the control logic
Step 1: Requirements of the Instruction Set

- Memory (MEM)
  - Instructions & data (will use one for each)
- Registers (R: 32 x 32)
  - Read RS
  - Read RT
  - Write RT or RD
- PC
- Extender (sign/zero extend)
- Add/Sub/OR unit for operation on register(s) or extended immediate
- Add 4 (+ maybe extended immediate) to PC
- Compare registers?

Generic Steps of Datapath

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Register Write
Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements + Clocking Methodology
- Building Blocks

ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:
  
  ADDU  R[rd] = R[rs] + R[rt]; …
  SUBU  R[rd] = R[rs] - R[rt]; …
  ORI   R[rt] = R[rs] | zero_ext(Imm16)…
  BEQ   if ( R[rs] == R[rt] )…

- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if A < B, 0 otherwise)
- ALU follows Chapter 5
Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block: Address valid ⇒ Data Out valid after “access time”

Storage Element: Register (Building Block)

- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - Negated (or deasserted) (0): Data Out will not change
  - Asserted (1): Data Out will become Data In on positive edge of clock
Storage Element: Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
  - Clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after “access time.”

Step 3: Assemble DataPath Meeting Requirements

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← “something else”
Step 3: Add & Subtract

- $R_{rd} = R_{rs} \text{ op } R_{rt}$ (addu rd, rs, rt)
  - $Ra$, $Rb$, and $Rw$ come from instruction’s $Rs$, $Rt$, and $Rd$ fields
  - ALUctr and RegWr: control logic after decoding the instruction

- ... Already defined the register file & ALU

Agenda

- MIPS-lite
- Administrivia
- Technology Break
- CPU Design
Agenda

• MIPS-lite
• Administrivia
• Technology Break
• CPU Design

Agenda

• MIPS-lite
• Administrivia
• Technology Break
• CPU Timing
Clocking Methodology

- Storage elements clocked by same edge
- Flip-flops (FFs) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- “Critical path” (longest path through logic) determines length of clock period

Register-Register Timing: One Complete Cycle

- PC: Instruction Memory Access Time
- Rs, Rt, Rd, Op, Func: Delay through Control Logic
- ALUctr: Register File Access Time
- RegWr: ALU Delay
- busA, B: Register Write Occurs Here
- busW: New Value
- ALU: New Value

Old Value | New Value
---|---
PC | Instruction Memory Access Time
Rs, Rt, Rd, Op, Func | Delay through Control Logic
ALUctr | Register File Access Time
RegWr | ALU Delay
busA, B | Register Write Occurs Here
busW | New Value
ALU | New Value
Logical Operations with Immediate

- \( R[rt] = R[rs] \text{ op} \text{ ZeroExt}[imm16] \)

\[
\begin{array}{ccccccc}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
31 & 26 & 21 & 16 & 15 & 0 \\
6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 16 \text{ bits} & \\
\hline
0000000000000000 & \text{immediate} & \\
16 \text{ bits} & 16 \text{ bits} & \\
\end{array}
\]

But we’re writing to Rt register??

What about Rt register read??

- Already defined 32-bit MUX; Zero Ext?
Load Operations

- \( R[rt] = Mem[R[rs] + \text{SignExt}[\text{imm16}]] \)

Example: \( lw \ rt, rs, \text{imm16} \)
add rd, rs, rt

- MEM[PC] Fetch the instruction from memory
- PC = PC + 4 Calculate the next instruction’s address

Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory:
  Instruction = MEM[PC]
  - same for all instructions
Single Cycle Datapath during Add

\[ R[rd] = R[rs] + R[rt] \]

Instruction Fetch Unit at End of Add

- PC = PC + 4
  - Same for all instructions except: Branch and Jump
Single Cycle Datapath during Or Immediate

- \( R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}] \)

\[ \begin{array}{c|c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{imm} \\
\hline
\text{Instruction<31:0>} & \text{Imm16} \\
\hline
\text{RegDst} = 0 & \text{RegWr} = 1 & \text{ALUctr} = \text{OR} \\
\hline
\text{nPC Sel} = +4 & \text{ExtOp} = \text{zero} & \text{MemWr} = 0 \\
\end{array} \]
Single Cycle Datapath during Load

- \( R[rt] = \) Data Memory \{ \( R[rs] + \text{SignExt}[\text{imm16}] \) \}

\[
\begin{array}{cccc}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
31 & 26 & 21 & 16 & 0
\end{array}
\]

\[
\begin{array}{cccc}
\text{Instr} & \text{op} & \text{rs} & \text{rt} & \text{imm16} \\
31 & 26 & 21 & 16 & 0
\end{array}
\]

10/30/10 - Lecture #26
Summary: Datapath’s Control Signals

- **ExtOp:** “zero”, “sign”
- **ALUsrc:** 0 ⇒ regB; 1 ⇒ immed
- **ALUctr:** “ADD”, “SUB”, “OR”
- **MemWr:** 1 ⇒ write memory
- **MemtoReg:** 0 ⇒ ALU; 1 ⇒ Mem
- **RegDst:** 0 ⇒ “rt”; 1 ⇒ “rd”
- **RegWr:** 1 ⇒ write register