EEC 170: Computer Architecture

Multicycle Control / Microprogramming

Summary of Multicycle Datapath

Differences from single cycle:

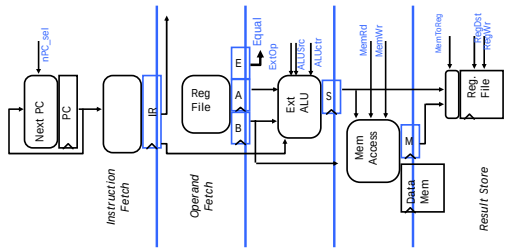
- All intermediate results -> intermediate registers
- Logical RTL -> more complex physical RTL
- Possible reuse of hardware (as in book)

Goal:

- Not all instructions take same amount of time
  - Savings from shorter instructions
- Costs:
  - More hardware (intermediate registers, control)
  - Imbalance between cycles

This is only an intermediate representation!

Example Multicycle Datapath



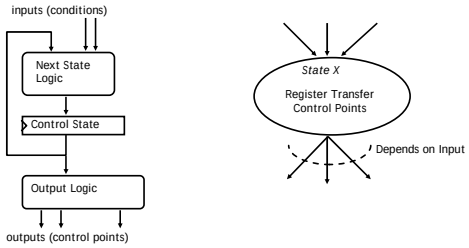
Critical Path?

Our Control Model

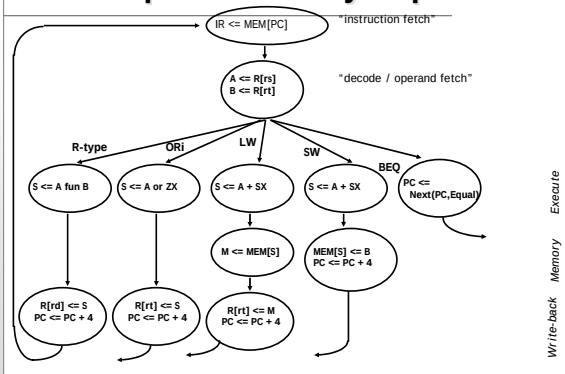
State => set up control logic for Register Transfer

State + Inputs => next state

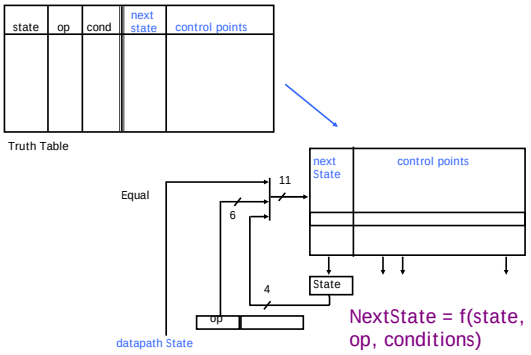
Transfer occurs upon exiting state (same clock edge)



Control Spec for multicycle proc



Traditional FSM Controller



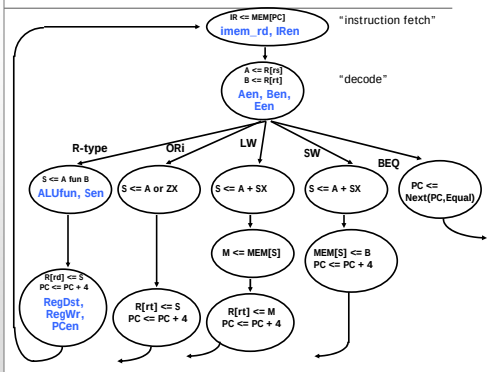
datapath + state diagram ⇒ control)

Translate RTs into control points

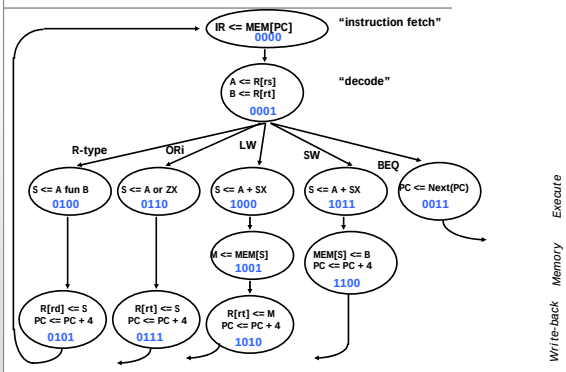
Assign states

Then go build the controller

Mapping Register Transfers to Control Points



Assigning States



(Mostly) Detailed Control Specs (missing⇒0)

	State	Op field	Eq	Next	IR	PC en sel	Ops A B E	Exec Ex Src ALU S	Mem R W M	Write-Back R-M Wr Dst
	0000	??????	?	0001	1					
	0001	BEQ	x	0011			1 1 1			
	0001	R-type	x	0100			1 1 1			
	0001	ORI	x	0110			1 1 1			
	0001	LW	x	1000			1 1 1			
	0001	SW	x	1011			1 1 1			
BEQ:	0011	xxxxxx	0	0000		1 0				x 0 x
	0011	xxxxxx	1	0000		1 1				x 0 x
R:	0100	xxxxxx	x	0101				0 1 fun 1		
	0101	xxxxxx	x	0000		1 0				0 1 1
ORI:	0110	xxxxxx	x	0111				0 0 or 1		
	0111	xxxxxx	x	0000		1 0				0 1 0
LW:	1000	xxxxxx	x	1001				1 0 add 1		
	1001	xxxxxx	x	1010					1 0 1	
	1010	xxxxxx	x	0000		1 0				1 1 0
SW:	1011	xxxxxx	x	1100				1 0 add 1		
	1100	xxxxxx	x	0000		1 0			0 1 0	

-all same in Moore machine

Setting control signals: not much different than SC!

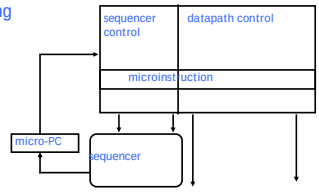
Controller Design

The state diagrams that define the controller for an instruction set processor are highly structured

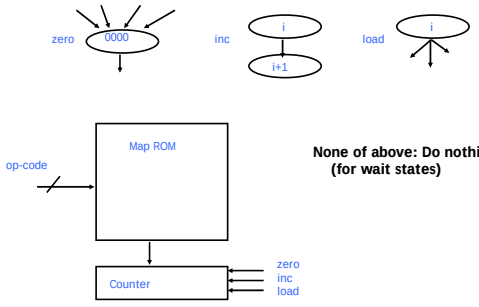
Use this structure to construct a simple "microsequencer"

Control reduces to programming this very simple device

- ⇒ microprogramming
- Goal: Simpler implementation!

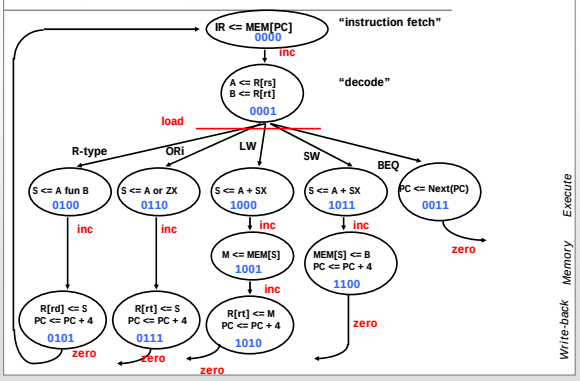


Example: Jump-Counter

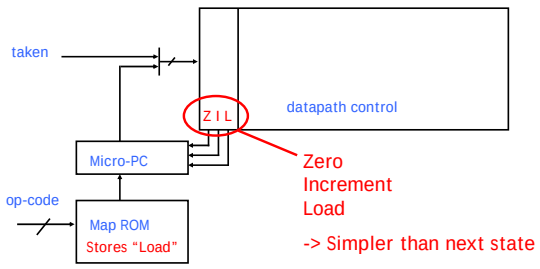


None of above: Do nothing (for wait states)

Using a Jump Counter



Our Microsequencer



Microprogram Control Specification

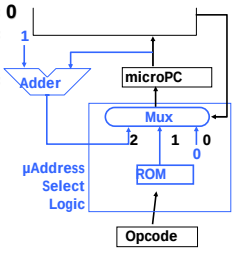
	μPC	Taken	NextIR	PC	Ops en sel	Exec A B	Mem Ex Sr ALU S	Write-Back R W M	M-R Wr Dst
	0000	?	inc	1					
	0001	x	load			1 1			
BEQ	0011	0	zero		1 0				
	0011	1	zero		1 1				
R:	0100	x	inc				0 1 fun 1		
	0101	x	zero		1 0				0 1 1
ORi:	0110	x	inc				0 0 or 1		
	0111	x	zero		1 0				0 1 0
LW:	1000	x	inc				1 0 add 1		
	1001	x	inc					1 0 1	
	1010	x	zero		1 0				1 1 0
SW:	1011	x	inc				1 0 add 1		
	1100	x	zero		1 0			0 1 0	

Adding the Dispatch ROM

Sequencer-based control

- Called "microPC" or "μPC" vs. state register

Control Value	Effect
00 (Zero)	Next μaddress = 0
01 (Load)	Next μaddress = dispatch ROM
10 (Inc)	Next μaddress = μaddress + 1



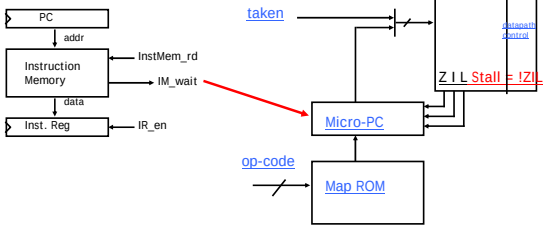
ROM:

R-type	000000	0100
BEQ	000100	0011
ori	001101	0110
LW	100011	1000
SW	101011	1011

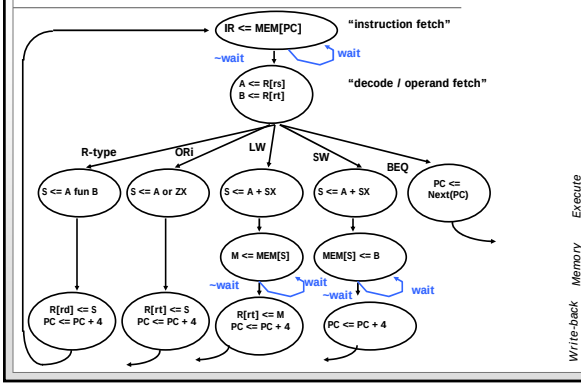
Example: Controlling Memory

This controller is powerful!

Example: What to do if memory is not ready?



Controller handles non-ideal memory



Multicycle Control Summary

What's the same as SC:

- Setting of control signals, generally

What's different from SC:

- Must handle intermediate registers
- New concept: *states*
- Need way to get from one state to another
  - Traditional FSM Controller
  - Jump table
  - Microcode

Announcements

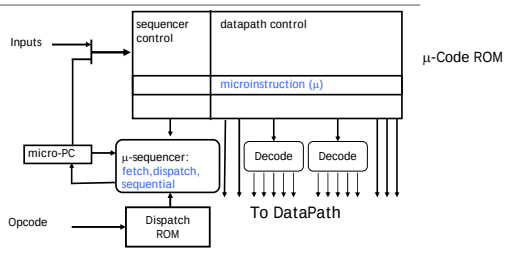
- **Midterm coming up next Wednesday (11/9)**
- **You can pick up your graded hw**
- **HW #4 (chapter 5) is posted on class webpage**
  - As usual, due Friday (11/11) at 5pm.
  - Try to finish it before the mid-term.
- **I will be out of town next Monday (11/7)**
  - Christophe will be here to teach the class
  - Mid-term review, problem solving, and questions
- **Additional office hours before the test**
  - Wednesday (11/9) 11am-1pm

End of Midterm Material

For the midterm, you need to know up to this slide

- Know:
  - Single cycle datapath/control
  - Multiple cycle datapath
  - Multiple cycle control, traditional FSM
  - Multiple cycle control, micro-sequencing
    - This is actually in the CD (section 5.7 of the book)
- Reading: Up to and including Ch. 5

Microprogramming



Microprogramming is a fundamental concept

- implement an instruction set by building a very simple processor and *interpreting* the instructions
- essential for very complex instructions and when few register transfers are possible
- overkill when ISA matches datapath 1-1

Microprogramming

Microprogramming is a convenient method for implementing structured control state diagrams:

- Random logic replaced by microPC sequencer and ROM
- Each line of ROM called a μinstruction:
  contains sequencer control + values for control points
- limited state transitions: (jump table)
  branch to zero, next sequential,
  branch to μinstruction address from dispatch ROM

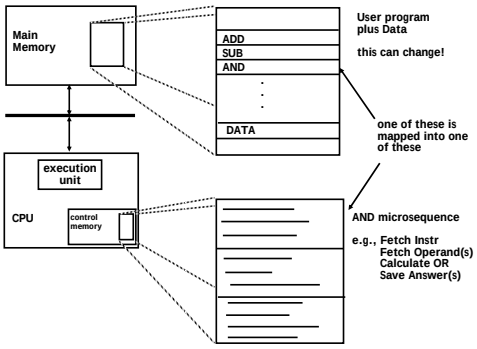
Horizontal μCode: one control bit in μInstruction for every control line in datapath (like what we've done before)

Vertical μCode: groups of control-lines coded together in μInstruction (e.g. possible ALU dest) (new!)

Control design reduces to Microprogramming

- Part of the design process is to develop a "language" that describes control and is easy for humans to understand

"Macroinstruction" Interpretation

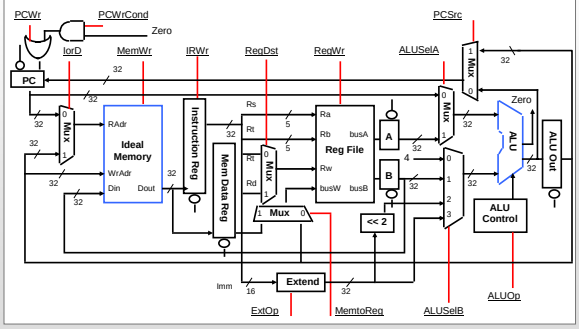


Designing a Microinstruction Set

1) Start with list of control signals
2) Group signals together that make sense (vs. random): called "fields"
3) Place fields in some logical order (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
4) To minimize the width, encode operations that will never be used at the same time - vertical
5) Create a symbolic legend for the microinstruction format, showing name of field values and how they set the control signals
   - Use computers to design computers

Again: Alternative multicycle datapath (book)

Goal here: Horizontal -> Vertical Microcode



1&2) Start with list of control signals, grouped into fields

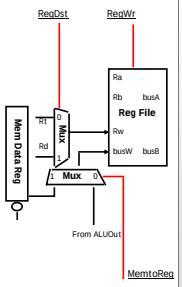
	Signal name	Effect when deasserted	Effect when asserted
Single Bit Control	ALUSelA	1st ALU operand = PC	1st ALU operand = Reg[rs]
	RegWrite	None	Reg. is written
	MemtoReg	Reg. write data input = ALU	Reg. write data input = memory
	RegDst	Reg. dest. no. = rt	Reg. dest. no. = rd
	MemRead	None	Memory at address is read, MDR <= Mem[addr]
	MemWrite	None	Memory at address is written
	IorD	Memory address = PC	Memory address = S
	IRWrite	None	IR <= Memory
	PCWrite	None	PC <= PCSource
	PCWriteCond	None	IF ALUzero then PC <= PCSource
	PCSource	PCSource = ALU	PCSource = ALUout
	ExtOp	Zero Extended	Sign Extended

	Signal name	Value	Effect
Multiple Bit Control	ALUOp	00	ALU adds
		01	ALU subtracts
		10	ALU does function code
		11	ALU does logical OR
	ALUSelB	00	2nd ALU input = 4
		01	2nd ALU input = Reg[rt]
		10	2nd ALU input = extended,shift left 2
		11	2nd ALU input = extended

Compression of Fields

Only look at writeback control signals:

- RegDst: Rt or Rd
- RegWr: Do we write?
- MemToReg: WB from mem or ALU?
- So this takes 3 bits! HORIZONTAL UCODE



But what are our possibilities?

- RegWr = 0, RegDst = MemToReg = X (00)
- RegWr = 1, RegDst = Rt, MemToReg = 1 (01)
- RegWr = 1, RegDst = Rt, MemToReg = 0 (10)
- RegWr = 1, RegDst = Rd, MemToReg = 0 (11)
- Only 2 bits! VERTICAL UCODE

3&4) Microinstruction Format: unencoded vs. encoded fields

Field Name	Width		Control Signals Set
	wide	narrow	
ALU Control	4	2	ALUOp
SRC1	2	1	ALUSelA
SRC2	5	3	ALUSelB, ExtOp
ALU Destination	3	2	RegWrite, MemtoReg, RegDst
Memory	3	2	MemRead, MemWrite, IorD
Memory Register	1	1	IRWrite
PCWrite Control	3	2	PCWrite, PCWriteCond, PCSource
Sequencing	3	2	AddrCtl
Total width	24	15	bits

5) Legend of Fields and Symbolic Names

Field Name	Values for Field	Function of Field with Specific Value
ALU	Add	ALU adds
	Subt.	ALU subtracts
	Func code	ALU does function code
	Or	ALU does logical OR
SRC1	PC	1st ALU input = PC
	rs	1st ALU input = Reg[rs]
SRC2	4	2nd ALU input = 4
	Extend	2nd ALU input = sign ext. IR[15-0]
	Extend0	2nd ALU input = zero ext. IR[15-0]
	Extshft	2nd ALU input = sign ex., sl IR[15-0]
	rt	2nd ALU input = Reg[rt]
destination	rd ALU	Reg[rd] = ALUout
	rt ALU	Reg[rt] = ALUout
	rt Mem	Reg[rt] = Mem
Memory	Read PC	Read memory using PC
	Read ALU	Read memory using ALUout for addr
	Write ALU	Write memory using ALUout for addr
Memory register	IR	IR = Mem
PC write	ALU	PC = ALU
	ALUoutCond	IF ALU Zero then PC = ALUout
Sequencing	Seq	Go to sequential µinstruction
	Fetch	Go to the first microinstruction
	Dispatch	Dispatch using ROM.

On your own time: what do these fieldnames mean?

Destination:

Code	Name	RegWrite	MemToReg	RegDest
00	---	0	X	X
01	rd ALU	1	0	1
10	rt ALU	1	0	0
11	rt MEM	1	1	0

SRC2:

Code	Name	ALUSelB	ExtOp
000	---	X	X
001	4	00	X
010	rt	01	X
011	ExtShft	10	1
100	Extend	11	1
111	Extend0	11	0

Horizontal vs. Vertical Microcode

Horizontal

- Larger size
- More straightforward
- Possibly more general

Vertical

- Smaller size
- Requires recoding
- More restrictive

Specific Sequencer (from before)

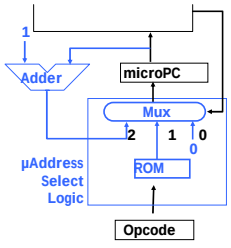
Sequencer-based control unit from earlier

- Called "microPC" or "µPC" vs. state register

Code	Name	Effect
00	fetch	Next µaddress = 0
01	dispatch	Next µaddress = dispatch ROM
10	seq	Next µaddress = µaddress + 1

ROM:

R-type	000000	0100
BEQ	000100	0011
ori	001101	0110
LW	100011	1000
SW	101011	1011



Legacy Software and Microprogramming

IBM bet company on 360 Instruction Set Architecture (ISA): single instruction set for many classes of machines

- (8-bit to 64-bit)

Stuart Tucker stuck with job of what to do about software compatibility

- If microprogramming could easily do same instruction set on many different microarchitectures, then why couldn't multiple microprograms do multiple instruction sets on the same microarchitecture?
- Coined term "emulation": instruction set interpreter in microcode for non-native instruction set
- Very successful: in early years of IBM 360 it was hard to know whether old instruction set or new instruction set was more frequently used

Microprogramming in IBM 360

	M30	M40	M50	M65
Datapath width (bits)	8	16	32	64
µinst width (bits)	50	52	85	87
µcode size (K µinsts)	4	4	2.75	2.75
µstore technology	CCROS	TCROS	BCROS	BCROS
µstore cycle (ns)	750	625	500	200
memory cycle (ns)	1500	2500	2000	750
Rental fee ($K/month)	4	7	15	35

Only fastest models (75 and 95) were hardwired

VLSI & Microprogramming

By late seventies

- technology assumption about ROM & RAM speed became invalid
- micromachines became more complicated
  - to overcome slower ROM, micromachines were pipelined
  - complex instruction sets led to the need for subroutine and call stacks in ucode
  - need for fixing bugs in control programs was in conflict with read-only nature of uROM
  - VAX instruction set had 400-500 kb of control store!
- introduction of caches and buffers, especially for instructions, made multiple-cycle execution of reg-reg instructions unattractive

Modern Usage

Microprogramming is far from extinct

Played a crucial role in micros of the Eighties

- Motorola 68K series
- Intel 386 and 486

Microcode is present in most modern CISC micros in an assisting role (e.g. AMD Athlon, Intel Pentium-4)

- Most instructions are executed directly, i.e., with hard-wired control
- Infrequently-used and/or complicated instructions invoke the microcode engine

Patchable microcode common for post-fabrication bug fixes, e.g. Intel Pentiums load mcode patches at bootup

Microprogramming Pros and Cons

Ease of design

Flexibility

- Easy to adapt to changes in organization, timing, technology
- Can make changes late in design cycle, or even in the field

Can implement very powerful instruction sets (just more control memory)

Generality

- Can implement multiple instruction sets on same machine.
- Can tailor instruction set to application.

Compatibility

- Many organizations, same instruction set

Costly to implement

Slow

Thought: Microprogramming one inspiration for RISC

If simple instruction could execute at very high clock rate…

If you could even write compilers to produce microinstructions…

If most programs use simple instructions and addressing modes…

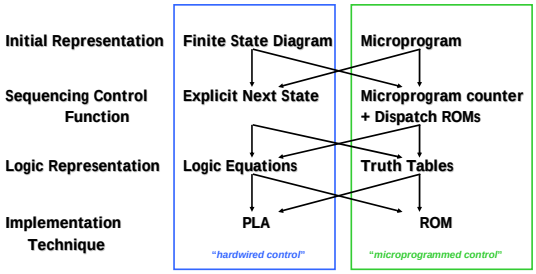
If microcode is kept in RAM instead of ROM so as to fix bugs …

If same memory used for control memory could be used instead as cache for "macroinstructions"…

Then why not skip instruction interpretation by a microprogram and simply compile directly into lowest language of machine? (microprogramming is overkill when ISA matches datapath 1-1)

Overview of Control

Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.



Summary (1 of 3)

Disadvantages of the Single Cycle Processor

- Long cycle time
- Cycle time is too long for all instructions except the Load

Multiple Cycle Processor:

- Divide the instructions into smaller steps
- Execute each step (instead of the entire instruction) in one cycle

Partition datapath into equal size chunks to minimize cycle time

- ~10 levels of logic between latches

Summary (cont'd) (2 of 3)

Control is specified by finite state diagram

Specialize state-diagrams easily captured by microsequencer

- simple increment & "branch" fields
- datapath control fields

Control design reduces to Microprogramming

Control is more complicated with:

- complex instruction sets
- restricted datapaths (see the book)

Simple Instruction set and powerful datapath ⇒ simple control

- could try to reduce hardware (see the book)
- rather go for speed => many instructions at once!

Summary (3 of 3)

Microprogramming is a fundamental concept

- implement an instruction set by building a very simple processor and interpreting the instructions
- essential for very complex instructions and when few register transfers are possible
- *Control design reduces to Microprogramming*

Design of a Microprogramming language

- Start with list of control signals
- Group signals together that make sense (vs. random): called "fields"
- Place fields in some logical order (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
- To minimize the width, encode operations that will never be used at the same time
- Create a symbolic legend for the microinstruction format, showing name of field values and how they set the control signals

Where to get more information?

Multiple Cycle Controller: Appendix C of your text book.

Microprogramming: Section 5.5 of your text book.

D. Patterson, "Microprogramming", Scientific American, March 1983.

D. Patterson and D. Ditzel, "The Case for the Reduced Instruction Set Computer," Computer Architecture News 8, 6 (October 15, 1980)