Dell may have shipped infected motherboards⇒

Dell is warning customers that its PowerEdge R410 rack-mount server may have shipped with spyware onboard in the embedded system management software (firmware). Good luck getting Norton McAfee to clean that.

http://www.theregister.co.uk/2010/07/21/dell_server_warning/
How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) => datapath \textit{requirements}  
   • meaning of each instruction is given by the \textit{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
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 (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

- **Clk**: Clock signal
- **PC**: Program Counter
- **Rs, Rt, Rd, etc**: Source registers
- **ALUctr**: ALU control
- **RegWr**: Register write
- **busA, B**: Bus signals
- **busW**: Bus write
- **RegFile**: Register file
- **ALU**: Arithmetic Logic Unit

**Instruction Memory Access Time**
- Old Value → New Value

**Delay through Control Logic**
- Old Value → New Value

**Register File Access Time**
- Old Value → New Value

**ALU Delay**
- Old Value → New Value

**Register Write Occurs Here**
3c: Logical Operations with Immediate

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

But we’re writing to Rt register??
3c: Logical Operations with Immediate

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

<table>
<thead>
<tr>
<th>op</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>immediate (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

What about $Rt$ register read??

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

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

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

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
</tr>
</tbody>
</table>

6 bits | 5 bits | 5 bits | 16 bits

![Diagram of CPU design](image)
3d: Load Operations

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

Example: `lw rt, rs, 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>

- **RegDst**
- **Rd**
- **Rt**
- **RegWk**
- **Rs**
- **Rt**
- **busW**
- **RegFile**
- **Rw**
- **Ra**
- **Rb**
- **clk**
- **Extender**
- **imm16**
- **16 bits**

- **ALUctr**
- **MemtoReg**
- **MemWk**
- **ALUSrc**
- **clk**
- **ExtOp**
- **Data Memory**

```
Example: lw rt, rs, 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>
```
3e: Store Operations

- Ex.: sw rt, rs, imm16

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

![Diagram of CPU design with labels and connections for store operations]
### 3e: Store Operations

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

```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>
```

- **ALUctr**
  - `RegDst`, `Rd`, `Rt` to `ALUctr`
- **MemtoReg**
  - `MemWr` to `MemtoReg`
- **RegFile**
  - `busA`, `busB` to `RegFile`
- **ALU**
  - `busA`, `busB` to `ALU`
- **Extender**
  - `imm16` to `Extender`
- **Data Memory**
  - `Data In` to `Data Memory`
  - `WrEn`, `Adr` to `Data Memory`

---

**RegFile**

- `Rw`, `Ra`, `Rb`
- `clk`
- `imm16`
- `ExtOp`

**ALU**

- `32`
- `WrEn`, `Adr`
- `Data In`
- `Data Memory`
- `clk`

---

**MemtoReg**

- `MemWr`

---

**ALUctr**

- `RegDst`, `Rd`, `Rt`

---

**Extender**

- `imm16`
- `32`
- `16`

---

**Cal**

---

CS61C L19 CPU Design: The Single-Cycle II & Control (10) Pearce, Summer 2010 © UCB
3f: The Branch Instruction

beq rs, rt, imm16

- mem[PC] Fetch the instruction from memory
- Zero = R[rs] - R[rt] Calculate branch condition
- if (Zero) 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 (zero)**

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

Inst Address

- Already have mux, adder, need special sign extender for PC, need equal compare (sub?)
An Abstract View of the Critical Path

Critical Path (Load Instruction) =
Delay clock through PC (FFs) +
Instruction Memory’s Access Time +
Register File’s Access Time, +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Stable Time for Register File Write

(Assumes a fast controller)
**Administrivia**

- HW 7 due Saturday
  - Specification update now online. Be sure to check it out. The changes are designed to make integration with project 2 easier. They are very minor.
- Project 2 online now!
- Reminder: Midterm regrades due Monday.
- Paul will be gone on Monday. Noah will be giving the lecture.
  - Paul will have OH on Monday (barring flight delays)
- Be prepared for a kinetic learning activity
Review: A Single Cycle Datapath

- We have everything but the **values of control signals**.
Recap: Meaning of the Control Signals

- **nPC_sel**: 
  - “+4” 0 ⇒ PC ← PC + 4
  - “br” 1 ⇒ PC ← PC + 4 + \{SignExt(Im16), 00\}

- Later in lecture: higher-level connection between mux and branch condition
Recap: Meaning of the Control Signals

- **ExtOp**: zero, sign
- **ALUsrc**: 0 ⇒ reg; 1 ⇒ imm
- **ALUctrl**: ADD, SUB, OR

° **MemWr**: 1 ⇒ write memory
° **MemtoReg**: 0 ⇒ alu; 1 ⇒ mem
° **RegDst**: 0 ⇒ rt; 1 ⇒ rd
° **RegWr**: 1 ⇒ write register
### RTL: The `add` Instruction

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
<td></td>
</tr>
<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>
<td></td>
</tr>
</tbody>
</table>

#### `add rd, rs, rt`

- **MEM[PC]**: Fetch the instruction from memory
- **R[rd] = R[rs] + R[rt]**: The actual operation
- **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

![Diagram showing instruction fetch unit]
The Single Cycle Datapath during Add

\[ R[rd] = R[rs] + R[rt] \]
Instruction Fetch Unit at the End of Add

- \( \text{PC} = \text{PC} + 4 \)
- This is the same for all instructions except: Branch and Jump

\[ \text{Inst Address} \]

\[ \text{Inst Memory} \]

\[ \text{nPC}_{\text{sel}} = +4 \]

\[ \text{clk} \]

\[ \text{imm}16 \]

\[ \text{PC Ext} \]
Single Cycle Datapath during Or Immediate?

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

- \( \text{RegDst=} \)
- \( \text{RegWr=} \)
- \( nPC \_ \text{sel=} \)
- \( \text{ExtOp=} \)
- \( \text{ALUSrc=} \)
- \( \text{MemtoReg=} \)
- \( \text{MemWr=} \)
- \( \text{WrEn Adr} \)
- \( \text{Data Memory} \)
- \( \text{Extender} \)
- \( \text{Data In} \)
- \( \text{Z} \)
- \( \text{ALU} \)
- \( \text{busA} \)
- \( \text{busB} \)
- \( \text{RegFile} \)
- \( \text{Instr fetch unit} \)
- \( \text{instr fetch unit} \)
- \( \text{Instruction<31:0>} \)
- \( \text{Rs} \)
- \( \text{Rt} \)
- \( \text{Rd} \)
- \( \text{Imm16} \)
- \( \text{clk} \)

- Instruction<31:0>
\[ R[rt] = R[rs] \text{ OR ZeroExt[Imm16]} \]
The Single Cycle Datapath during Load?

- \( R[rt] = \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\} \)
The Single Cycle Datapath during Load

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

```
31 26 21 16  0
| op | rs | rt | immediate |
```

- \( \text{nPC\_sel} = +4 \)
- \( \text{RegDst} = rt \)
- \( \text{RegWr} = 1 \)
- \( \text{busW} = 32 \)
- \( \text{busA} = 32 \)
- \( \text{busB} = 32 \)
- \( \text{imm16} = 16 \)
- \( \text{ExtOp} = \text{sign} \)
- \( \text{ALUSrc} = \text{imm} \)
- \( \text{MemtoReg} = \text{mem} \)
- \( \text{MemWr} = 0 \)
- \( \text{ALUctr} = \text{ADD} \)
- \( \text{Mem\_Sel} = +4 \)

```
<table>
<thead>
<tr>
<th>instr fetch unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction&lt;31:0&gt;</td>
</tr>
<tr>
<td>Rs</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>zero</td>
</tr>
</tbody>
</table>

```

- \( \text{clk} \)
- \( \text{RegFile} \)
- \( \text{Data Memory} \)
- \( \text{Z}\)
The Single Cycle Datapath during Store?

- Data Memory \( \{R[rs] + \text{SignExt}[\text{imm16}]\} = R[rt] \)
The Single Cycle Datapath during Store

- Data Memory $\{R[rs] + \text{SignExt}[\text{imm16}]\} = R[rt]$
The Single Cycle Datapath during Branch?

- if \( (R[rs] - R[rt] == 0) \) then Zero = 1 ; else Zero = 0
The Single Cycle Datapath during Branch

- if \((R[rs] - R[rt] == 0)\) then \(Zero = 1\); else \(Zero = 0\)
Instruction Fetch Unit at the End of Branch

- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ;
- else PC = PC + 4

What is encoding of nPC_sel?

- Direct MUX select?
- Branch inst. / not branch

Let’s pick 2nd option

- nPC_sel = +4 \rightarrow 0
- nPC_sel = br \rightarrow 1

Q: What logic gate?
1) In the worst case, the biggest delay is the memory access time.

2) With only changes to control, our datapath could write (something) to memory and registers in one cycle.

Options:

12
a) FF
b) FT
c) TF
d) TT
Summary: A Single Cycle Datapath

- We have everything! Now we just need to know how to BUILD CONTROL!
Summary: Single-cycle Processor

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
   - Formulate Logic Equations
   - Design Circuits