Outline

❖ Last time:
  ➔ Introduction to Computer Organization
  ➔ Control
  ➔ Datapath
  ➔ I/O Interface
  ➔ Bussing Strategies
❖ This lecture:
  ➔ Deriving the State Diagram & Datapath (Cont.)
  ➔ Mapping the Datapath onto Control

Finite State Machines for Simple CPUs

**State Diagram and Datapath Derivation**

Processor Specification:
Instruction Format:

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Code</td>
<td>LD</td>
<td>ST</td>
<td>ADD</td>
<td>BRN</td>
</tr>
</tbody>
</table>

Load from memory: Mem[XXX] → AC;
Store to memory:  AC → Mem[XXX];
Add from memory:  AC + Mem[XXX] → AC;
Branch if accumulator is negative: AC < 0 ⇒ XXX → PC;

Memory Interface:

- M: Memory
- A: Address
- R: Request
- B: Bitwidth
- MBR: Memory Buffer Request
- 14: Memory Request
- 16: Memory Bitwidth
Finite State Machines for Simple CPUs

Deriving the State Diagram and Datapath

First pass state diagram:

```
  Reset
  /     \
//      \/
Instruction Fetch
  \
//      \/
Operation Decode
  \
//      \/
LD     ST     ADD     BRN
  \
//      \/
Operation Execution
```

Assume Synchronous Mealy Machine:
Transitions associated with arcs rather than states

Reset State (State 0) and Instruction Fetch Sequence

On Reset:
- zero the PC
- Mem Request unasserted
- Mem asserts Wait signal
Deriving the State Diagram and Datapath

Assume Synchronous Mealy Machine:
Transitions associated with arcs rather than states

Reset State (State 0)
and Instruction Fetch
Sequence

On Reset:
zero the PC
Mem Request unasserted
Mem asserts Wait signal

Instruction Fetch:
issue read request
4 cycle handshake on Wait signal

Note: No explicit mention of the
busses being used to implement
register transfers!
11.2.7 CS150 Newton/Pister

Deriving the State Diagram and Datapath

Operation Decode State

Four Way Next State Branch based on opcode bits

IR<15:14>=00

LD0, ST0, AD0, BR0

11.2.8 CS150 Newton/Pister

Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence
like IFetch, except that operand address comes from IR and data should be loaded into AC

IR<15:14>=00/
IR<13:0> \rightarrow MAR

LD0
Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence

like IFetch, except that operand address comes from IR and data should be loaded into AC

CS150 Newton/Pister
Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence
like IFetch, except that operand address comes from IR and data should be loaded into AC

Store Execution Sequence
Memory write sequence
Deriving the State Diagram and Datapath

Store Execution Sequence

Memory write sequence
Deriving the State Diagram and Datapath

Add Execution Sequence
Similar to Load sequence
Add MBR, AC rather than simply transfer MBR to AC
Deriving the State Diagram and Datapath

Add Execution Sequence

Similar to Load sequence
Add MBR, AC rather than simply transfer MBR to AC

Wait/ OD
IR<15:14>=10/
Wait/ IR<13:0> → MAR

Wait/ ADR
Wait/ MAR → Memory,
1 → Read/Write,
1 → Request

Wait/ ADR
Wait/ MAR → Memory,
1 → Read/Write,
1 → Request

Wait/ Mem → MBR

CS150 Newton/Pister 11.2.17
Deriving the State Diagram and Datapath

Add Execution Sequence

Similar to Load sequence
Add MBR, AC rather than simply transfer MBR to AC

Branch Execution Sequence
Deriving the State Diagram and Datapath

Branch Execution Sequence

Replace PC with Operand Address if AC < 0

Otherwise, do nothing

IR<15:14> = 1
IR<13:0> = PC
AC<15> = 0
AC<15> = 1

Simplify Wait Looping
Eliminate some Wait states

At this point, Wait must be asserted, so why loop on Wait?
Why loop on Wait when resync will take place at state IF0?
Deriving the State Diagram and Datapath

State Machine Inputs and Outputs so far:

**Inputs:**
- Reset
- Wait
- IR<15:14>
- AC<15>

**Outputs:**
- 0 → PC
- PC + 1 → PC
- PC → MAR
- MAR → Memory Address Bus
- Memory Data Bus → MBR
- MBR → Memory Data Bus
- MBR → IR
- MBR → AC
- AC → MBR
- AC + MBR → AC
- IR<13:0> → MAR
- IR<13:0> → PC
- 1 → Read/Write
- 0 → Read/Write
- 1 → Request

Processor Signal Flow

Memory

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td></td>
</tr>
<tr>
<td>Wait</td>
<td></td>
</tr>
<tr>
<td>IR&lt;15:14&gt;</td>
<td></td>
</tr>
<tr>
<td>AC&lt;15&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Mapping onto Datapath Control

Specification so far is independent of bussing strategy
Implied transfers:

This is the point-to-point connection scheme

Observe that instruction fetch and operand fetch take place at different times

This implies that IR, PC, and MAR transfers can be implemented by single bus (Address Bus)

Combine MBR, IR, ALU B, and AC connections (Memory Bus)
Combine ALU, AC, and MBR connections (Result Bus)

Three bus architecture:
AC + MBR $\rightarrow$ AC implemented in single state
Mapping onto Datapath Control

AC has two inputs, RBUS and MBUS
(Other registers except MBR have single input and output)

Dual ported configuration is more complex

Better idea: reuse existing paths were possible
MBR → AC transfer implemented by PASS B ALU operation

---

Mapping onto Datapath Control

Detailed implementation of register transfer operations
More detailed control operations are called microoperations

One register transfer operation = several microoperations

Some operations directly implemented by functional units:
e.g., ADD, Pass B, 0 → PC, PC + 1 → PC

Some operations require multiple control operations:
e.g., PC → MAR implemented as
PC → ABUS and ABUS → MAR
11.2.29
CS150 Newton/Pister

Mapping onto Datapath Control

Load Input
Tri-state Control

PC implemented by counter with COUNT and CLEAR inputs

11.2.30
CS150 Newton/Pister

Mapping onto Datapath Control
Timing of State Changes and Microoperations

Deferred til next clock edge
Takes place immediately
Deferred til next clock edge

MAR latches ABUS
# Mapping onto Datapath Control

Relationship between register transfer and microoperations:

<table>
<thead>
<tr>
<th>Register Transfer</th>
<th>Microoperations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 → PC</td>
<td>0 → PC (delayed);</td>
</tr>
<tr>
<td>PC + 1 → PC</td>
<td>PC + 1 → PC (delayed);</td>
</tr>
<tr>
<td>PC → MAR</td>
<td>PC → ABUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>ABUS → MAR (delayed);</td>
</tr>
<tr>
<td>MAR → Address Bus</td>
<td>MAR → Address Bus (immediate);</td>
</tr>
<tr>
<td>Data Bus → MBR</td>
<td>Data Bus → MBR (delayed);</td>
</tr>
<tr>
<td>MBR → Data Bus</td>
<td>MBR → Data Bus (immediate);</td>
</tr>
<tr>
<td>MBR → IR</td>
<td>MBR → ABUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>ABUS → IR (delayed);</td>
</tr>
<tr>
<td>MBR → AC</td>
<td>MBR → MBUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>MBUS → ALU B (immediate),</td>
</tr>
<tr>
<td></td>
<td>ALU PASS B (immediate),</td>
</tr>
<tr>
<td></td>
<td>ALU Result → RBUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>RBUS → AC (delayed);</td>
</tr>
</tbody>
</table>

Special microoperations for AC → ALU and ALU Result → RBUS not strictly necessary since these connections can be hardwired.
11.2.33

CS150 Newton/Pister

Mapping onto Datapath Control

Revised microoperation signal flow

5 inputs
make sure that Reset and Wait are synchronized

16 datapath control lines
2 memory control lines