DARPA $s drying up…ouch!

“I'm worried and depressed,”
– David Patterson, president of the ACM.

There is a significant shift of $ from “Blue Sky” research to military contractors. This is a significant shift, mostly for the worse.
Review: 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 effects the register transfer.
5. Assemble the control logic

Control is the hard part

MIPS makes that easier

- Instructions same size
- Source registers always in same place
- Immediates same size, location
- Operations always on registers/immediates
# A Summary of the Control Signals (2/2)

<table>
<thead>
<tr>
<th>func</th>
<th>add</th>
<th>sub</th>
<th>ori</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>op 00 0000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>op 00 0010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>op 00 1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>op 10 0011</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>op 10 1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>op 00 0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>op 00 0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>op 10 0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>op 10 0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ALUctr&lt;2:0&gt;</td>
<td>Add</td>
<td>Subtract</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td>x</td>
</tr>
</tbody>
</table>

**R-type**
- op | rs | rt | rd | shamt | funct
- add, sub

**I-type**
- op | rs | rt | immediate
- ori, lw, sw, beq

**J-type**
- op | target address
- jump
• New PC = { PC[31..28], target address, 00 }
The Single Cycle Datapath during Jump

- New PC = \{ PC[31..28], target address, 00 \}
Instruction Fetch Unit at the End of Jump

- New PC = { PC[31..28], target address, 00 }

How do we modify this to account for jumps?
Instruction Fetch Unit at the End of Jump

- New PC = { PC[31..28], target address, 00 }

Query
- Can Zero still get asserted?
- Does nPC_sel need to be 0?
  - If not, what?
Build CL to implement Jump on paper now
Build CL to implement Jump on paper now.
Administrivia

• HW7 out today, due in a week
Review: Finite State Machine (FSM)

- **States**
  represent possible output values.

- **Transitions**
  represent changes between states based on inputs.

- **Implement**
  with CL and clocked register feedback.
Finite State Machines extremely useful!

- They define
  - How output signals respond to input signals and previous state.
  - How we change states depending on input signals and previous state.

- The output signals could be our familiar control signals
  - Some control signals may only depend on CL, not on state at all...

- We could implement very detailed FSMs with Programmable Logic Arrays
Taking advantage of sum-of-products

• Since sum-of-products is a convenient notation and way to think about design, offer hardware building blocks that match that notation

• One example is **Programmable Logic Arrays** (PLAs)

• Designed so that can select (program) ands, ors, complements **after** you get the chip
  
  • Late in design process, fix errors, figure out what to do later, …
Programmable Logic Arrays

• Pre-fabricated building block of many AND/OR gates
  • “Programmed” or “Personalized" by making or breaking connections among gates
  • Programmable array block diagram for sum of products form

And Programming:
• How many inputs?
• How to combine inputs?
• How many product terms?

Or Programming:
• How to combine product terms?
• How many outputs?
Enabling Concept

• Shared product terms among outputs

\[
\begin{align*}
F_0 &= A + B'C' \\
F_1 &= A'C' + AB \\
F_2 &= B'C' + AB \\
F_3 &= B'C + A
\end{align*}
\]

example:

<table>
<thead>
<tr>
<th>Personality Matrix</th>
<th>Product term</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>inputs</td>
</tr>
<tr>
<td>A       B       C</td>
<td>F0  F1  F2  F3</td>
</tr>
<tr>
<td>AB</td>
<td>1   1   -</td>
</tr>
<tr>
<td>B'C</td>
<td>-   0   1</td>
</tr>
<tr>
<td>AC'</td>
<td>1   -   0</td>
</tr>
<tr>
<td>B'C'</td>
<td>-   0   0</td>
</tr>
<tr>
<td>A</td>
<td>1   -   -</td>
</tr>
</tbody>
</table>

- **input side:** 3 inputs
  - 1 = uncomplemented in term
  - 0 = complemented in term
  - - = does not participate

- **output side:** 4 outputs
  - 1 = term connected to output
  - 0 = no connection to output

reuse of terms; 5 product terms
Before Programming

- All possible connections available before “programming”
After Programming

• Unwanted connections are "blown"
  • Fuse (normally connected, break unwanted ones)
  • Anti-fuse (normally disconnected, make wanted connections)
Alternate Representation

- Short-hand notation—don't have to draw all the wires
  - X Signifies a connection is present and perpendicular signal is an input to gate

notation for implementing

\[
F_0 = A \overline{B} + A' \overline{B}' \\
F_1 = C \overline{D} + C' D
\]
A. MemToReg=‘x’ & ALUctr=‘sub’. SUB or BEQ?

B. ALUctr=‘add’. Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?

C. “Don’t Care” signals are useful because we can simplify our PLA personality matrix. F / T?
And in Conclusion... Single cycle control

° 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

° Control is the hard part

° MIPS makes that easier
  • Instructions same size
  • Source registers always in same place
  • Immediates same size, location
  • Operations always on registers/immediates