Micro–programmable State Machine Design
Dr. Shoab A Khan
Microprogrammed Control Unit

- Substitution of combinational cloud of FSM with Programmable Memory (PM)
- During each clock period, the appropriate level control signals are read from PM instead of being generated by combination logic.
PM–based State Machine Implementation

Mealy machine: ROM address bits A0–A5 are function of Input x1 and x2 and current states S0–S3
PM–based Micro–programmable state machine
Example Design

![Diagram of a simple state machine with inputs, outputs, and state transitions. The diagram shows inputs labeled as 'input', 'addr', and 'current state', with outputs labeled as 'data', 'cntr', and 'next state'. The diagram also includes a 'PM' block with connections to and from the state machine.]
Micro–programmed State Machine Implementation

(a) State Diagram

(b) Program Memory

Current State | Input | Next State
---|---|---
00 | 0 | 00
00 | 1 | 00
01 | 0 | 01
01 | 1 | 01
10 | 0 | 10
10 | 1 | 10
11 | 0 | 11
11 | 1 | 11

Program Memory

<table>
<thead>
<tr>
<th>addr</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Moore Machine

(a) (b)
Design Example: 4 Entry FIFO

1. Design a first-in, first-out (FIFO) queue that consists of four registers R0, R1, R2, and R3
2. Write and Delete are the two operations on the queue
3. *Write* moves data from the *fifo_in* to R0 that is the tail of the queue
4. *Delete* deletes the first entry at the head of the queue
5. The head of the queue is available on the *fifo_out*
6. Writing into a full queue or deletion from an empty queue causes an *ERROR* condition
7. Assertion of *Write* and *Delete* at the same time also causes an *ERROR* condition
Design consists of datapath and controller

Datapath
- Four registers
  - Shift registers
- One 4:1 MUX

Controller
- Input signals
  - Write
  - Delete
- Output signals
  - out_sel
  - write_en
  - Error
Controller

- FSM based design
- Mealy machine
- Five states
  - Idle
  - One entry
  - Two entries
  - Three entries
  - Full
FIFO/LIFO

(a)

(b)
Counter-based FSM: Bit serial adder
Counter–based State Machine Implementation

Machine useful only in generating sequence of control Signals without any input
Counter-based State Machine Implementation

Microprogram Memory

- Counter
- Rst_n
- Clk
- Data
- Control signals

Microprogram Memory

- Addr
- N
- M
- Control signals
Modifications to Counter–based Microcontrollers 1

- Mechanism for changing count sequence
- Begin another sequence under control of micro–program memory
Counter-based State Machine Implementation cont.

A new branch address is loaded in the counter if load is asserted.
Loadable Counter for *Goto* capability
A Generic Computing Architecture
Example: Complete Design
Modifications to Control–based Microcontrollers 2

- Introduction of decision–making capabilities
- to microcontroller implementation
  - Incorporating the branch address in the instruction
  - The counter loads the branch address if the signal load is asserted
Counter–based ASM with Conditional Branching

When \textit{CONDITION} = 1 counter is reset with \textit{branch address} value

**Diagram:**
- **Counter**
  - \text{RESET}
  - \text{SYSCLK}
  - \text{CONDITION}
  - \text{BRANCH VECTOR}
  - \text{CLR}
  - \text{LOAD}
  - \text{Q0}
  - \text{Q1}
  - \text{Qk}
  - \text{Ak}

**Microprogram Memory:**
- \(2^{K+1}\) Words of \(m+1\) bits
- \(D_{m-0}\)

**Branch Address:**
- \text{BRANCH ADDRESS}
- \text{LOAD}
- \text{OTHER}

**To Architecture**
Blocking Condition Testing

- **Conditional execution**
  - 2-bit control allowing on selection of two conditional inputs and two signals true and false
  - The conditions come from the status register in the data path
  - Example:
    - zero and positive flags are used for providing conditional execution
    - if \( r1 > r2 \) jump label3; // compute \( r1 - r2 \) jump if result is positive
    - if \( r1 == r2 \) jump label1; // compute \( r1 - r2 \), jump if result is zero
    - jump label 2 // unconditional jump
Counter-based ASM with Conditional Branching cont.

Microprogram Memory

2^{k+1} Words of m+1 bits

Branch Address

Load Sel

Other

Condition

False

Cond 0

Cond 1

True

Sel

 CLR

 Q0

 A0

 Q1

 A1

 Qk

 Ak

 D_{m-0}

 From Architecture

 To Architecture

Cond0, Cond1

RESET

CLK

Condition

Microprogram Memory

Branch Address

Load Sel

Other
Conditional Branching Capability

![Diagram of conditional branching]

- Counter
- Microprogram Memory
- Load
- Select
- Branch Address
- To Datapath
- Reset
- Clock

Conditional Branching Capability

- Conditional branch on the basis of a condition code (cond_0, cond_1)
- Microprogram memory location is loaded into the counter
- Counter value is used to select an address from the microprogram memory
- Branch address is loaded into the datapath based on the select signal
<table>
<thead>
<tr>
<th>sel</th>
<th>load</th>
<th>Conditional load</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>FALSE</td>
<td>(normal execution: never load branch address)</td>
</tr>
<tr>
<td>01</td>
<td>COND0</td>
<td>(load branch address if COND0 is TRUE)</td>
</tr>
<tr>
<td>10</td>
<td>COND1</td>
<td>(load branch address if COND1 is TRUE)</td>
</tr>
<tr>
<td>11</td>
<td>TRUE</td>
<td>(unconditional jump: always load branch address)</td>
</tr>
</tbody>
</table>
Pipelined Register

- Often counters are replaced with an ALU based program counter register.
- The critical path of the design is long as it goes from the counter to ROM to architecture to functional units generating COND0 and COND1 to conditional MUX.
- The critical path can be broken by inserting a pipeline register in the design.
Counter–based ASM with Pipeline Register

Counter

Microprogram Memory

2^{k+1} Words of m+1 bits

Branch Address

Load Sel

Other

_RESET_

_SYSCLK_

_CONDITION_

_FALSE_, _COND0_, _COND1_

_TRUE_

_MUX_

_SELECT_

_Load_

_Load Sel_

_Pipeline Registered_

_To Architecture_

_Load_

_Full Adder_

_k+1 bits_
Micro PC Based Controller

Micro PC

Microprogram Memory

cond
sel
branch
addr
cntr
signals

0
1
cond 0
cond 1
to datapath
from datapath
next addr mux
addr bus
data bus
cond sel
branch addr
cntr signals
Addition of Parity bit
Controller with data path
Instruction word design

(a) 
<table>
<thead>
<tr>
<th>cond sel</th>
<th>parity</th>
<th>label</th>
<th>out sel</th>
<th>src 1</th>
<th>src 2</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

(b) 
<table>
<thead>
<tr>
<th>type</th>
<th>sel</th>
<th>cond sel</th>
<th>parity</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>out sel</th>
<th>src 1</th>
<th>src 2</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Conditional Branching

- Conditional execution
  - 2-bit control allowing on selection of two conditional inputs and two signals true and false.
  - A parity bit for inverse selection of the condition

Example:
- zero and positive flags are used for providing conditional execution
- Inverse selection is provided by parity bit
- if (r1 > r2) jump label3; // compute r1–r2 jump if result is positive
- if (r1 < r2) jump label3; // compute r1–r2 jump if result is negative (positive flag 0 and polarity bit is set)
- if (r1 = r2) jump label1; // compute r1–r2, jump if result is zero
- if (r1 != r2) jump label1; // compute r1–r2, jump if result is not zero (zero flag is 0 and polarity bit is set)
- jump label 2 // unconditional jump
micro PC–based ASM with Condition Multiplexer
Subroutine Execution

- Temporary storage location added for the copy of contents of mPC register
  - Register referred to as Subroutine Return Address Register
- Address kept in register allows on returning to next address of microprogram execution after complete subroutine call
Register-based Controller with Subroutine Capability

- Microprogram Memory
- cond
- sel
- branch
- addr
- cntr
- signals
- branch addr 0
- branch addr 1
- cond 0
- cond 1
- next addr
to datapath
from datapath
- addr bus
- data bus
- parity
- bit
- branch addr
- cntr signals
- sub retrn addr reg
- micro PC
- next addr sel
- +
- 1
- next addr mux
## Design Problem

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>$ACC = R_i + c$</td>
<td>Add a constant into $R_i$</td>
</tr>
<tr>
<td>0001</td>
<td>$ACC = \pm R_i \pm R_j$</td>
<td>Signed addition of two registers</td>
</tr>
<tr>
<td>0010</td>
<td>$ACC = \pm R_i \pm R_j \pm R_k$</td>
<td>Signed addition of three registers</td>
</tr>
<tr>
<td>0011</td>
<td>$ACC = R_i + R_j + R_k + c$</td>
<td>Addition of three registers and a constant</td>
</tr>
<tr>
<td>0100</td>
<td>$ACC = R_i \times R_j$</td>
<td>Multiplication of 8 LSBs of two registers</td>
</tr>
<tr>
<td>0101</td>
<td>Load $R_i$</td>
<td>Load register into accumulator</td>
</tr>
<tr>
<td>0110</td>
<td>Store $R_i$</td>
<td>Store the contents of the accumulator to register</td>
</tr>
<tr>
<td>0111</td>
<td>Load $n$</td>
<td>Load an integer into accumulator</td>
</tr>
<tr>
<td>1000</td>
<td>Branch if Acc = 0</td>
<td>Branch if accumulator is zero</td>
</tr>
<tr>
<td>1001</td>
<td>Branch if Acc &lt; 0</td>
<td>Branch if accumulator is negative</td>
</tr>
<tr>
<td>1010</td>
<td>Always Branch</td>
<td>Unconditional branch</td>
</tr>
<tr>
<td>1011</td>
<td>Call Subroutine</td>
<td>Jump to subroutine address</td>
</tr>
<tr>
<td>1100</td>
<td>Return</td>
<td>Return from a subroutine call</td>
</tr>
<tr>
<td>1101</td>
<td>$Acc = R_i \lll n$</td>
<td>Right or left shift a register by 16, result in Accumulator</td>
</tr>
</tbody>
</table>
Arithmetic Insts

### Arithmetic Instructions

<table>
<thead>
<tr>
<th>X * X X X</th>
<th>X X X</th>
<th>X X</th>
<th>X X</th>
<th>X X</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4-bit opcode)</td>
<td>(Sign bits for the operands)</td>
<td>(Reg 1)</td>
<td>(Reg 2)</td>
<td>(Reg 3)</td>
</tr>
</tbody>
</table>
Branch/Load Integer Inst

\[
\begin{array}{c|c}
\text{X X X X X} & \text{X X X X X X X X X X X} \\
\end{array}
\]

(4-bit opcode) (9-bit Address/Integer)
Storing ACC in a register

\[
\begin{array}{cccc|cc|c}
\end{array}
\]

(4-bit opcode) \hspace{1cm} (Reg. No.)
Nested Subroutine Nested Subroutine Execution

- Subroutine Return Address Register replaced by Subroutine Return Address Memory of stack structure
  - Possibility to store more than one return address – necessity in implementing nested subroutine calls
- Up/down counter (TOP OF STACK) added for stack pointer manipulations
Micro-PC Controller with Nested Subroutine Capability
Nested Subroutine Support

- 4 levels of nesting
- 1 stack
- Stack Pointer Reg.
Loop Support

[_diagram of loop support with addresses and loop count_]

[note to EDITOR: was with no number in author’s text - now in numbered sequence]
Single Loop Instruction Support

![Diagram showing the logic of single loop instruction support.]

- **Comparator**
  - Inputs: Loop start addr reg, Loop end addr reg, Loop counter reg
  - Outputs: PC

- **From IR**
  - Inputs: Loop end addr reg, Loop counter reg
  - Outputs: loop inst

- **OR**
  - Inputs: Loop end addr flag, Loop count zero
  - Outputs: loop_end_inst

- **NOR**
  - Inputs: Loop end flag
  - Outputs: Loop count zero

- **To next addr logic**
  - Inputs: loop_end_inst, count is not 0
  - Outputs: to next addr logic
Nested Loop Support

(a)

<table>
<thead>
<tr>
<th>Loop start addr</th>
<th>Loop end addr</th>
<th>Loop count</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>27</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>12</td>
</tr>
</tbody>
</table>

(b)
Loop Machine

- Write enable (write_en)
- Incremented start address
- End address from microcode
- Loop count fields from microcode
- Stack pointer logic
- Down counter
- Loop end flag
- Push SP OR Loop End Flag
- Zero check logic
- NOR gate
- Loop instr push
- To Next Address Generation Unit
- Current
- Stack pointer logic
- Loop count not 0
- Loop end instr flag
- Loop end flag
- PC
- Comparator
Implementation of Microprogrammed Control in FPGAs

Bruce W. Bomar, Senior Member, IEEE
A programmable microprocessor with a 64-deep ROM for program storage.
- Four 4-bit general purpose registers.
- 6-bit address space and branching support
- Addition/subtraction of 4-bit signed integers (maximum of 4 operands).
- Multiplication Add defined on 4-bit numbers (two + two operands).
- Logical operations (and or xor nor) on 4-bit numbers (maximum of 4 operands).
- Unconditional Branch.
- 2-deep Subroutine call and return mechanism.
- Non nested loops.
- Load a 4-bit from a 4x 64 Data ROM to any of the 4 registers.
- Load a 4-bit from an input data line to any of the 4 registers.
- Store the contents of any of the 4 registers to an output data register with output valid flag enabled for one cycle.
- Three condition flags, Zero, Positive and Overflow.
- Conditional execution of any of the instructions based on three condition flags.
- All instructions complete in one machine cycle.
Example: Motion Estimation
current block  next column  Last target block in first row
daisy chain search
next row

target