Topics

- Microinstructions
  - Tanenbaum 4.3.1 & 4.3.2
- Pipelining (Mic-3)
  - Tanenbaum 4.4.4
- Data Hazards

Mic-1 Simulator Window

- **Microinstruction** – current microinstruction being executed
- **NextMicroinstruction** – next microinstruction to be executed
- **MPC** – current location in the control store
- **Registers** – value stored in each microarchitecture register
### IJVM Code Fragment

```
..constant
Fall 231
Spring 343
Interstate 65
November 11
..end-constant
..var
w
x
y
z
..end-var
..main
    BIPUSH 7
    ISTORE x
    ...
    HALT
..end-main
```

### Stepping Through Microinstructions

**Step 1**

- Instruction: `PC = PC + 1; fetch; goto (MBR)`

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>PC</th>
<th>MDR</th>
<th>PC</th>
<th>MBR</th>
<th>SP</th>
<th>LV</th>
<th>CPP</th>
<th>TOS</th>
<th>OPC</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>0x10</td>
<td>0x07</td>
<td>0x36</td>
<td>0x02</td>
<td>603</td>
<td>600</td>
<td>100</td>
<td>65</td>
<td>343</td>
</tr>
</tbody>
</table>

- Main loop is executed
- Multiway branch to BIPUSH microinstruction sequence
Stepping Through Microinstructions

### Step 2

**Instruction:** \( SP = MAR = SP + 1 \)

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Start of cycle</th>
<th>End of cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR</td>
<td>1</td>
<td>0x10</td>
<td>604</td>
</tr>
<tr>
<td>MDR</td>
<td>2</td>
<td>0x07</td>
<td>604</td>
</tr>
<tr>
<td>PC</td>
<td>3</td>
<td>0x36</td>
<td>604</td>
</tr>
<tr>
<td>MBR</td>
<td>4</td>
<td>0x02</td>
<td>604</td>
</tr>
<tr>
<td>SP</td>
<td>603</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>LV</td>
<td>600</td>
<td>604</td>
<td></td>
</tr>
<tr>
<td>CPP</td>
<td>100</td>
<td>101</td>
<td>11</td>
</tr>
<tr>
<td>TOS</td>
<td>102</td>
<td>102</td>
<td>65</td>
</tr>
<tr>
<td>OPC</td>
<td>103</td>
<td>101</td>
<td>343</td>
</tr>
<tr>
<td>H</td>
<td>104</td>
<td>101</td>
<td>231</td>
</tr>
</tbody>
</table>

- Stack Pointer and Memory Address Register updated
- Next byte is fetched into Memory Buffer Register

---

Stepping Through Microinstructions

### Step 3

**Instruction:** \( PC = PC + 1; \text{ fetch} \)

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Start of cycle</th>
<th>End of cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR</td>
<td>604</td>
<td>1</td>
<td>0x10</td>
</tr>
<tr>
<td>MDR</td>
<td>2</td>
<td>0x07</td>
<td>604</td>
</tr>
<tr>
<td>PC</td>
<td>3</td>
<td>0x36</td>
<td>604</td>
</tr>
<tr>
<td>MBR</td>
<td>4</td>
<td>0x02</td>
<td>604</td>
</tr>
<tr>
<td>SP</td>
<td>604</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>LV</td>
<td>600</td>
<td>604</td>
<td></td>
</tr>
<tr>
<td>CPP</td>
<td>100</td>
<td>101</td>
<td>11</td>
</tr>
<tr>
<td>TOS</td>
<td>102</td>
<td>102</td>
<td>65</td>
</tr>
<tr>
<td>OPC</td>
<td>103</td>
<td>101</td>
<td>343</td>
</tr>
<tr>
<td>H</td>
<td>104</td>
<td>101</td>
<td>231</td>
</tr>
</tbody>
</table>

- Program Counter is updated
- Initiate a fetch operation
Stepping Through Microinstructions

- Operand is stored in Top Of Stack and Memory Data Register
- Next byte is fetched into Memory Buffer Register

Performance Limitations of Mic-1

- Only H register is connected to A input of ALU
  - Requires two cycles for ALU operation on two arbitrary registers
- Long cycle time with idle sections
- Many microinstructions to execute each IJVM instruction
- Many instructions use datapath to update PC for fetch
- Fetching and assembling 2-byte offset
Desirable Characteristics

- Cheap
- Low-power
- Fast
- Reliable
- Small chip size

Trade-off: Speed vs. Cost drives most design choices

Improving Speed with Architecture Design

Three basic approaches
- Reduce number of clock cycles needed to execute an instruction
- Simplify organization to shorten clock cycle
- Overlap execution of instructions
  - Finding “independent” operations

Generally have some serial operations that affect cycle time
Datapath of Mic-2 Microarchitecture

- **Three-bus architecture**
  - Possible to add any register to any other register in a single cycle

- **Instruction Fetch Unit (IFU)**
  - Efficiently fetch and process instruction stream

- **Merge interpreter Main loop with microcode (Fig 4-30)**
  - Fully utilize instruction

Timing Diagram for Mic-2 Datapath Cycle
Major Components of Datapath Cycle

- Time to drive selected registers onto A bus and B bus
- Time for ALU and shifter to execute
- Time for results to get back to registers and be stored

Highly sequential in the execution
Must wait for each stage to complete

Pipelining Lessons

- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Use different resources to execute multiple tasks simultaneously
- Potential speedup = Number pipe stages
- Time to “fill” pipeline and time to “drain” it reduces speedup
- Stall for Dependences

Adapted from David Patterson’s CS 252 lecture notes. Copyright 2001 UCB.
Datapath of Mic-3 Microarchitecture

- Add latches on A bus, B bus, and C bus

- Benefits
  - Speed up the clock cycle because maximum delay is shorter
  - Use all parts of datapath during every cycle

Implementation of SWAP Instruction

<table>
<thead>
<tr>
<th>Label</th>
<th>Operations</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>swap1</td>
<td>MAR = SP - 1; rd</td>
<td>Read 2nd word from stack; set MAR to SP</td>
</tr>
<tr>
<td>swap2</td>
<td>MAR = SP</td>
<td>Prepare to write new 2nd word</td>
</tr>
<tr>
<td>swap3</td>
<td>H = MDR; wr</td>
<td>Save new TOS; write 2nd word to stack</td>
</tr>
<tr>
<td>swap4</td>
<td>MDR = TOS</td>
<td>Copy old TOS to MDR</td>
</tr>
<tr>
<td>swap5</td>
<td>MAR = SP - 1; wr</td>
<td>Write old TOS to 2nd place on stack</td>
</tr>
<tr>
<td>swap6</td>
<td>TOS = H; goto (MBR1)</td>
<td>Update TOS</td>
</tr>
</tbody>
</table>

Figure 4.32. The Mic-2 code for SWAP.

Figure 4.33. The implementation of SWAP on the Mic-3.
Data Hazards

• Created whenever there is a dependence between instructions where pipelining overlap changes the order of access

• True dependence: Read After Write (RAW)
  – Instructions are dependent upon previous data results that aren’t available
  – Causes stalls in the pipeline to wait for data

• Other hazard types include WAW and WAR
  – Can be handled with register renaming
  – Note that RAR is not a hazard