Processor Controller Implementation

Acknowledgement- These slides are part of the slides by Prof. Randy Katz, Univ. of Berkley, selected for this course.
Controller Implementation

*Alternative Ways to Implement Processor FSMs*

- "Random Logic" based on Moore and Mealy Design
  *Classical* Finite State Machine Design

- Divide and Conquer Approach: Time-State Method
  Partition FSM into multiple communicating FSMs

- Exploit MSI Components: Jump Counters
  Counters, Multiplexors, Decoders

- Microprogramming: ROM-based methods
  Direct encoding of next states and outputs
Controller Implementation

Random Logic

Perhaps poor choice of terms for "classical" FSMs

Contrast with structured logic: PAL/PLA, FPGA, ROM

Could just as easily construct Moore and Mealy machines with these components
Finite State Machines for Simple CPUs

State Diagram and Datapath Derivation

Processor Specification:

Instruction Format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Op Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>LD</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>ST</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td>11</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:
Finite State Machines for Simple CPUs

*Deriving the State Diagram and Datapath*

First pass state diagram:
Controller Implementation

Moore Machine State Diagram

Note capture of MBR in these states

- **Reset**: 0 → PC
- **PC → MAR, PC + 1 → PC**
- **MAR → Mem, 1 → Read/Write, 1 → Request, Mem → MBR**
- **MBR → IR**
- **MBR → AC MBR + AC → AC**

- **LD0**: IR → MAR
- **LD1**: MAR → Mem, 1 → Read/Write, 1 → Request, Mem → MBR
- **LD2**: MBR → AC
- **ST0**: IR → MAR, AC → MBR
- **ST1**: MAR → Mem, 0 → Read/Write, 1 → Request, MBR → Mem
- **AD0**: MAR → Mem, 1 → Read/Write, 1 → Request, Mem → MBR
- **AD1**: MAR → Mem, 1 → Request, Mem → MBR
- **AD2**: MBR + AC → AC
- **BR0**: IR → PC
- **BR1**: IR → PC

No. 12-6
Finite State Machines for Simple CPUs

Mapping onto Datapath Operations

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
Finite State Machines for Simple CPUs

Mapping onto Datapath Operations

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), 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), ABUS → IR (delayed);</td>
</tr>
<tr>
<td>MBR → AC</td>
<td>MBR → MBUS (immediate), MBUS → ALU B (immediate), ALU PASS B (immediate), ALU Result → RBUS (immediate), RBUS → AC (delayed);</td>
</tr>
</tbody>
</table>
Finite State Machines for Simple CPUs

Mapping onto Datapath Operations

Relationship between register transfer and microoperations:

<table>
<thead>
<tr>
<th>Register Transfer</th>
<th>Microoperations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC → MBR</td>
<td>AC → RBUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>RBUS → MBR (delayed);</td>
</tr>
<tr>
<td>AC + MBR → AC</td>
<td>AC → ALU A (immediate),</td>
</tr>
<tr>
<td></td>
<td>MBR → MBUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>MBUS → ALU B (immediate),</td>
</tr>
<tr>
<td></td>
<td>ALU ADD (immediate),</td>
</tr>
<tr>
<td></td>
<td>ALU Result → RBUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>RBUS → AC (delayed);</td>
</tr>
<tr>
<td>IR&lt;13:0&gt; → MAR</td>
<td>IR → ABUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>ABUS → IR (delayed);</td>
</tr>
<tr>
<td>IR&lt;13:0&gt; → PC</td>
<td>IR → ABUS (immediate),</td>
</tr>
<tr>
<td></td>
<td>ABUS → PC (delayed);</td>
</tr>
<tr>
<td>1 → Read/Write</td>
<td>Read (immediate);</td>
</tr>
<tr>
<td>0 → Read/Write</td>
<td>Write (immediate);</td>
</tr>
<tr>
<td>1 → Request</td>
<td>Request (immediate);</td>
</tr>
</tbody>
</table>

Special microoperations for AC → ALU and ALU Result → RBUS not strictly necessary since these connections can be hardwired
Finite State Machines for Simple CPUs

Mapping onto Datapath Operations

Revised microoperation signal flow

5 inputs
make sure that Reset and Wait are synchronized

16 datapath control lines
2 memory control lines
Controller Implementation

Moore Machine Block Diagram

- 16 states, 4 bit state register
- Next State Logic: 9 Inputs, 4 Outputs
- Output Logic: 4 Inputs, 18 Outputs

These can be implemented via ROM or PAL/PLA

- Next State: 512 x 4 bit ROM
- Output: 16 x 18 bit ROM

No. 12-11
Controller Implementation

Moore Machine State Diagram

Note capture of MBR in these states

- **Reset**: 0 → PC
- **PC → MAR, PC + 1 → PC**
- **MAR → Mem, 1 → Read/Write, 1 → Request, Mem → MBR**
- **MBR → IR**
- **IR → MAR, AC → MBR**
- **IR → MAR**
- **MAR → Mem, 1 → Read/Write, 1 → Request, Mem → MBR**
- **MBR + AC → AC**
# Controller Implementation

## Moore Machine Next State Table

<table>
<thead>
<tr>
<th>Reset</th>
<th>Wait</th>
<th>IR&lt;15&gt;</th>
<th>IR&lt;14&gt;</th>
<th>AC&lt;15&gt;</th>
<th>Current State</th>
<th>Next State</th>
<th>Register Transfer Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>RES (0000)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>RES (0000)</td>
<td>IF0 (0001) 0 → PC</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF0 (0001)</td>
<td>IF1 (0001)</td>
<td>PC → MAR, PC + 1 → PC</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF1 (0010)</td>
<td>IF1 (0010)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF1 (0010)</td>
<td>IF2 (0011)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF2 (0011)</td>
<td>IF2 (0011)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF2 (0011)</td>
<td>IF3 (0100)</td>
<td>MAR → Mem, Read, Request, Mem → MBR</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF3 (0100)</td>
<td>IF3 (0100)</td>
<td>MBR → IR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF3 (0100)</td>
<td>OD (0101)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>OD (0101)</td>
<td>LD0 (0110)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>OD (0101)</td>
<td>ST0 (1001)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>OD (0101)</td>
<td>AD0 (1011)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>OD (0101)</td>
<td>BR0 (1110)</td>
<td></td>
</tr>
</tbody>
</table>

No. 12-13
## Controller Implementation

### Moore Machine Next State Table

<table>
<thead>
<tr>
<th>Reset</th>
<th>Wait</th>
<th>IR&lt;15&gt;</th>
<th>IR&lt;14&gt;</th>
<th>AC&lt;15&gt;</th>
<th>Current State</th>
<th>Next State</th>
<th>Register Transfer Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>LD0 (0110)</td>
<td>LD1 (0111)</td>
<td>IR → MAR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>LD1 (0111)</td>
<td>LD1 (0111)</td>
<td>MAR → Mem, Read,</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>LD1 (0111)</td>
<td>LD2 (1000)</td>
<td>Request, Mem → MBR</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>LD2 (1000)</td>
<td>IF0 (0001)</td>
<td>MBR → AC</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ST0 (1001)</td>
<td>ST1 (1010)</td>
<td>IR → MAR, AC → MBR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ST1 (1010)</td>
<td>ST1 (1010)</td>
<td>MAR → Mem, Write,</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ST1 (1010)</td>
<td>IF0 (0001)</td>
<td>Request, MBR → Mem</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AD0 (1011)</td>
<td>AD1 (1100)</td>
<td>IR → MAR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AD1 (1100)</td>
<td>AD1 (1100)</td>
<td>MAR → Mem, Read,</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AD1 (1100)</td>
<td>AD2 (1101)</td>
<td>Request, Mem → MBR</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AD2 (1101)</td>
<td>IF0 (0001)</td>
<td>MBR + AC → AC</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>BR0 (1110)</td>
<td>IF0 (0001)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>BR0 (1110)</td>
<td>BR1 (1111)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>BR1 (1111)</td>
<td>IF0 (0001)</td>
<td>IR → PC</td>
</tr>
</tbody>
</table>

No. 12-14
Controller Implementation

Moore Machine State Transition Table

Observations:

• Extensive use of Don't Cares

• Inputs used only in a small number of states
e.g., AC<15> examined only in BR0 state
IR<15:14> examined only in OD state

• Some outputs always asserted in a group

• ROM-based implementations cannot take advantage of don't cares

• However, ROM-based implementation can skip state assignment step
Controller Implementation

Moore Machine Implementation

Assume PAL/PLA implementation style

First idea:
run ESPRESSO with naive state assignment

21 product terms

Compare with 512 product terms in ROM implementation!
**Controller Implementation**

*Moore Machine Implementation*

NOVA assignment does better

**NOVA State Assignment SUMMARY**

<table>
<thead>
<tr>
<th>States</th>
<th>State Code</th>
<th>Best code</th>
</tr>
</thead>
<tbody>
<tr>
<td>states[0]:IF0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>states[1]:IF1</td>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>states[2]:IF2</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>states[3]:IF3</td>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>states[4]:OD</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>states[5]:LD0</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>states[6]:LD1</td>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>states[7]:LD2</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>states[8]:ST0</td>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>states[9]:ST1</td>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>states[10]:AD0</td>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>states[11]:AD1</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>states[12]:AD2</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>states[13]:BR0</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>states[14]:BR1</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>states[15]:RES</td>
<td>1110</td>
<td>1110</td>
</tr>
</tbody>
</table>

onehot_products = 22

best_products = 18

best_size = 414

18 product terms improves on 21!
Controller Implementation

Synchronization of signals for implementation as a Mealy Machine

Implications for Processor FSM Already Derived

Consider inputs: Reset, Wait, IR<15:14>, AC<15>

Latter two already come from registers, and are sync'd to clock

Possible to load IR with new instruction in one state and perform multiway branch on opcode in next state

Best solution for Reset and Wait: synchronized inputs

Place D flipflops between these external signals and the control inputs to the processor FSM

Sync'd versions of Reset and Wait delayed by one clock cycle
Controller Implementation

Time-State (Divide & Conquer)

Overview

Classical Approach: Monolithic Implementations

Alternative "Divide & Conquer" Approach:

Decompose FSM into several simpler communicating FSMs

- Time state FSM (e.g., IFetch, Decode, Execute)
- Instruction state FSM (e.g., LD, ST, ADD, BRN)
- Condition state FSM (e.g., AC < 0, AC ≥ 0)
Controller Implementation

_Time-State (Divide & Conquer)_

Time State FSM

Most instructions follow same basic sequence

Differ only in detailed execution sequence

Time State FSM can be parameterized by opcode and AC states

Instruction State: stored in IR<15:14>

- IR = 00
- IR = 01
- IR = 10
- IR = 11

LD  ST  ADD  BRN

Condition State: stored in AC<15>

- AC < 0
- AC < 15> = 0
- AC < 15> = 1

0  0/
Controller Implementation

Time State (Divide & Conquer)

Generation of Microoperations

0 → PC: Reset
PC + 1 → PC: T0
PC → MAR: T0
MAR → Memory Address Bus: T2 + T6 • (LD + ST + ADD)
Memory Data Bus → MBR: T2 + T6 • (LD + ADD)
MBR → Memory Data Bus: T6 • ST
MBR → IR: T4
MBR → AC: T7 • LD
AC → MBR: T5 • ST
AC + MBR → AC: T7 • ADD
IR<13:0> → MAR: T5 • (LD + ST + ADD)
IR<13:0> → PC: T6 • BRN
1 → Read/Write: T2 + T6 • (LD + ADD)
0 → Read/Write: T6 • ST
1 → Request: T2 + T6 • (LD + ST + ADD)
Controller Implementation

Branch Sequencers

Concept

Implement Next State Logic via ROM

Address ROM with current state and inputs

Problem: ROM doubles in size for each additional input

Branch Sequencer:
Next State stored in ROM
Each state limited to small number of next states
Always a power of 2

Observe: only a small set of inputs are examined in any state
Controller Implementation

Branch Sequencers

4 Way Branch Sequencer

Current State selects two inputs to form part of ROM address

These select one of four possible next states (and output sets)

Every state has exactly four possible next states
Controller Implementation

Branch Sequencer

Processor CPU Design Example

Alpha, Beta multiplexer input setup
### Branch Sequencers

#### Example Processor FSM

<table>
<thead>
<tr>
<th>ROM ADDRESS (Reset, Current State, a, b)</th>
<th>ROM CONTENTS</th>
<th>Next State</th>
<th>Register Transfer Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>__RES 0 0000 X X</td>
<td></td>
<td>0001 (IF0)</td>
<td>PC → MAR, PC + 1 → PC</td>
</tr>
<tr>
<td>IF0 0 0001 0 0</td>
<td></td>
<td>0001 (IF0)</td>
<td></td>
</tr>
<tr>
<td>IF0 0 0001 1 1</td>
<td></td>
<td>0010 (IF1)</td>
<td>MAR → Mem, Read, Request</td>
</tr>
<tr>
<td>IF1 0 0010 0 0</td>
<td></td>
<td>0011 (IF2)</td>
<td>MAR → Mem, Read, Request</td>
</tr>
<tr>
<td>IF1 0 0010 1 1</td>
<td></td>
<td>0010 (IF1)</td>
<td>Mem → MBR</td>
</tr>
<tr>
<td>IF2 0 0011 0 0</td>
<td></td>
<td>0011 (IF2)</td>
<td></td>
</tr>
<tr>
<td>IF2 0 0011 1 1</td>
<td></td>
<td>0100 (OD)</td>
<td>MBR → IR</td>
</tr>
<tr>
<td>OD 0 0100 0 0</td>
<td></td>
<td>0101 (LD0)</td>
<td>IR → MAR</td>
</tr>
<tr>
<td>OD 0 0100 0 1</td>
<td></td>
<td>1000 (ST0)</td>
<td>IR → MAR, AC → MBR</td>
</tr>
<tr>
<td>OD 0 0100 1 0</td>
<td></td>
<td>1001 (AD0)</td>
<td>IR → MAR</td>
</tr>
<tr>
<td>OD 0 0100 1 1</td>
<td></td>
<td>1101 (BR0)</td>
<td>IR → MAR</td>
</tr>
</tbody>
</table>
# Branch Sequencers

## Example Processor FSM

<table>
<thead>
<tr>
<th>ROM ADDRESS</th>
<th>ROM CONTENTS</th>
<th>Register Transfer Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Reset, Current State, a, b)</td>
<td>Next State</td>
<td>MAR $\rightarrow$ Mem, Read, Request</td>
</tr>
<tr>
<td>LD0 0 0101 X X</td>
<td>0110 (LD1)</td>
<td>MAR $\rightarrow$ Mem, Read, Request</td>
</tr>
<tr>
<td>LD1 0 0110 0 0</td>
<td>0111 (LD2)</td>
<td>Mem $\rightarrow$ MBR</td>
</tr>
<tr>
<td>LD2 0 0111 X X</td>
<td>0000 (RES)</td>
<td>MBR $\rightarrow$ AC</td>
</tr>
<tr>
<td>ST0 0 1000 X X</td>
<td>1001 (ST1)</td>
<td>MAR $\rightarrow$ Mem, Write, Request, MBR $\rightarrow$ Mem</td>
</tr>
<tr>
<td>ST1 0 1001 0 0</td>
<td>0000 (RES)</td>
<td>MAR $\rightarrow$ Mem, Write, Request, MBR $\rightarrow$ Mem</td>
</tr>
<tr>
<td>AD0 0 1010 X X</td>
<td>1011 (AD1)</td>
<td>MAR $\rightarrow$ Mem, Read, Request</td>
</tr>
<tr>
<td>AD1 0 1011 0 0</td>
<td>1100 (AD2)</td>
<td></td>
</tr>
<tr>
<td>AD2 0 1100 X X</td>
<td>0000 (RES)</td>
<td>MBR + AC $\rightarrow$ AC</td>
</tr>
<tr>
<td>BR0 0 1101 0 0</td>
<td>0000 (RES)</td>
<td></td>
</tr>
<tr>
<td>BR0 0 1101 1 1</td>
<td>0000 (RES)</td>
<td>IR $\rightarrow$ PC</td>
</tr>
</tbody>
</table>
Controller Implementation

Branch Sequencers

Alternative Horizontal Implementation

Input MUX controlled by encoded signals, not state
Much fewer inputs than unique states!
In example FSM, input MUX can be 2:1!

Adding length to ROM word saves on bits vs. doubling words
Vertical format: \( (14 + 4) \times 64 = 1152 \) ROM bits
Horizontal format: \( (14 + 4 \times 4 + 2) \times 16 = 512 \) ROM bits
Controller Implementation

Microprogramming

How to organize the control signals?

Implement control signals by storing 1's and 0's in a ROM

*Horizontal vs. vertical microprogramming*

Horizontal: 1 ROM output for each control signal

Vertical: encoded control signals in ROM, decoded externally

some mutually exclusive signals can be combined

helps reduce ROM length
Controller Implementation

Microprogramming

Register Transfer/Microoperations

14 Register Transfer operations become 22 Microoperations:

PC → ABUS       ABUS → MAR
IR → ABUS        Data Bus → MBR
MBR → ABUS       RBUS → MBR
RBUS → AC        MBR → MBUS
AC → ALU A       0 → PC
MBUS → ALU B     PC + 1 → PC
ALU ADD          ABUS → PC
ALU PASS B       Read/Write
MAR → Address Bus Request
MBR → Data Bus   AC → RBUS
ABUS → IR        ALU Result → RBUS
Controller Implementation

Horizontal Microprogramming

Horizontal Branch Sequencer

\( \alpha, \beta \) Mux bits
4 x 4 Next State bits
22 Control operation bits

40 bits total
### Controller Implementation

#### Horizontal Microprogramming

#### Moore Processor ROM

**Current State (Address)** | **αmux** | **βmux** | **A0** | **A1** | **A2** | **A3** | **Next States** | **PC** | **ABUS** | **IR** | **ABUS** | **AC** | **MBR** | **ALU** | **ABUS** | **MBR** | **ALU** | **ABUS** | **MBR** | **ALU** | **Read/Write** | **Request** | **AC** | **RBUS** | **ALU Result** | **RBUS**
RES (0000) | 0 | 0 | 0001 | 0001 | 0001 | 0001 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
IF0 (0001) | 0 | 0 | 0010 | 0010 | 0010 | 0010 | 0010 | 0100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
IF1 (0010) | 0 | 0 | 0010 | 0010 | 0010 | 0010 | 0010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
IF2 (0011) | 0 | 0 | 0100 | 0100 | 0100 | 0100 | 0100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
IF3 (0100) | 0 | 0 | 0100 | 0100 | 0100 | 0100 | 0100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
OD (0101) | 1 | 1 | 0110 | 1001 | 1011 | 1110 | 0110 | 0100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
LD0 (0110) | 0 | 0 | 0111 | 0111 | 0111 | 0111 | 0111 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
LD1 (0111) | 0 | 0 | 1000 | 1000 | 1000 | 1000 | 1000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
LD2 (1000) | 0 | 0 | 0001 | 0001 | 0001 | 0001 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
ST0 (1001) | 0 | 0 | 1010 | 1010 | 1010 | 1010 | 1010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
ST1 (1010) | 0 | 0 | 0001 | 0001 | 1010 | 1010 | 1010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
AD0 (1011) | 0 | 0 | 1100 | 1100 | 1100 | 1100 | 1100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
AD1 (1100) | 0 | 0 | 1101 | 1101 | 1101 | 1101 | 1101 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
AD2 (1101) | 0 | 0 | 0001 | 0001 | 0001 | 0001 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
BR0 (1110) | 0 | 1 | 0001 | 1111 | 0001 | 1111 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
BR1 (1111) | 0 | 0 | 0001 | 0001 | 0001 | 0001 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

**Alpha inputs:** 0 = Wait, 1 = IR<15>  
**Beta inputs:** 0 = AC<15>, 1 = IR<14>  

No. 12-31
Controller Implementation

Horizontal Microprogramming

Advantages:
most flexibility -- complete parallel access to datapath control points

Disadvantages:
very long control words -- 100+ bits for real processors

NOTE: Not all microoperation combinations make sense!

Output Encodings:
  Group mutually exclusive signals
  Use external logic to decode

Example:
  0 \rightarrow \text{PC, PC + 1} \rightarrow \text{PC, ABUS} \rightarrow \text{PC mutually exclusive}

  Save ROM bit with external 2:4 Decoder
Controller Implementation

Horizontal Microprogramming

Partially Encoded Control Outputs

ALU ADD
ALU PASS B
MAR → Address Bus
MBR → Data Bus
ABUS → MAR
RBUS → MBR
Read/Write
Request
AC → RBUS

RBUS → AC
AC → ALU A
MBUS → ALU B
MBR → MBUS
ALU Result → RBUS
MBR → ABUS

ABUS → IR

0 → PC
PC + 1 → PC
ABUS → PC

PC → ABUS
IR → ABUS
Data Bus → MBR

No. 12-33
Controller Implementation

Vertical Microprogramming

More extensive encoding to reduce ROM word length

Typically use multiple microword formats:

- horizontal microcode -- next state + control bits in same word
- separate formats for control outputs and "branch jumps"
- may require several microwords in a sequence to implement same function as single horizontal word
- in the extreme, very much like assembly language programming
Controller Implementation

Vertical Microprogramming

**Branch Jump**
Compare indicated signal to 0 or 1

<table>
<thead>
<tr>
<th>Type</th>
<th>Condition Select</th>
<th>Condition Compare</th>
<th>Next Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>00 = Wait</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>01 = AC&lt;15&gt;</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>10 = IR&lt;15&gt;</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>11 = IR&lt;14&gt;</td>
</tr>
</tbody>
</table>

**Register Transfer**
Source, Destination, Operation

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000: NO OP</td>
<td>000: NO OP</td>
<td>000: NO OP</td>
</tr>
<tr>
<td>001: PC → ABUS</td>
<td>001: RBUS → AC</td>
<td>001: ALU ADD</td>
</tr>
<tr>
<td>010: IR → ABUS</td>
<td>010: MBUS → IR</td>
<td>010: ALU PASS B</td>
</tr>
<tr>
<td>011: MBR → MBUS</td>
<td>011: ABUS → MAR</td>
<td>011: 0 → PC</td>
</tr>
<tr>
<td>100: MAR → M</td>
<td>100: M → MBR</td>
<td>100: PC + 1 → PC</td>
</tr>
<tr>
<td>101: AC → RBUS</td>
<td>101: RBUS → MBR</td>
<td>101: Read</td>
</tr>
<tr>
<td>110: ALU Res → RBUS</td>
<td>110: ABUS → PC</td>
<td>110: Write</td>
</tr>
<tr>
<td>111: MBR → M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No. 12-35
Controller Implementation

*Vertical Microprogramming*

**ROM Contents**

<table>
<thead>
<tr>
<th>ROM ADDRESS</th>
<th>SYMBOLIC CONTENTS</th>
<th>BINARY CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>RES RT PC → MAR, PC +1 → PC</td>
<td>0 001 011 100</td>
</tr>
<tr>
<td>000001</td>
<td>IF0 RT MAR → M, Read</td>
<td>0 100 000 101</td>
</tr>
<tr>
<td>000010</td>
<td>BJ Wait=0, IF0</td>
<td>1 000 000 001</td>
</tr>
<tr>
<td>000011</td>
<td>IF1 RT MAR → M, M → MBR, Read</td>
<td>0 100 100 101</td>
</tr>
<tr>
<td>000100</td>
<td>BJ Wait=1, IF1</td>
<td>1 001 000 011</td>
</tr>
<tr>
<td>000101</td>
<td>IF2 RT MBR → IR</td>
<td>0 011 010 000</td>
</tr>
<tr>
<td>000110</td>
<td>BJ Wait=0, IF2</td>
<td>1 000 000 101</td>
</tr>
<tr>
<td>000111</td>
<td>RT IR → MAR</td>
<td>0 010 011 000</td>
</tr>
<tr>
<td>001000</td>
<td>OD BJ IR&lt;15&gt;=1, OD1</td>
<td>1 101 010 101</td>
</tr>
<tr>
<td>001001</td>
<td>BJ IR&lt;14&gt;=1, ST0</td>
<td>1 111 010 000</td>
</tr>
<tr>
<td>001010</td>
<td>LD0 RT MAR → M, Read</td>
<td>0 100 000 101</td>
</tr>
<tr>
<td>001011</td>
<td>LD1 RT MAR → M, M → MBR, Read</td>
<td>0 100 100 101</td>
</tr>
<tr>
<td>001100</td>
<td>BJ Wait=1, LD1</td>
<td>1 001 001 011</td>
</tr>
<tr>
<td>001101</td>
<td>LD2 RT MBR → AC</td>
<td>0 110 001 010</td>
</tr>
<tr>
<td>001110</td>
<td>BJ Wait=0, RES</td>
<td>1 000 000 000</td>
</tr>
<tr>
<td>001111</td>
<td>BJ Wait=1, RES</td>
<td>1 001 000 000</td>
</tr>
</tbody>
</table>
Controller Implementation

Vertical Microprogramming

ROM Contents

<table>
<thead>
<tr>
<th>ROM ADDRESS</th>
<th>SYMBOLIC CONTENTS</th>
<th>BINARY CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>ST0 RT AC → MBR</td>
<td>0 101 101 000</td>
</tr>
<tr>
<td>010001</td>
<td>RT MAR → M, MBR → M, Write</td>
<td>0 100 111 110</td>
</tr>
<tr>
<td>010010</td>
<td>ST1 RT MAR → M, MBR → M, Write</td>
<td>0 100 111 110</td>
</tr>
<tr>
<td>010011</td>
<td>BJ Wait=0, RES</td>
<td>1 000 000 000</td>
</tr>
<tr>
<td>010100</td>
<td>BJ Wait=1, ST1</td>
<td>1 001 010 010</td>
</tr>
<tr>
<td>010101</td>
<td>OD1 BJ IR&lt;14&gt;=1, BR0</td>
<td>1 111 011 101</td>
</tr>
<tr>
<td>010110</td>
<td>AD0 RT MAR → M, Read</td>
<td>0 100 000 101</td>
</tr>
<tr>
<td>010111</td>
<td>AD1 RT MAR → M, M → MBR, Read</td>
<td>0 100 100 101</td>
</tr>
<tr>
<td>011000</td>
<td>BJ Wait=1, AD1</td>
<td>1 001 010 111</td>
</tr>
<tr>
<td>011001</td>
<td>AD2 RT AC + MBR → AC</td>
<td>0 110 001 001</td>
</tr>
<tr>
<td>011010</td>
<td>BJ Wait=0, RES</td>
<td>1 000 000 000</td>
</tr>
<tr>
<td>011011</td>
<td>BJ Wait=1, RES</td>
<td>1 000 000 000</td>
</tr>
<tr>
<td>011100</td>
<td>BR0 BJ AC&lt;15&gt;=0, RES</td>
<td>1 010 000 000</td>
</tr>
<tr>
<td>011101</td>
<td>RT IR → PC</td>
<td>0 010 110 000</td>
</tr>
<tr>
<td>011110</td>
<td>BJ AC&lt;15&gt;=1, RES</td>
<td>1 011 000 000</td>
</tr>
</tbody>
</table>

31 words x 10 ROM bits = 310 bits total versus 16 x 38 = 608 bits horizontal

No. 12-37
Controller Implementation

Vertical Microprogramming

Controller Block Diagram

ROM

<table>
<thead>
<tr>
<th>T</th>
<th>SRC</th>
<th>DST</th>
<th>OP</th>
</tr>
</thead>
</table>

3:8 DEC 0
ALU ADD
ALU PASS B

3:8 DEC 1
PC + 1 → PC
Read

3:8 DEC 2
Write

3:8 DEC 3
RBUS → AC

3:8 DEC 4
ABUS → IR
M → MBR

3:8 DEC 5
RBUS → MBR
ABUS → PC

3:8 DEC 6
MBR → M

3:8 DEC 7
PC → ABUS
IR → ABUS
M → ABUS
MAR → M
AC → RBUS
ALU Res → RBUS

Cond Logic
LD
CLR
μPC
CNT

Cond Logic

Reset
Clk

Wait
AC<15>
IR<15>
IR<14>

Read/Write Request

0 → PC

Controller Implementation
Vertical Microprogramming
Condition Logic

Condition Selector

Condition Comparator

Wait
AC<15>
IR<15>
IR<14>

4:1 MUX

Microinstruction Type

Microinstruction Type

LD

CNT

No. 12-39
Controller Implementation

Vertical Microprogramming
Writeable Control Store

Part of control store addresses map into RAM

Allows assembly language programmer to implement own instructions

Extend "native" instruction set with application specific instructions

Requires considerable sophistication to write microcode

Not a popular approach with today's RISC machines

Make the native instruction set simple and fast

Write "higher level" functions as assembly language sequences
Controller Implementation

Chapter Summary

- Control Unit Organization
  - Register transfer operation
  - Classical Moore and Mealy machines
  - Time State Approach
  - Jump Counter
  - Branch Sequencers
  - Horizontal and Vertical Microprogramming