Microprogramming

Datapath and control
Set 4 -
microprogramming

modern advertizing

When'd you get the car?

It's the darndest thing, we bought it as a prize for the 100,000th visitor to our website.

And they didn't want it? Apparently.

Maybe they didn't see the notice. It was flashing and everything!

How bizarre.
Overview

- We’ll consider the following topics over the next two lessons:
  - Tracing instruction execution through a multi-cycle datapath
  - Specifying the control necessary to perform that execution – using FSMs or Microinstructions
  - How to implement the control given the control specification.

*Goal: Implement a control for the multi-cycle datapath using microprogramming techniques*

Tracing Multi-Cycle Execution

*Breaking instruction execution into multiple cycles*

1. Instruction fetch step
2. Instruction decode and register fetch step
3. Execution, memory address computation, or branch completion
4. Memory access or R-type instruction completion
5. Memory read completion step – write to register file

*Instructions take from 3-5 cycles!!*
Specifying the Multi-Cycle Control
Describing relationship of control inputs to outputs

- Value of control signals is dependent upon:
  - what *instruction* is being executed
  - which *step* is being performed

- Use the information we’ve accumulated to specify a finite state machine
  - specify the *finite state machine* (FSM) graphically, or
  - use *microprogramming*

- Implementation can be derived from specification

---

Each unlabeled arrow represents an unconditional transition to the target state.

Each labeled arrow represents a selectable transition to the target state.

*This type of specification doesn’t scale well to large instruction sets!*
Specifying the Multi-Cycle Control

Using Microinstructions

• Each microinstruction defines...
  – The set of control signals (de)asserted in a given state.
  – The way to get to the next state.
• Each microinstruction is composed of a number of fields...
  – The first field is a label like those used in Assembly
  – The last field determines the next microinstruction
    • Seq – simply execute the next microinstruction in the program
    • Fetch – Start the next assembly instruction
    • Dispatch – choose the next microinstruction based on the opcode of the assembly instruction and the specified dispatch table
  – The other fields hold the value of specific control signals

Tracing Multi-Cycle Execution:

Step 1: Instruction Fetch (Same for all instructions)

• Use PC to get instruction and put it in the Instruction Register.
• Increment the PC by 4 and put the result back in the PC.
• Can be described succinctly using the Register Transfer Language (RTL)

\[
\begin{align*}
IR &= \text{Memory}[PC]; \\
PC &= PC + 4;
\end{align*}
\]
The Multi-Cycle Datapath

Step 1: Instruction Fetch (Same for all instructions)

If we were to design a finite state for all five steps, it would start out looking like this...

Finite State Machine – described by a finite number of states (each consisting of a number of signals to be asserted and de-asserted) and the inputs necessary to transition from each state to another state, if possible.
Specifying Control with Microprogramming

Step 1: Instruction Fetch (Same for all instructions)

Goal: IR = Memory[PC];
     PC = PC + 4;

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
</tbody>
</table>

We start here at the start of the execution of each assembly instruction.

Unconditionally go to the next instruction.

Check out P&H page 404 (2nd edition)

The Multi-Cycle Datapath:

Step 2: Instruction Decode & Register Fetch (Same for all instructions!!)

- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- RTL:
  
  \[
  A = \text{Reg}[\text{IR}[25-21]]; \\
  B = \text{Reg}[\text{IR}[20-16]]; \\
  \text{ALUOut} = \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2);
  \]

- We aren't setting any control lines based on the instruction type
  (we are busy "decoding" it in our control logic)
The Multi-Cycle Datapath:
Step 2: Instruction Decode & Register Fetch

1. A = Reg[IR[25-21]]

2. B = Reg[IR[20-16]]

3. ALUOut = PC + (sign-extend(IR[15-0]) << 2)

Specifying Control with an FSM
Step 2: Instruction Decode & Register Fetch

Start
Specifying Control with Microprogramming

Step 2: Instruction Decode & Register Fetch

Goal: \[ A = \text{Reg}[IR[25-21]]; \]
\[ B = \text{Reg}[IR[20-16]]; \]
\[ \text{ALUOut} = \text{PC} + (\text{sign-extend}(IR[15-0]) \ll 2); \]

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>PC</td>
<td>Extshift</td>
<td>Read</td>
<td></td>
<td>Dispatch 1</td>
<td></td>
</tr>
</tbody>
</table>

Use dispatch table 1 to choose the next microinstruction address.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Fields</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ A = \text{Reg}[IR[25-21]]; ] [ B = \text{Reg}[IR[20-16]]; ]</td>
<td>Register Control</td>
<td>Use the rs and rt fields to select which registers to read. Place those values in ( A ) and ( B ) reg.</td>
</tr>
<tr>
<td>[ \text{ALUOut} = \text{PC} + (\text{sign-extend}(IR[15-0]) \ll 2); ]</td>
<td>ALU control, SRC1, SRC2</td>
<td>Store ( \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2) ) into ( \text{ALUOut} )</td>
</tr>
</tbody>
</table>

The Multi-Cycle Datapath:

Steps 3-5

- Steps 3-5 vary with the instruction.
- Let’s investigate how each instruction executes through these steps
The Multi-Cycle Datapath:
Step 3: Supporting the j instruction

- Step 3: Update PC:

\[ PC = PC[31-28] \text{\[31-28\]} + \text{sign-extend(IR[25-0])} \ll 2; \]

- Done!
Specifying Control with an FSM

Step 3: Supporting the j instruction

Start → Fetch → MemRead → ALUSrcA=0, ALUSrcB=0, IRWrite = 0 → ALUSrcA=0, ALUSrcB=11 → ALUOp=00, PCWrite → PCWrite, PCSource=10 → Jump Complete

Specifying Control with Microprogramming

Step 3: Supporting the j instruction

Goal: \( PC = PC[31-28] + \text{sign-extend}(IR[25-0]) \ll 2; \)

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>Jump1</td>
<td>Add</td>
<td>PC</td>
<td>Extshift</td>
<td>Read</td>
<td>Jump Address</td>
<td>Dispatch 1</td>
<td></td>
</tr>
</tbody>
</table>

Label for Dispatch 1

Time to start next Assembly instruction!

Goal Fields Effect

\( PC = PC[31-28] + \text{sign-extend}(IR[25-0]) \ll 2; \) PCWrite Control Causes PC to be written with the j target address.
The Multi-Cycle Datapath:  
Step 3: Supporting the \texttt{beq} instruction

- Step 3: Update PC:

  \[
  \text{if (A==B) PC = ALUOut;}
  \]

- Done!
Specifying Control with an FSM

Step 3: Supporting the `beq` instruction

![FSM Diagram]

Specifying Control with Microprogramming

Step 3: Supporting the `beq` instruction

**Goal:** \( \text{if} \ (A==B) \ \text{PC} = \text{ALUOut}; \)

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>PC</td>
<td>ExtShift</td>
<td>Read</td>
<td></td>
<td></td>
<td>Dispatch 1</td>
<td></td>
</tr>
<tr>
<td>Jump1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Jump Address</td>
<td>Fetch</td>
</tr>
<tr>
<td>Beq1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALUOut-Cond</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>

**Label for Dispatch 1**

<table>
<thead>
<tr>
<th>Goal</th>
<th>Fields</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ((A==B)) PC = ALUOut;</td>
<td>ALU Control, SRC1, SRC2</td>
<td>ALU subtracts A from B to determine value for Zero</td>
</tr>
<tr>
<td></td>
<td>PCWrite Control</td>
<td>Causes the PC to be written with the value in ALUOut</td>
</tr>
</tbody>
</table>
The Multi-Cycle Datapath:
Steps 3-4: Supporting the R-type instruction

- Step 3: Store result in ALUOut

\[ \text{ALUOut} = \text{A op B}; \]

- Step 4: Write result to the write register

\[ \text{Reg}[\text{IR}[15-11]] = \text{ALUOut}; \]
The Multi-Cycle Datapath:
Step 4: Write result to the write register

Specifying Control with an FSM
Steps 3 & 4: Supporting the R-type instruction
Specifying Control with Microprogramming
Steps 3 & 4: Supporting the R-type instruction

Goal: \( ALUOut = A \text{ op } B; \)
\( \text{Reg}[IR[15-11]] = ALUOut; \)

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>PC</td>
<td>Extshft</td>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALUOut-Cond</td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beq1</td>
<td>Func Code</td>
<td>A</td>
<td>B</td>
<td>Seq</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rformat1</td>
<td>Func Code</td>
<td>A</td>
<td>B</td>
<td>Write ALU</td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal | Fields | Effect
---|---|---
Step 3: \( ALUOut = A \text{ op } B; \) | ALU Control, SRC1, SRC2 | Use the funct code, A & B regs to compute some value
Step 4: \( \text{Reg}[IR[15-11]] = ALUOut; \) | Register Control | The value computed is written to the write register

The Multi-Cycle Datapath:
Steps 3-4: Supporting the sw instruction

- Step 3: Compute the memory address
  \[ ALUOut = A + \text{sign-extend}(IR[15-0]); \]
- Step 4: Write result to memory
  \[ \text{Memory}[ALUOut] = B; \]
The Multi-Cycle Datapath:
Step 3: Compute the Memory Address

Step 4: Write the result to memory
Specifying Control with an FSM
Steps 3-4: Supporting the \texttt{sw} instruction

```
Label | ALU Control | SRC1 | SRC2 | Register Control | Memory | PCWrite Control | Sequencing
Fetch  | Add          | PC   | 4    | Read PC          | ALU    | Seq             | Seq
       | Add          | PC   | ExtShift | Read              |        |                 | Dispatch 1
Jump1  |              |      |       |                  |        | Jump Address    | Fetch
Beq1   | Subt         | A    | B    | ALUOut-Cond      |        | Fetch           |              
Rformat1 | Func Code   | A    | B    |                  |        | Seq             |              
Mem1   | Add          | A    | Extend | Write ALU        |        | Fetch           |              
SW 2   |              |      |       |                  |        |                 |              
```

Specifying Control with Microprogramming
Steps 3 & 4: Supporting the \texttt{sw} instruction

Goal: \( ALUOut = A + \text{sign-extend}(IR[15-0]) \);

\( Memory[ALUOut] = B \);

Goal Fields Effect

<table>
<thead>
<tr>
<th>Step 3: ( ALUOut = A + \text{sign-extend}(IR[15-0]) );</th>
<th>ALU Control, SRC1, SRC2</th>
<th>Compute the memory address for the \texttt{sw/lw}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 4: ( Memory[ALUOut] = B );</td>
<td>Memory</td>
<td>Write contents of B to memory using address in ALUOut</td>
</tr>
</tbody>
</table>
• Step 3: Compute the memory address

\[ \text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0]); \]

• Step 4: Retrieve value from memory

\[ \text{MDR} = \text{Memory}[\text{ALUOut}]; \]

• Step 5: Write the value to the write reg

\[ \text{Reg}[\text{IR}[20-16]] = \text{MDR}; \]
The Multi-Cycle Datapath:  
Step 4: Retrieve value from memory

Step 5: Write the value to the write reg
Specifying Control with an FSM
Steps 3-5: Supporting the `lw` instruction

Specifying Control with Microprogramming
Steps 3-5: Supporting the `lw` instruction

**Goal:**
- \( \text{ALUOut} = A + \text{sign-extend} (\text{IR}[15-0]); \)
- \( \text{MDR} = \text{Memory}[\text{ALUOut}]; \)
- \( \text{Reg}[\text{IR}[20-16]] = \text{MDR}; \)

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register Control</th>
<th>Memory</th>
<th>PCWrite Control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>PC</td>
<td>Extshft</td>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump1</td>
<td></td>
<td></td>
<td></td>
<td>Jump Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beq1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALUOut-Cond</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rformat1</td>
<td>Func Code</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem1</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Write ALU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW 2</td>
<td>Add</td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW 2</td>
<td></td>
<td></td>
<td></td>
<td>Read ALU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write MDR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal Fields Effect**

- **Step 4:** \( \text{MDR} = \text{Memory}[\text{ALUOut}]; \)
  - Memory: Read memory using \( \text{ALUOut} \) as the memory address

- **Step 5:** \( \text{Reg}[\text{IR}[20-16]] = \text{MDR}; \)
  - Register Control: Write the contents of MDR to the write register.
Implementing Control
Implement using a block of combinational logic and state register

Implementing Control
μcode in storage & explicit μPC
Implementing Control
μcode in storage & explicit μPC

Control Unit

Microcode Storage

Outputs

Input

1

Adder

Microprogram Counter

Address Select Logic

PCWrite
PCWriteCond
Load
MemRead
MemWrite
Write
MemRef
PCInSel
ALUOrd
ALUExt
ALUsrcA
MathWrite
MathClose

Sequencing Control

16 Control outputs

4 Sequence choices (2 bits)

IR – Opcode Field

Implementing Control
μcode in storage & explicit μPC

Microprogram Counter

Address Select Logic

3 2 1 0

Dispatch ROM 1

Dispatch ROM 2

IR – Opcode Field
Implementing Control

μcode in storage & explicit μPC

This slide mimics the old text but needs clarification (see next slide)
Implementing Control (reordered)

μcode in storage & explicit μPC

<table>
<thead>
<tr>
<th>state</th>
<th>Label</th>
<th>ALU_CTL</th>
<th>SRC1</th>
<th>SRC2</th>
<th>REGCtl</th>
<th>Mem</th>
<th>PCWrite</th>
<th>Seq</th>
<th>Mux</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Fetch</td>
<td>Add PC</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Add</td>
<td>PC ExtOr Read</td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Mem1</td>
<td>Add A</td>
<td>A Ext</td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>LW2</td>
<td>Read PC</td>
<td>ALU</td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Write MDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>SW2</td>
<td>Write ALU</td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Store</td>
<td>Func Code</td>
<td>A</td>
<td>B</td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Write ALU</td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Beq1</td>
<td>Subt A</td>
<td>B</td>
<td>ALUOut-Cond</td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Jump1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
<td>Seq</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Implementing Control

μcode in storage & explicit μPC

Control Unit

Microcode Storage

Outputs

PCWrite
PCWriteCond
Cond
MmWrite
MmRead
Type
MemWrite
MemRef
MemReg
PCSource
ALUSrcA
ALUSrcB
RegWrite
RegRead

Sequence choices (2 bits)

16 Control outputs

4 Sequence choices (2 bits)

16

Adder

Microprogram Counter

Address Select Logic

Sequencing Control

IR – Opcode Field
Control Signal Values

<table>
<thead>
<tr>
<th>Signal</th>
<th>fetch</th>
<th>decode</th>
<th>jump</th>
<th>beq</th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemRead</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IRWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IorD</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Mem Read</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mem Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IR write</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mem to reg</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>PC Read</td>
<td>00</td>
<td>X</td>
<td>X</td>
<td>10</td>
<td>XX</td>
<td>XX</td>
</tr>
<tr>
<td>ALU OP</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>XX</td>
</tr>
<tr>
<td>ALU A</td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>XX</td>
<td>XX</td>
</tr>
<tr>
<td>Reg Write</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reg Dest</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>Sequence</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>State</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

CS 2160 Spring, 2011
Multi Cycle Control