CS152
Computer Architecture and Engineering
Lecture 12: Designing a Multiple Cycle Controller

March 1, 1995

Dave Patterson (patterson@cs) and
Shing Kong (shing.kong@eng.sun.com)

Slides available on http://http.cs.berkeley.edu/~patterson

Review of a Multiple Cycle Implementation

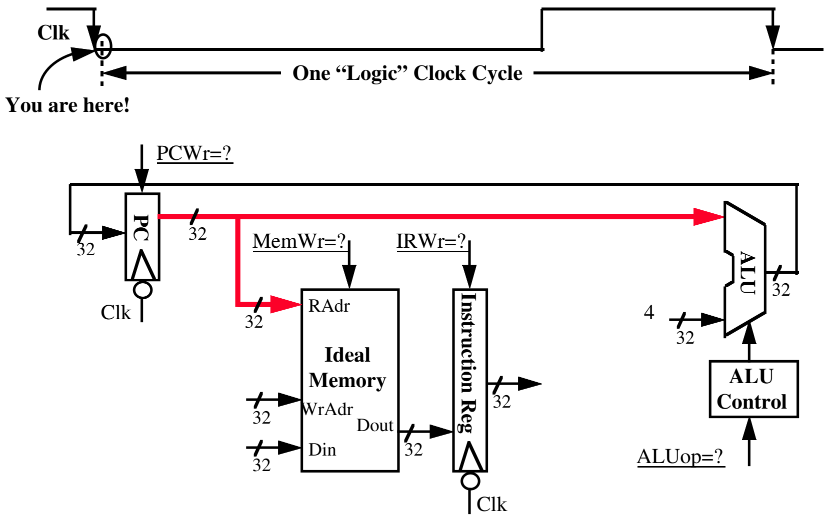
° The root of the single cycle processor's problems:
  • The cycle time has to be long enough for the slowest instruction

° Solution:
  • Break the instruction into smaller steps
  • Execute each step (instead of the entire instruction) in one cycle
    - Cycle time: time it takes to execute the longest step
    - Keep all the steps to have similar length
  • This is the essence of the multiple cycle processor

° The advantages of the multiple cycle processor:
  • Cycle time is much shorter
  • Different instructions take different number of cycles to complete
    - Load takes five cycles
    - Jump only takes three cycles
  • Allows a functional unit to be used more than once per instruction

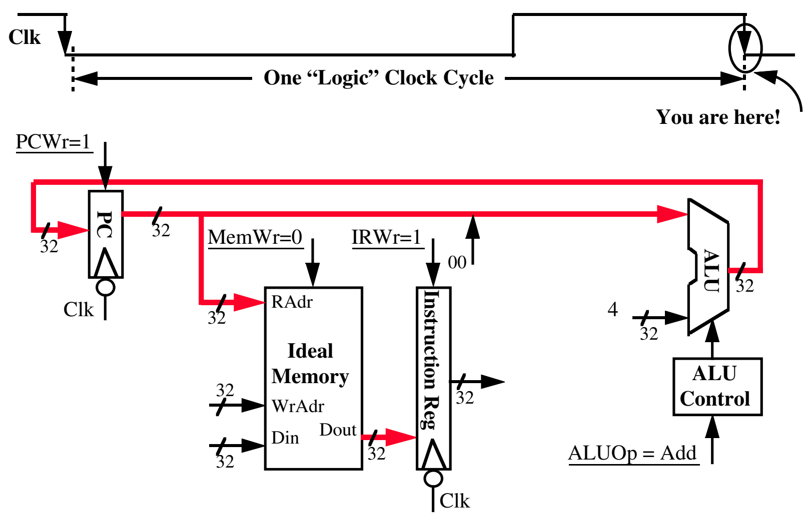
Review: Instruction Fetch Cycle, In the Beginning

- Every cycle begins right AFTER the clock tick:
  - mem[PC]   PC<31:0> + 4

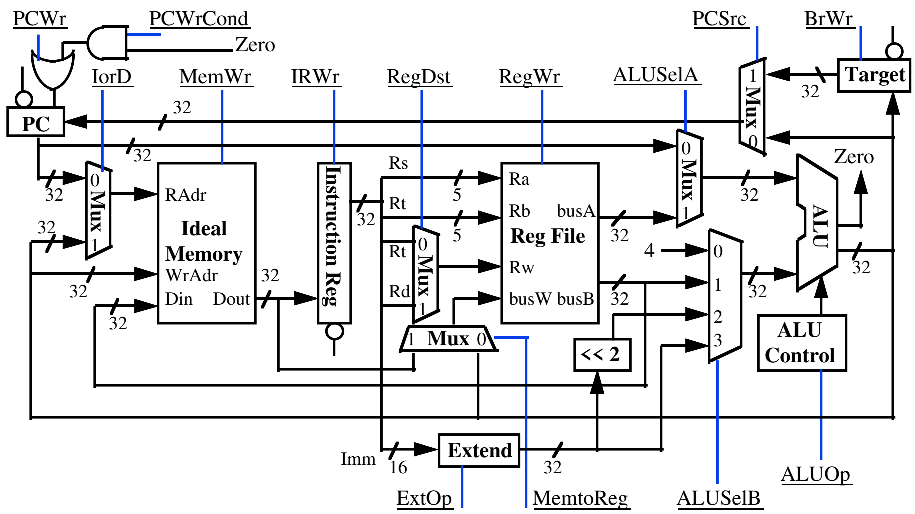


Review: Instruction Fetch Cycle, The End

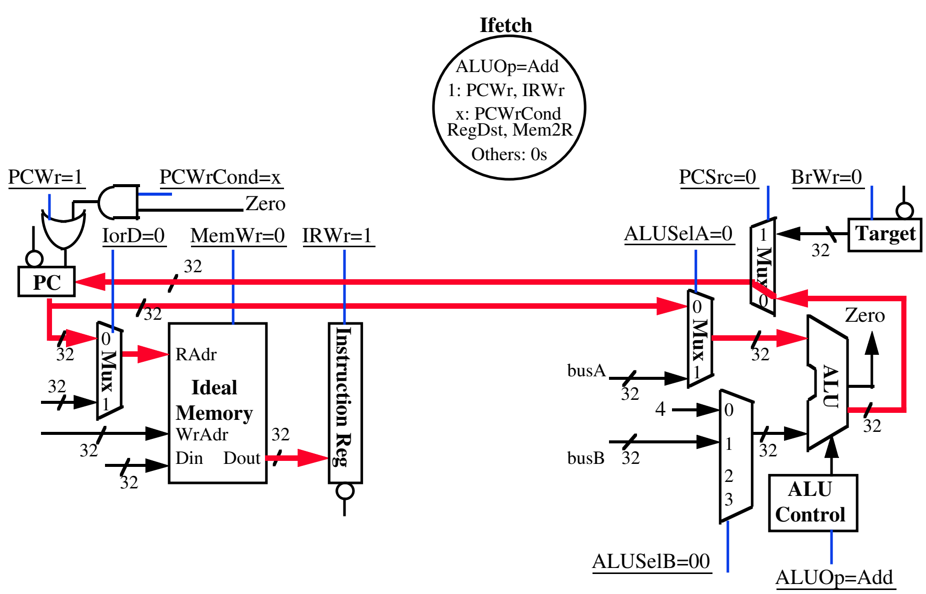
- Every cycle ends AT the next clock tick (storage element updates):
  - IR <-- mem[PC]   PC<31:0> <-- PC<31:0> + 4



Putting it all together: Multiple Cycle Datapath

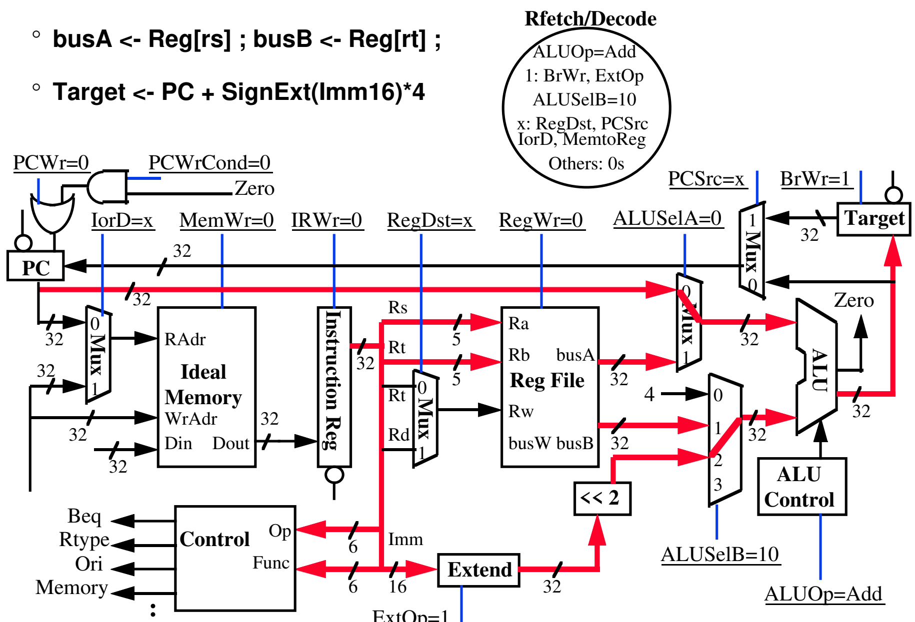


Instruction Fetch Cycle: Overall Picture



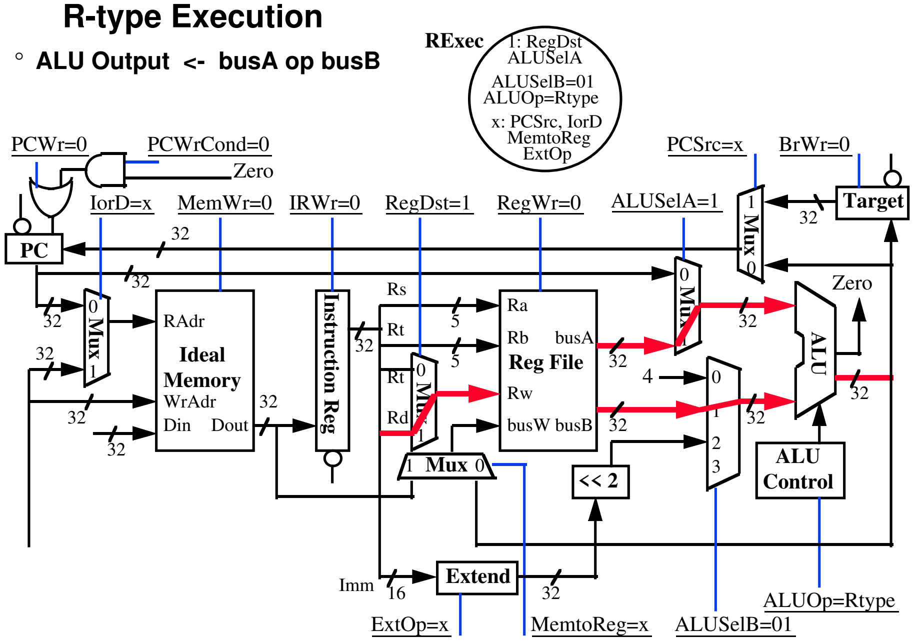
Register Fetch / Instruction Decode (Continue)

- busA <- Reg[rs] ; busB <- Reg[rt] ;
- Target <- PC + SignExt(Imm16)*4



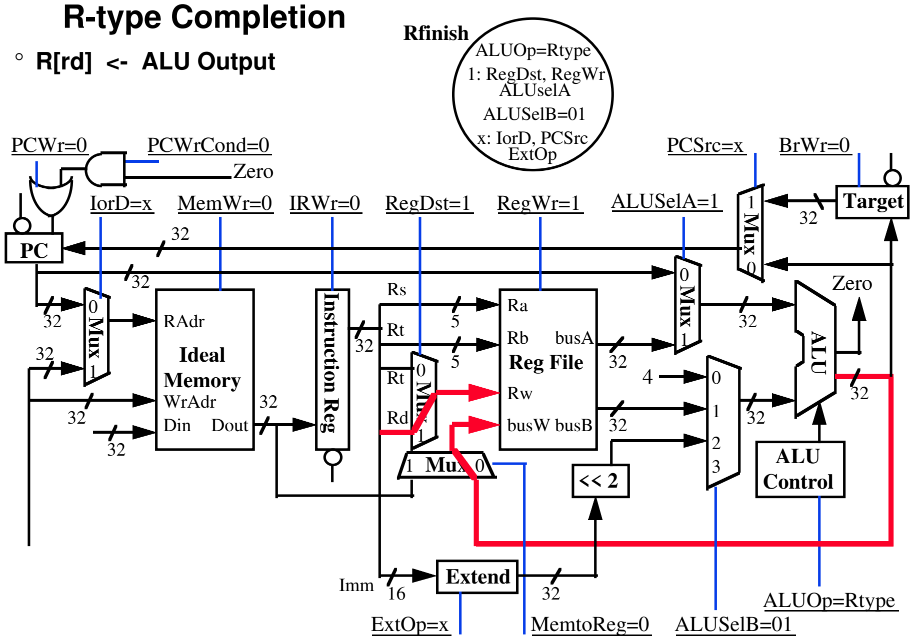
R-type Execution

- ALU Output <- busA op busB



R-type Completion

° R[rd] <- ALU Output



Outline of Today’s Lecture

° Recap (5 minutes)

° Review of FSM control (15 minutes)

° Questions and Administrative Matters (5 minutes)

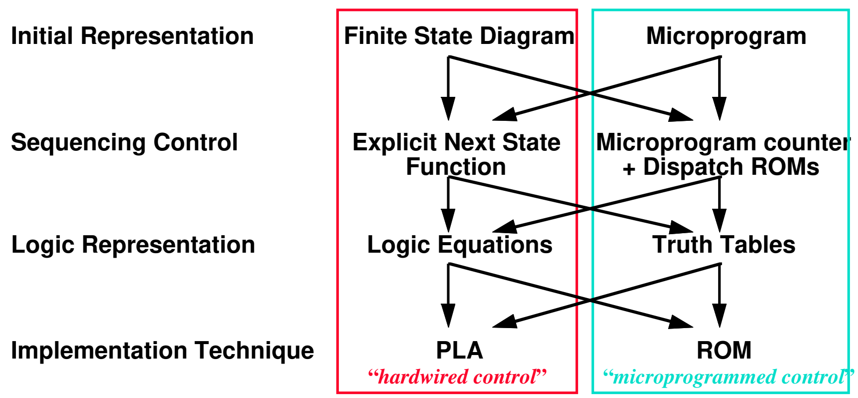
° From Finite State Diagrams to Microprogramming (25 minutes)

° Break (5 minutes)

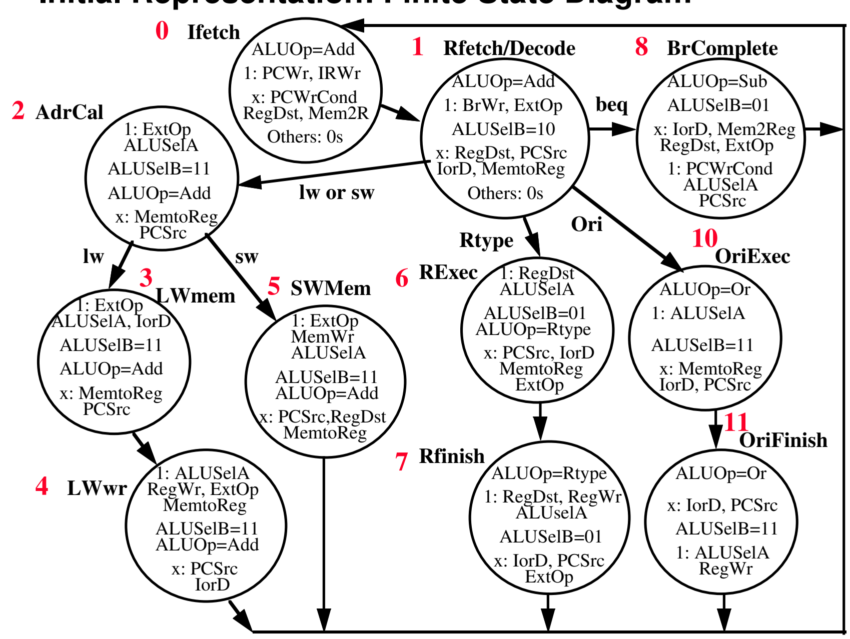
° ABCs of microprogramming (25 minutes)

Overview of Next Two Lectures

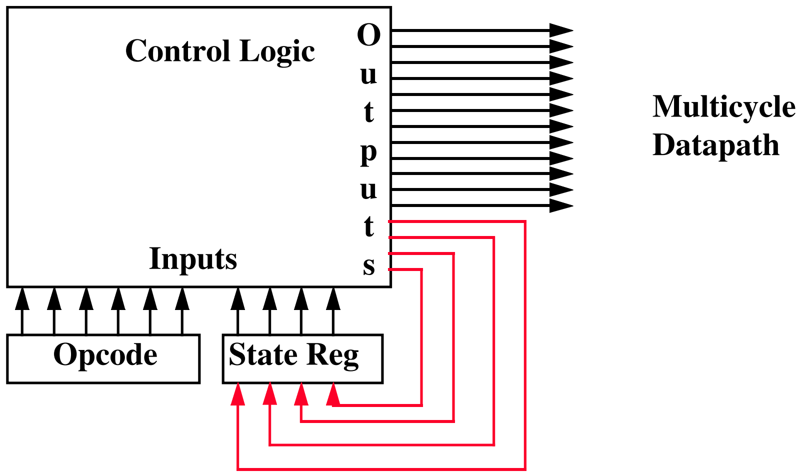
° Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.



Initial Representation: Finite State Diagram



Sequencing Control: Explicit Next State Function



- Next state number is encoded just like datapath controls

Logic Representative: Logic Equations

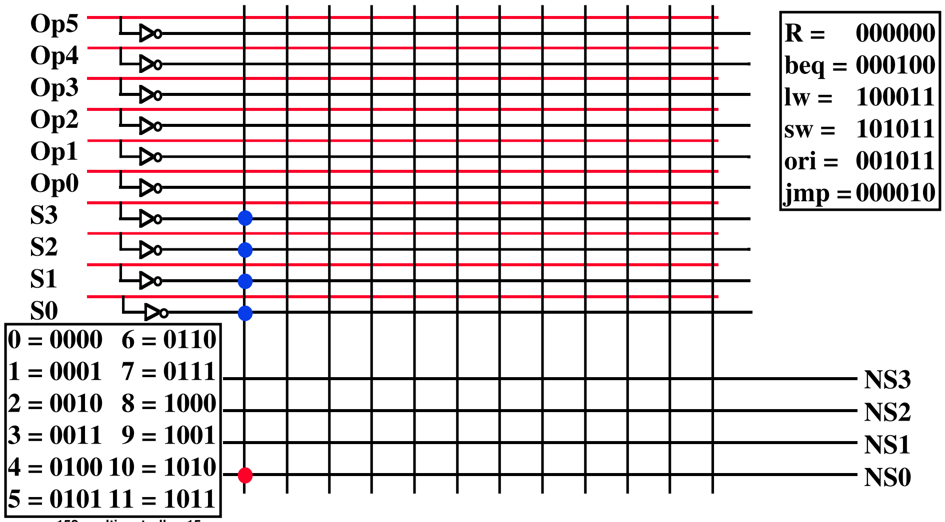
- Next state from current state
  - State 0 -> State1
  - State 1 -> S2, S6, S8, S10
  - State 2 ->___________
  - State 3 ->___________
  - State 4 ->State 0
  - State 5 -> State 0
  - State 6 -> State 7
  - State 7 -> State 0
  - State 8 -> State 0
  - State 9-> State 0
  - State 10 -> State 11
  - State 11 -> State 0

- Alternatively, prior state & condition

S4, S5, S7, S8, S9, S11 -> State0
__________________ -> State 1
__________________ -> State 2
__________________ -> State 3
__________________ -> State 4
State2 & op = sw -> State 5
__________________ -> State 6
State 6 -> State 7
__________________ -> State 8
State2 & op = jmp -> State 9
__________________ -> State 10
State 10 -> State 11

Implementation Technique: Programmed Logic Arrays

° Each output line the logical OR of logical AND of input lines or their complement: AND minterms specified in top AND plane, OR sums specified in bottom OR plane



Questions and Administrative Matters

° Apologize for problems with the franklin.cs
  • Lessons learned for your career?

° "Midterm" for instructors and TAs: constructive criticism by Friday
  • Please put your name, as I want to hear from everyone
  • If you want to submit an anonymous form, just take a second copy
  • Be careful what you wish for, it may come true
    - this semester we switched to the faster HP workstations (which is the cause of the instability) and doubled the number of Powerview licenses and disk space per group
  • Return in class Friday, right after 5 minute break

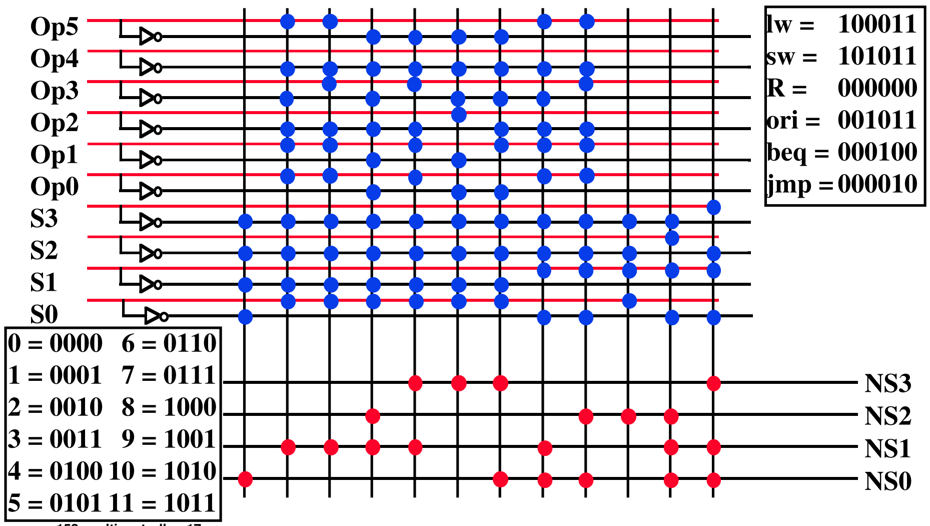
° Email progress reports 4PM Friday

° Assume reasonable dealys for modules for next assignment

° Go to discussion section so that you can meet with your group!!!

Implementation Technique: Programmed Logic Arrays

- Each output line the logical OR of logical AND of input lines or their complement: AND minterms specified in top AND plane, OR sums specified in bottom OR plane



Multicycle Control

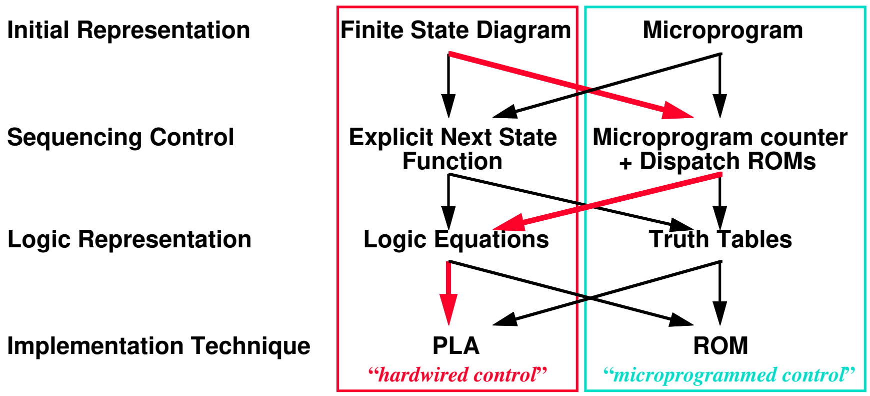
- Given numbers of FSM, can turn determine next state as function of inputs, including current state
- Turn these into Boolean equations for each bit of the next state lines
- Can implement easily using PLA
- What if many more states, many more conditions?
- What if need to add a state?

Next Iteration: Using Sequencer for Next State

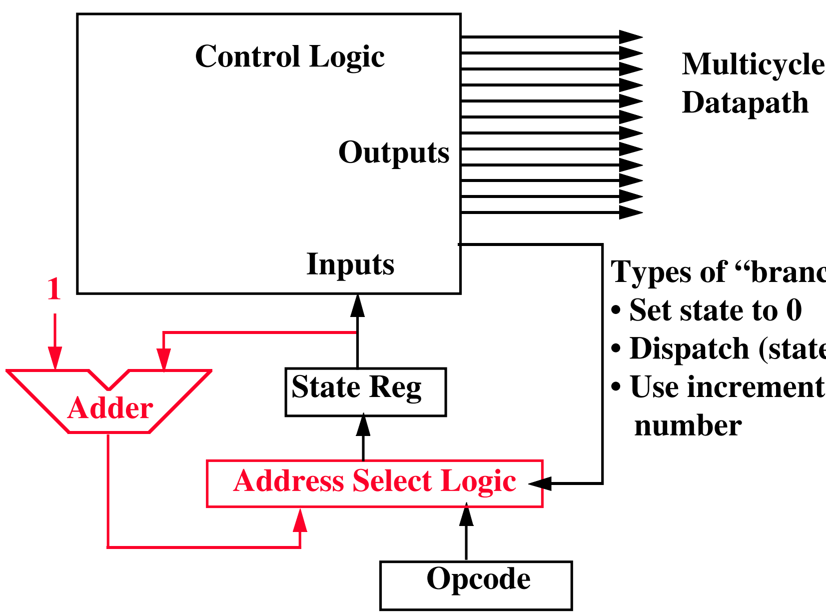
° Before Explicit Next State: Next try variation 1 step from right hand side

° Few sequential states in small FSM: suppose added floating point?

° Still need to go to non-sequential states: e.g., state 1 => 2, 6, 8, 10

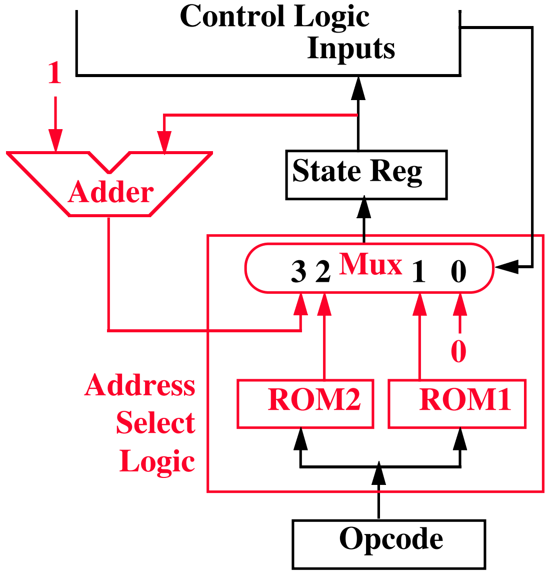


Sequencer-based control unit



Types of "branching"
- Set state to 0
- Dispatch (state 1 & 2)
- Use incremented state number

Sequencer-based control unit details



Dispatch ROM 1		
Op	*Name*	*State*
000000	Rtype	0110
000010	jmp	1001
000100	beq	1000
001011	ori	1010
100011	lw	0010
101011	sw	0010

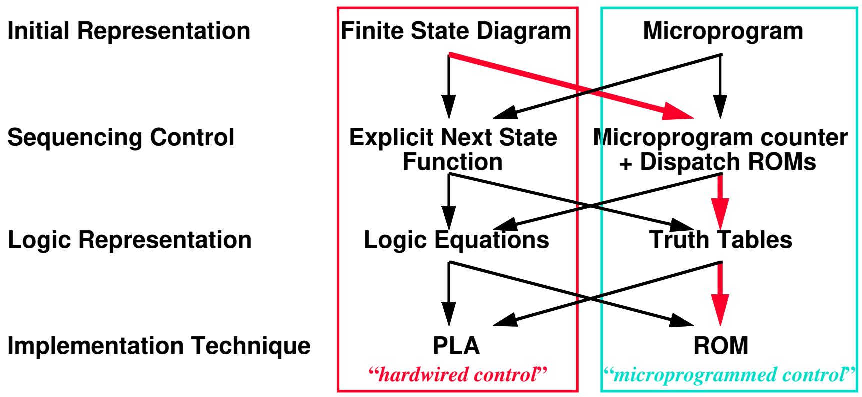
Dispatch ROM 2		
Op	*Name*	*State*
100011	lw	0011
101011	sw	0101

Implementing Control with a ROM

° Instead of a PLA, use a ROM with one word per state (“Control word”)

State number	*Control Word Bits 18-2*	*Control Word Bits 1-0*
0	10010100000001000	11
1	00000000010011000	01
2	00000000000010100	10
3	00110000000010100	11
4	00110010000010110	00
5	00101000000010100	00
6	00000000001000100	11
7	00000000001000111	00
8	01000000100100100	00
9	10000001000000000	00
10	…	11
11	…	00

Next Iteration: Using Microprogram for Representation



- ROM can be thought of as a sequence of control words
- Control word can be thought of as instruction: "microinstruction"
- Rather than program in binary, use assembly language

Break (5 Minutes)

Microprogramming

° Control is the hard part of processor design
  - ° Datapath is fairly regular and well-organized
  - ° Memory is highly regular
  - ° Control is irregular and global

Microprogramming:

-- A Particular Strategy for Implementing the Control Unit of a processor by "programming" at the level of register transfer operations

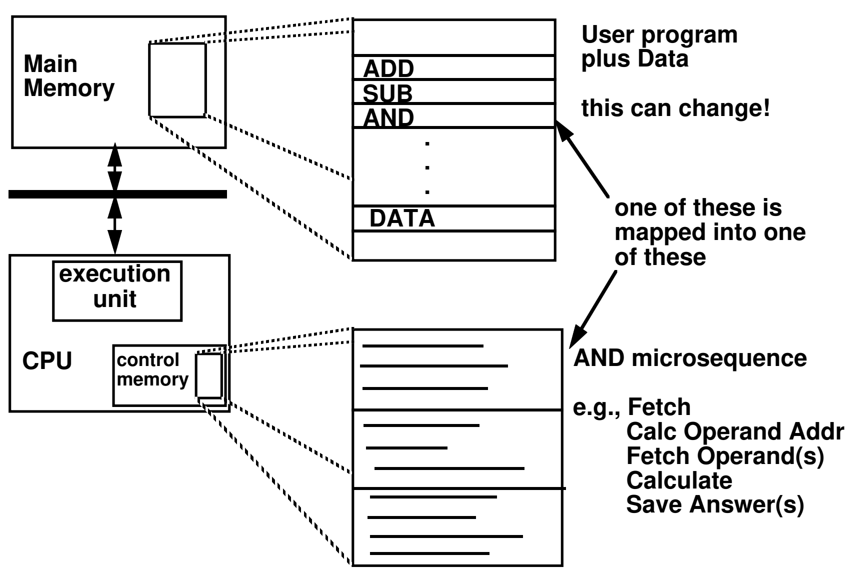
Microarchitecture:

-- Logical structure and functional capabilities of the hardware as seen by the microprogrammer

Historical Note:

IBM 360 Series first to distinguish between architecture & organization
Same instruction set across wide range of implementations, each with different cost/performance

Macroinstruction Interpretation



Variations on Microprogramming

° **Horizontal Microcode**

– control field for each control point in the machine

μseq	μaddr	A-mux	B-mux	bus enables	register enables	

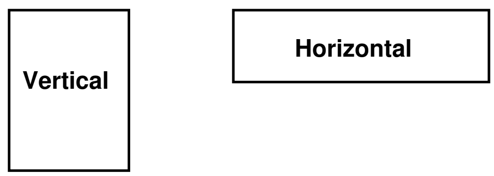
° **Vertical Microcode**

– compact microinstruction format for each class of microoperation

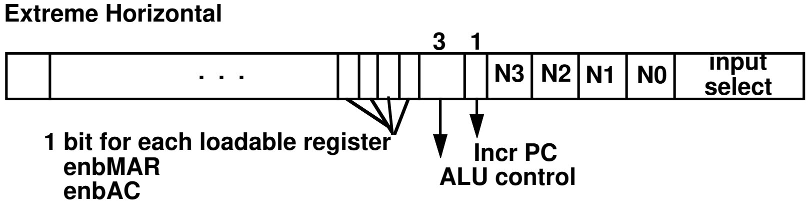
branch: μseq-op μadd

execute: ALU-op A,B,R

memory: mem-op S, D



Extreme Horizontal



Depending on bus organization, many potential control combinations simply not wrong, i.e., implies transfers that can never happen at the same time.

Makes sense to encode fields to save ROM space

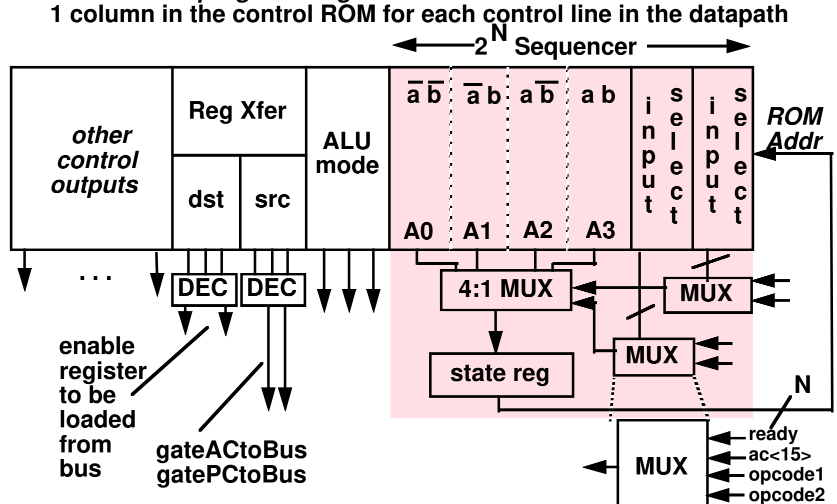
Example: gate Rx and gate Ry to same bus should never happen encoded in single bit which is decoded rather than two separate bits

NOTE: <u>encoding should be just sufficient that parallel actions that the datapath supports should still be specifiable in a single microinstruction</u>

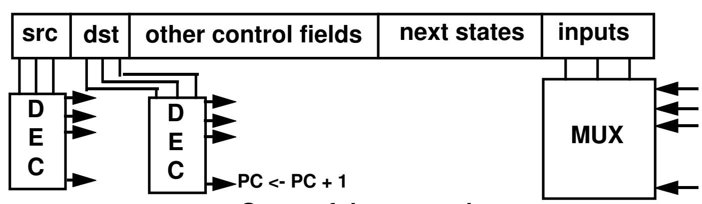
Microprogramming (example)

° use of ROM/RAM to generate control points rather than through discrete logic
° including control of the next-state logic (the microsequencer)

Horizontal Microprogramming
1 column in the control ROM for each control line in the datapath

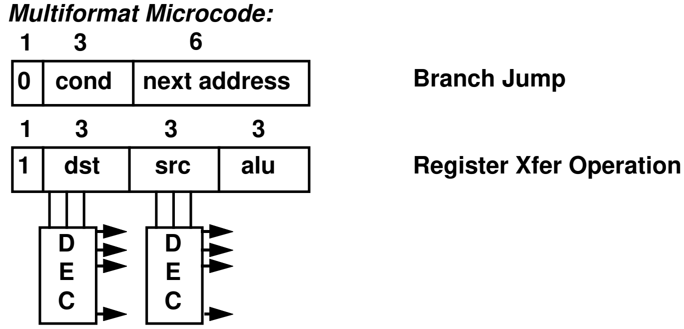


More Vertical Format



Some of these may have nothing to do with registers!

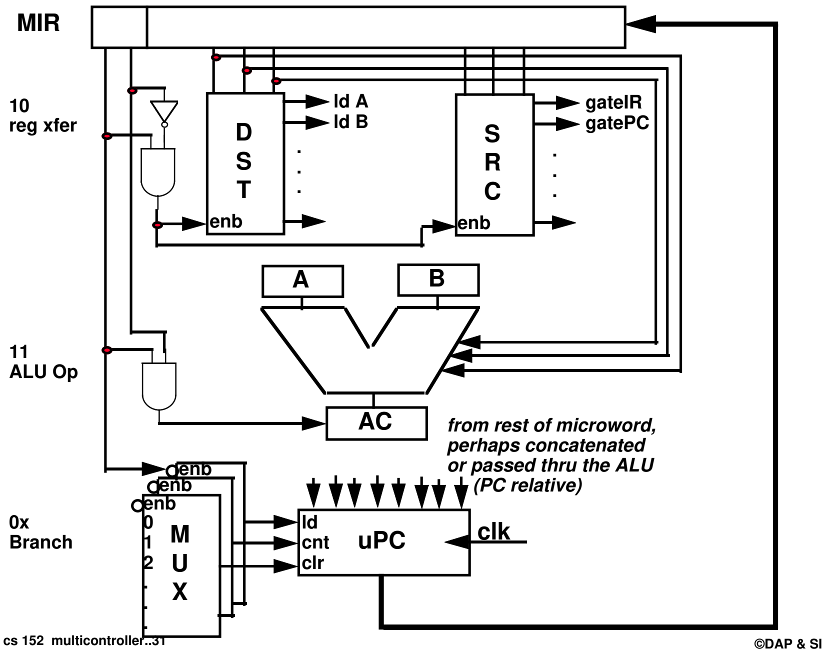
Multiformat Microcode:



Branch Jump

Register Xfer Operation

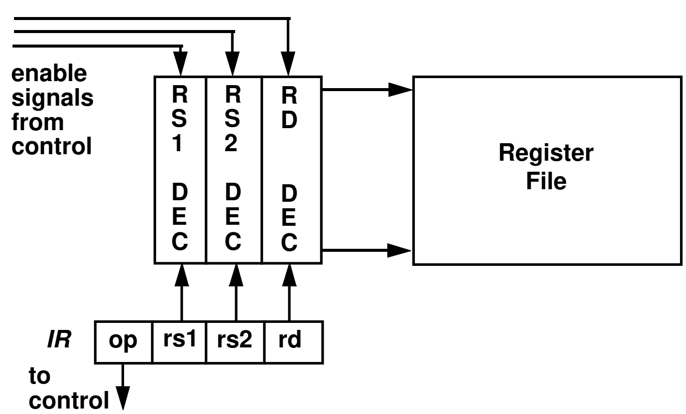
Controller Implementation



Hierarchy of States

Not all critical control information is derived from control logic

E.g., IR contains useful control information, such as register sources, destinations, opcodes, etc.



Horizontal vs. Vertical Microprogramming

NOTE: previous organization is not TRUE horizontal microprogramming; register decoders give flavor of *encoded* microoperations

Most microprogramming-based controllers vary between:

horizontal organization (1 control bit per control point)

vertical organization (fields encoded in the control memory and must be decoded to control something)

Horizontal

+ more control over the potential parallelism of operations in the datapath

- uses up lots of control store

Vertical

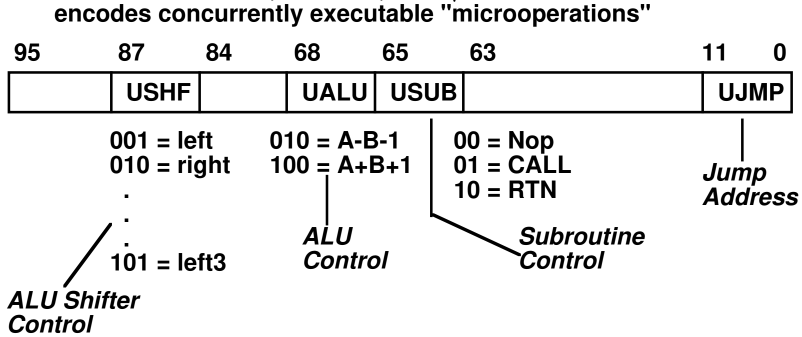
+ easier to program, not very different from programming a RISC machine in assembly language

- extra level of decoding may slow the machine down

Vax Microinstructions

VAX Microarchitecture:

96 bit control store, 30 fields, 4096 µinstructions for VAX ISA
encodes concurrently executable "microoperations"



Legacy Software and Microprogramming

- IBM bet company on 360 Instruciton Set Architecture (ISA): single instruction set for many classes of machines (8-bit to 64-bit)
- Stewart Tucker stuck with job of what to do about software compatability
- 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 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

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?

Summary: Multicycle Control

- Microprogramming and hardwired control have many similarities, perhaps biggest difference is initial representation and ease of change of implementation, with ROM generally being easier than PLA