Anatomy: 5 components of any Computer

Personal Computer

Computer

Processor

Control ("brain")

Datapath ("brawn")

Memory

(where programs, data live when running)

Devices

Input

Output

Keyboard, Mouse

Disk
(where programs, data live when not running)

Display, Printer

This week
Outline

• Design a processor: step-by-step
• Requirements of the Instruction Set
• Hardware components that match the instruction set requirements
How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) => datapath requirements
   - meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting requirements

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic
Step 1: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. 3 formats:

  1. **R-type**
     - 6 bits
     - 5 bits
     - 5 bits
     - 5 bits
     - 5 bits
     - 6 bits

  2. **I-type**
     - 6 bits
     - 5 bits
     - 5 bits
     - 16 bits

  3. **J-type**
     - 6 bits
     - 26 bits

- The different fields are:
  - **op**: operation ("opcode") of the instruction
  - **rs, rt, rd**: the source and destination register specifiers
  - **shamt**: shift amount
  - **funct**: selects the variant of the operation in the "op" field
  - **address / immediate**: address offset or immediate value
  - **target address**: target address of jump instruction
Step 1: The MIPS-lite Subset for today

- **ADD and SUB**
  - 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
Step 1: Register Transfer Language

- RTL gives the meaning of the instructions

\{\text{op, rs, rt, rd, shamt, funct}\} = \text{MEM}[\ PC\ ]

\{\text{op, rs, rt, Imm16}\} = \text{MEM}[\ PC\ ]

- All start by fetching the instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register Transfers</th>
<th>PC = PC + 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDU</td>
<td>( R[rd] = R[rs] + R[rt]; )</td>
<td></td>
</tr>
<tr>
<td>SUBU</td>
<td>( R[rd] = R[rs] - R[rt]; )</td>
<td></td>
</tr>
<tr>
<td>ORI</td>
<td>( R[rt] = R[rs] \</td>
<td>\text{zero_ext}(\text{Imm16}); )</td>
</tr>
<tr>
<td>LOAD</td>
<td>( R[rt] = \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})];\text{PC} = \text{PC} + 4 )</td>
<td></td>
</tr>
<tr>
<td>STORE</td>
<td>( \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})] = R[rt];\text{PC} = \text{PC} + 4 )</td>
<td></td>
</tr>
<tr>
<td>BEQ</td>
<td>if ( ( R[rs] == R[rt] )) then PC = PC + 4 + ( \text{sign_ext}(\text{Imm16}) ) &lt;&lt; 2 else PC = PC + 4</td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Requirements of the Instruction Set

- Memory (MEM)
  - instructions & data
- Registers (R: 32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Extender (sign extend)
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC
Step 1: Abstract Implementation

Datapath

Ideal Instruction Memory

Control

Control Signals

Conditions

Instruction Address

Next Address

Rw Ra Rb

32 32-bit Registers

A

B

ALU

Data Address

Data In

Data Out

Ideal Data Memory

Instruction

Rd Rs Rt

5

5

5

Clk
How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) => datapath requirements
   - meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting requirements

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic (hard part!)
Step 2a: Components of the Datapath

- Combinational Elements
- Storage Elements
  - Clocking methodology
Combinational Logic: More Elements

- **Adder**
  - Inputs: A (32), B (32)
  - Outputs: Sum (32), Carry (32)

- **MUX**
  - Inputs: A (32), B (32), Select
  - Output: Y (32)

- **ALU**
  - Inputs: A (32), B (32), OP
  - Output: Result (32)
ALU Needs for MIPS-lite + Rest of MIPS

• Addition, subtraction, logical OR, ==:
  ADDU \( R[rd] = R[rs] + R[rt]; \ldots \)
  SUBU \( R[rd] = R[rs] - R[rt]; \ldots \)
  ORI \( R[rt] = R[rs] \mid \text{zero\_ext}(\text{Imm16}) \ldots \)
  BEQ if ( \( R[rs] == R[rt] \) ) \ldots

• 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 chap 5
Storage Element: Idealized Memory

• Memory (idealized)
  • One input bus: Data In
  • One output bus: Data Out

• Memory word is selected 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)
  • The 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
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)
  • The 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.”
Administrivia

• Project 2 due Friday
  • Hope you’ve already started
• HW5 (maybe HW56?) out soon
Step 3: Assemble DataPath meeting requirements

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
3a: Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC = PC + 4
    - Branch and Jump: PC = “something else”

```
<table>
<thead>
<tr>
<th>Clk</th>
<th>PC</th>
<th>Next Address Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>กก</td>
<td>ปี</td>
<td>เรื่อง</td>
</tr>
<tr>
<td>สย</td>
<td>ปี</td>
<td>เรื่อง</td>
</tr>
</tbody>
</table>
```

- Instruction Word
  - Address
  - Instruction Memory
  - Next Address Logic
  - Instruction Word 32

A Carle, Summer 2006 © UCB
3b: Add & Subtract

• \( R[rd] = R[rs] \text{ op } R[rt] \)  
  Ex.: \( \text{addU rd,rs,rt} \)

• \( Ra, Rb, \) and \( Rw \) come from instruction’s \( Rs, Rt, \) and \( Rd \) fields

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

• \( \text{ALUctr} \) and \( \text{RegWr} \): control logic after decoding the instruction

• Already defined register file, ALU
Clocking Methodology

- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) 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, 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

- **Clk**
- **PC**
  - Old Value
  - New Value
- **Rs, Rt, Rd, Op, Func**
  - Old Value
  - New Value
- **ALUctr**
  - Old Value
  - New Value
- **RegWr**
  - Old Value
  - New Value
- **busA, B**
  - Old Value
  - New Value
- **busW**
  - Old Value
  - New Value

Instruction Memory Access Time
Delay through Control Logic
Register File Access Time
ALU Delay
Register Write Occurs Here
3c: Logical Operations with Immediate

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

**What about \( Rt \) register read??**

Already defined 32-bit MUX; Zero Ext?
3d: Load Operations

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

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

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>-----------</td>
<td></td>
</tr>
</tbody>
</table>
```
3e: Store Operations

- **Mem[ R[rs] + SignExt[imm16] ] = R[rt]**
  - Example: `sw rt, rs, imm16`

![Diagram](image)
3f: The Branch Instruction

- **beq rs, rt, imm16**
  - **mem[PC]** Fetch the instruction from memory
  - **Equal = R[rs] == R[rt]** Calculate branch condition
  - **if (Equal)** Calculate the next instruction’s address
    - **PC = PC + 4 + ( SignExt(imm16) x 4 )**
  - **else**
    - **PC = PC + 4**
Datapath for Branch Operations

- beq rs, rt, imm16
  Datapath generates condition (equal)

\[
\begin{array}{c|c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
\hline
\text{6 bits} & \text{5 bits} & \text{5 bits} & \text{16 bits} \\
\end{array}
\]

- Already MUX, adder, sign extend, zero
Putting it All Together: A Single Cycle Datapath
Peer Instruction

A. Our **ALU** is a synchronous device

B. We should use the **main ALU** to compute PC=PC+4

C. The **ALU is inactive** for memory reads or writes.
Summary: Single cycle datapath

- 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 affects the register transfer.
  5. Assemble the control logic

- Control is the hard part
- Next time!