Outline

- Alternative controller FSM implementation approaches based on:
  - *Classical Moore and Mealy machines*
  - *Time-State: Divide and Conquer*
  - *Jump counters*
  - Microprogramming (ROM) based approaches
    - » Branch sequencers
    - » Horizontal microcode
    - » Vertical microcode

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

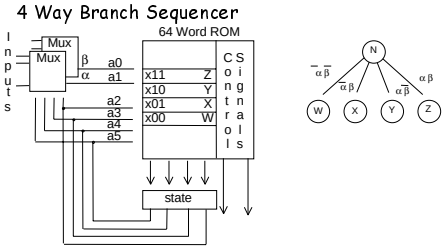
Note: Jump counter trades off ROM size vs. external logic
Only jump states kept in ROM
Even in hybrid approach, state + input subset form ROM address

Branch Sequencer: between the extremes
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

Branch Sequencers

4 Way Branch Sequencer



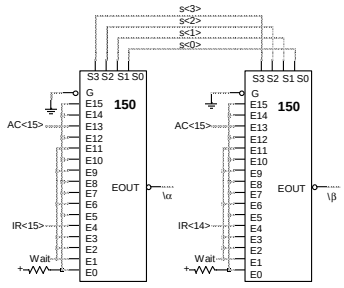
Current State selects two inputs to form part of ROM address

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

Every state has exactly four possible next states

Branch Sequencer

Processor CPU Design Example



Alpha, Beta multiplexer input setup

Example Processor FSM

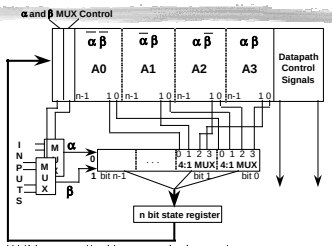
ROM ADDRESS (Reset, Current State, a, b)				ROM CONTENTS Next State	Register Transfer Operations
RES	0	0000	X X	0001 (IF0)	PC → MAR, PC + 1 → PC
IF0	0	0001	0 0	0001 (IF0)	
	0	0001	1 1	0010 (IF1)	MAR → Mem, Read, Request
IF1	0	0010	0 0	0011 (IF2)	MAR → Mem, Read, Request
	0	0010	1 1	0010 (IF1)	Mem → MBR
IF2	0	0011	0 0	0011 (IF2)	
	0	0011	1 1	0100 (OD)	MBR → IR
OD	0	0100	0 0	0101 (LD0)	IR → MAR
	0	0100	0 1	1000 (ST0)	IR → MAR, AC → MBR
	0	0100	1 0	1001 (AD0)	IR → MAR
	0	0100	1 1	1101 (BR0)	IR → MAR

Example Processor FSM

ROM ADDRESS (Reset, Current State, a, b)				ROM CONTENTS Next State	Register Transfer Operations
LD0	0	0101	X X	0110 (LD1)	MAR → Mem, Read, Request
LD1	0	0110	0 0	0111 (LD2)	Mem → MBR
	0	0110	1 1	0110 (LD1)	MAR → Mem, Read, Request
LD2	0	0111	X X	0000 (RES)	MBR → AC
ST0	0	1000	X X	1001 (ST1)	MAR → Mem, Write, Request, MBR → Mem
ST1	0	1001	0 0	0000 (RES)	
	0	1001	1 1	1001 (ST1)	MAR → Mem, Write, Request, MBR → Mem
AD0	0	1010	X X	1011 (AD1)	MAR → Mem, Read, Request
AD1	0	1011	0 0	1100 (AD2)	
	0	1011	1 1	1011 (AD1)	MAR → Mem, Read, Request
AD2	0	1100	X X	0000 (RES)	MBR + AC → AC
BR0	0	1101	0 0	0000 (RES)	
	0	1101	1 1	0000 (RES)	IR → PC

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) x 64 = 1152 ROM bits
- Horizontal format: (14 + 4 x 4 + 2) x 16 = 512 ROM bits

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

Microprogramming

Register Transfer/Microoperations

14 Register Transfer operations become 22 Microoperations:

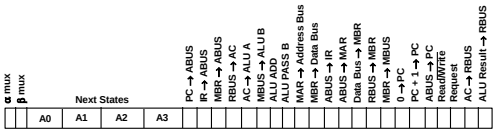
- PC → ABUS
- IR → ABUS
- MBR → ABUS
- RBUS → AC
- AC → ALU A
- MBUS → ALU B
- ALU ADD
- ALU PASS B
- MAR → Address Bus
- MBR → Data Bus
- ABUS → IR
- ABUS → MAR
- Data Bus → MBR
- RBUS → MBR
- MBR → MBUS
- 0 → PC
- PC + 1 → PC
- ABUS → PC
- Read/Write
- Request
- AC → RBUS
- ALU Result → RBUS

Horizontal Microprogramming

Horizontal Branch Sequencer

α, β Mux bits
4 x 4 Next State bits
22 Control operation bits

40 bits total



Horizontal Microprogramming

Moore Processor ROM

Current State (Address)	α mux	β mux	Next States A0	A1	A2	A3	PC → ABUS	IR → ABUS	MBR → ABUS	RBUS → AC	AC → ALU A	MBUS → ALU B	ALU ADD	ALU PASS B	MAR → Address Bus	MBR → Data Bus	ABUS → IR	ABUS → MAR	Data Bus → MBR	RBUS → MBR	MBR → MBUS	0 → PC	PC + 1 → PC	ABUS → PC	Read/Write	Request	AC → RBUS	ALU Result → RBUS
RES (0000)	0	0	0001	0001	0001	0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
IF0 (0001)	0	0	0010	0010	0010	0010	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
IF1 (0010)	0	0	0010	0010	0011	0011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IF2 (0011)	0	0	0100	0100	0011	0011	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0
IF3 (0100)	0	0	0100	0100	0101	0101	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
OD (0101)	1	1	0110	1001	1011	1110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LD0 (0110)	0	0	0111	0111	0111	0111	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
LD1 (0111)	0	0	1000	1000	0111	0111	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0
LD2 (1000)	0	0	0001	0001	0001	0001	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
ST0 (1001)	0	0	1010	1010	1010	1010	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
ST1 (1010)	0	0	0001	0001	1010	1010	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
AD0 (1011)	0	0	1100	1100	1100	1100	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
AD1 (1100)	0	0	1101	1101	1100	1100	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0
AD2 (1101)	0	0	0001	0001	0001	0001	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
BR0 (1110)	0	1	0001	1111	0001	1111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BR1 (1111)	0	0	0001	0001	0001	0001	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

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

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

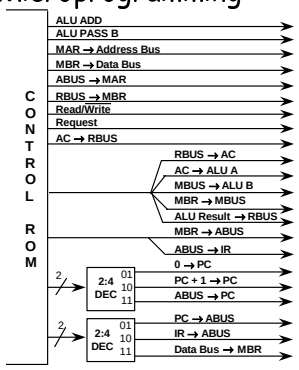
Example:

0 → PC, PC + 1 → PC, ABUS → PC mutually exclusive

Save ROM bit with external 2:4 Decoder

Horizontal Microprogramming

Partially Encoded Control Outputs



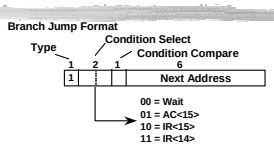
Vertical Microprogramming

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

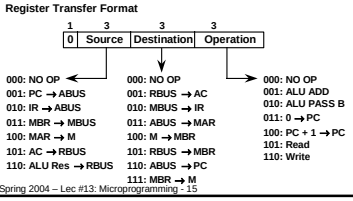
Vertical Microprogramming

Branch Jump
Compare indicated signal to 0 or 1



Register Transfer
Source, Destination, Operation

10 ROM Bits



Vertical Microprogramming

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

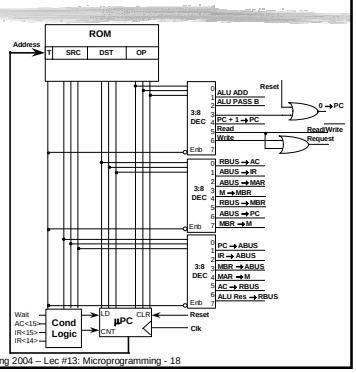
Vertical Microprogramming

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

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

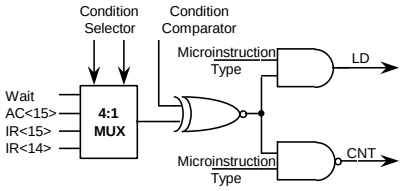
Vertical Programming

Controller Block Diagram



Vertical Microprogramming

Condition Logic



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 processors
  - Make the native instruction set simple and fast
  - Write "higher level" functions as assembly language sequences

Controller Implementation Summary

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