CMPE 411
Computer Architecture

Lecture 12

Micro-programming & Exceptions

October 10, 2017

www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm

Lecture's Overview

❑ *Previous Lecture:*

- 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
- Control is specified by finite state diagram
- Follow same 5-step method for designing "real" processor

❑ *This Lecture:*

- Micro-programmed control
- Processor exceptions

Overview of Processor Design

Design Steps:

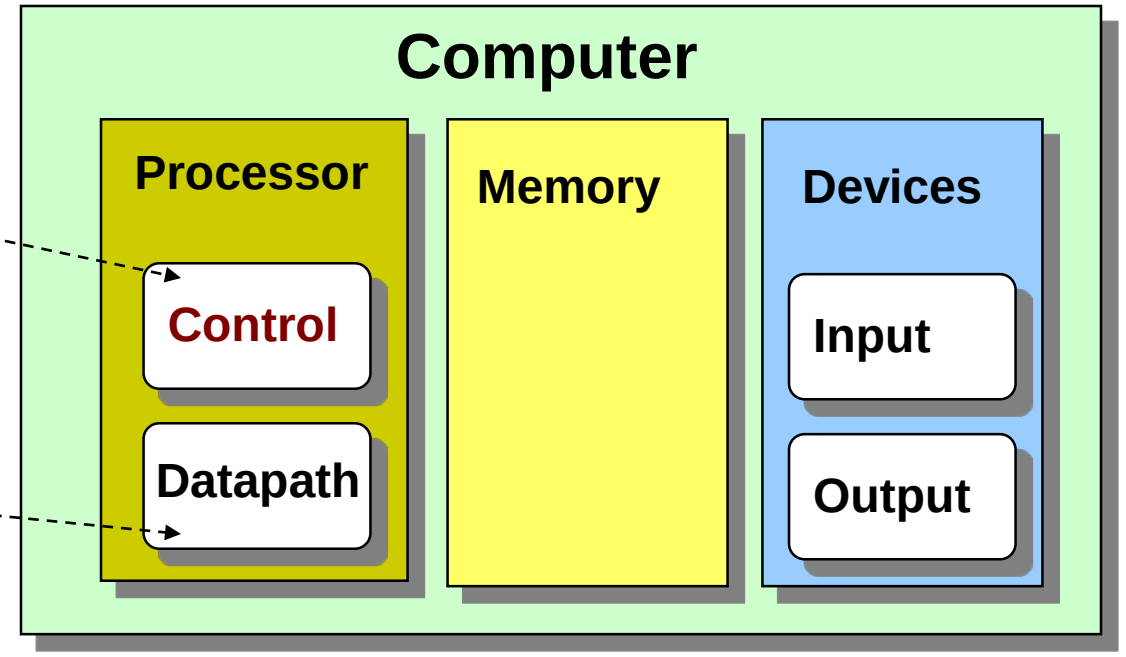
1. Analyze instruction set => datapath requirements
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
5. Assemble the control logic

✓ Applied for multi-cycle processor

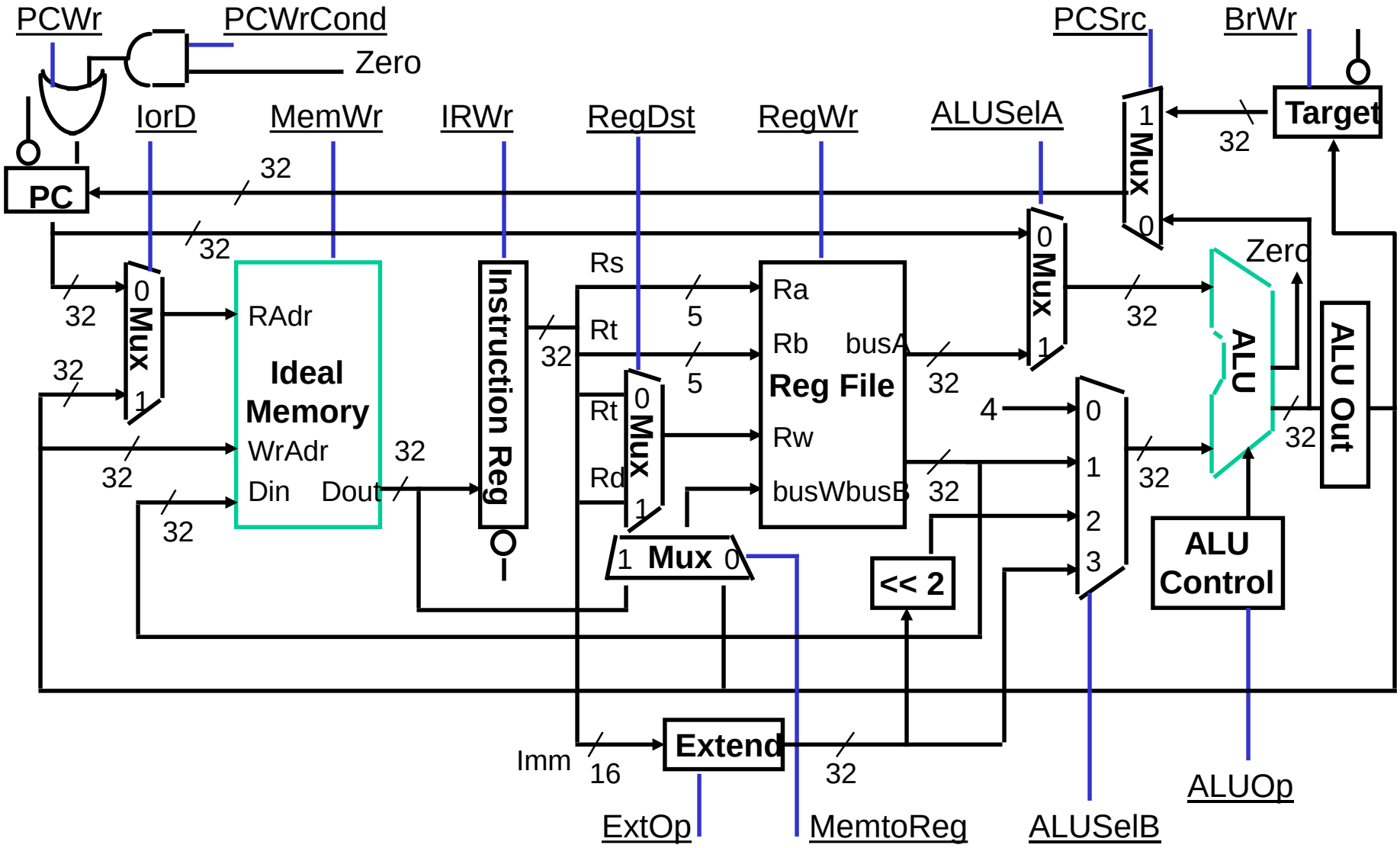
In today's class we study how to design control in a micro-instr.

Coordination for proper operation

Connections for Information flow

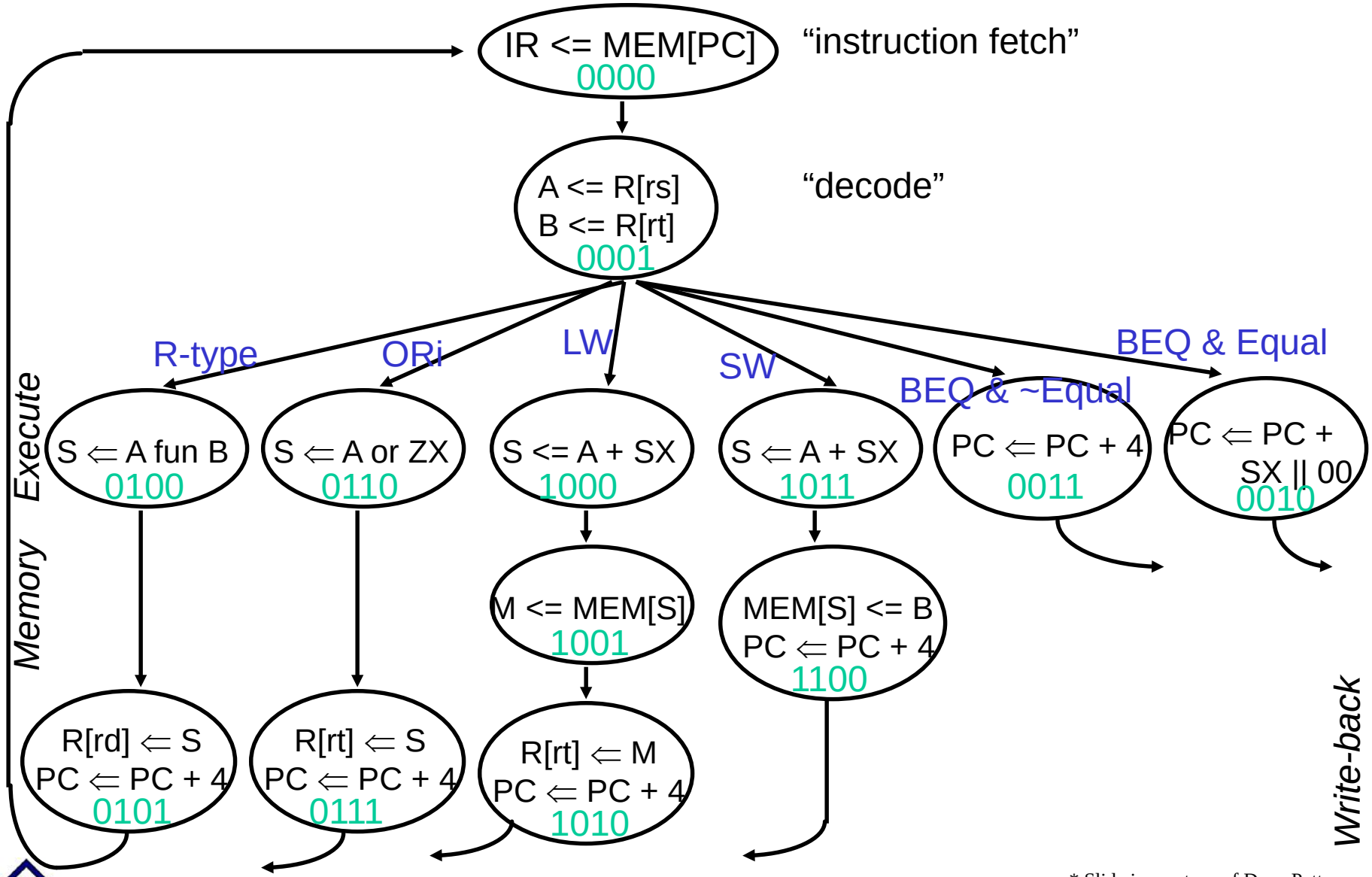


Multiple Cycle Datapath



* Slide is courtesy of Dave Patterson

Controller FSM Specifications

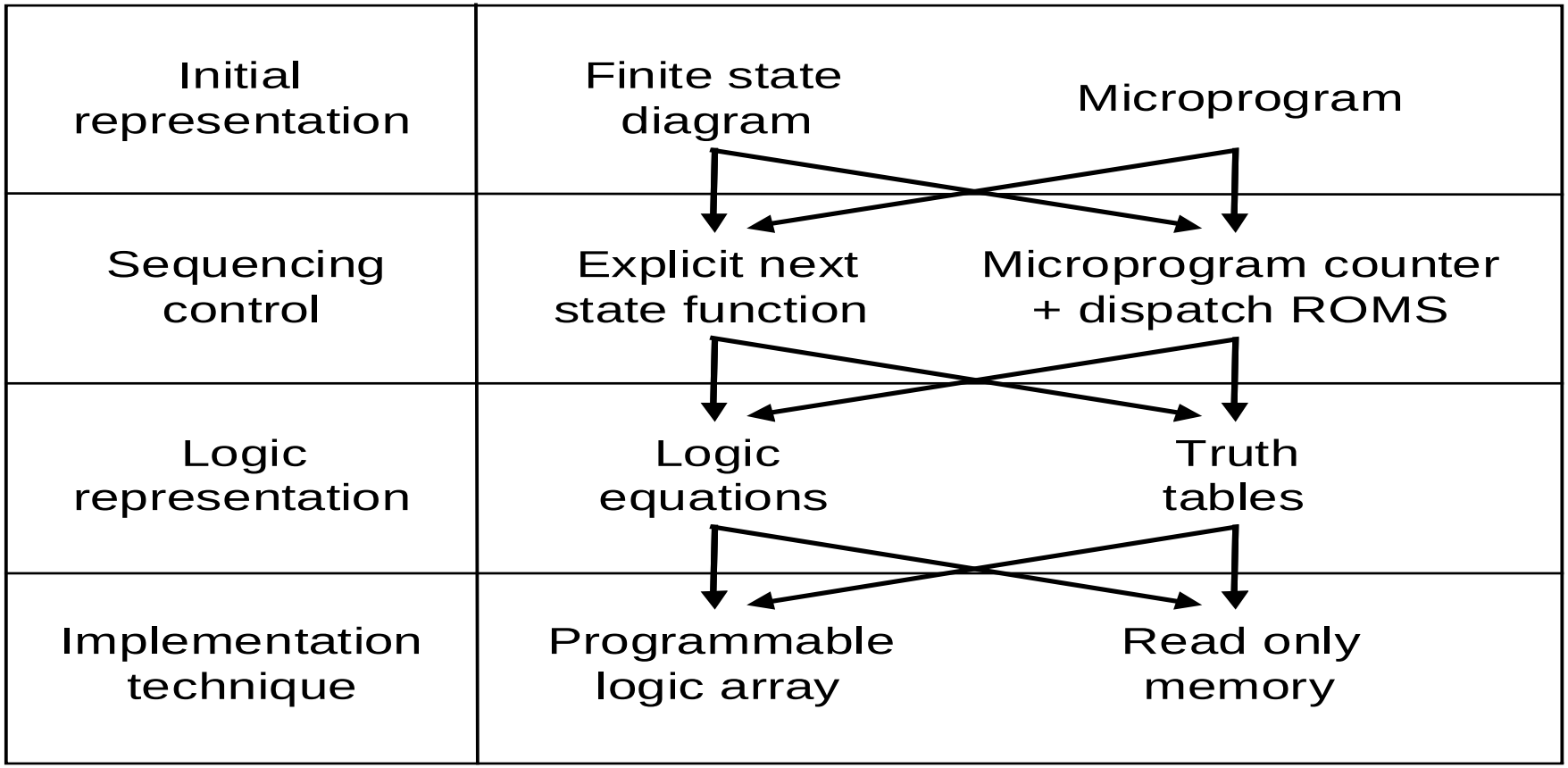


* Slide is courtesy of Dave Patterson

Detailed Control Specification

	State	Op field	Eq	Next	IR	PC en	PC sel	Ops A	Ops B	Exec Ex	Exec Sr	Exec ALU	Exec S	Mem R	Mem W	Mem M	Write-Back M-R	Write-Back Wr	Write-Back Dst
	0000	??????	?	0001	1														
	0001	BEQ	0	0011				1	1										
	0001	BEQ	1	0010				1	1										
	0001	R-type	x	0100				1	1										
	0001	orI	x	0110				1	1										
	0001	LW	x	1000				1	1										
	0001	SW	x	1011				1	1										
	0010	xxxxxx	x	0000		1	1												
	0011	xxxxxx	x	0000		1	0												
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	0			
	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				

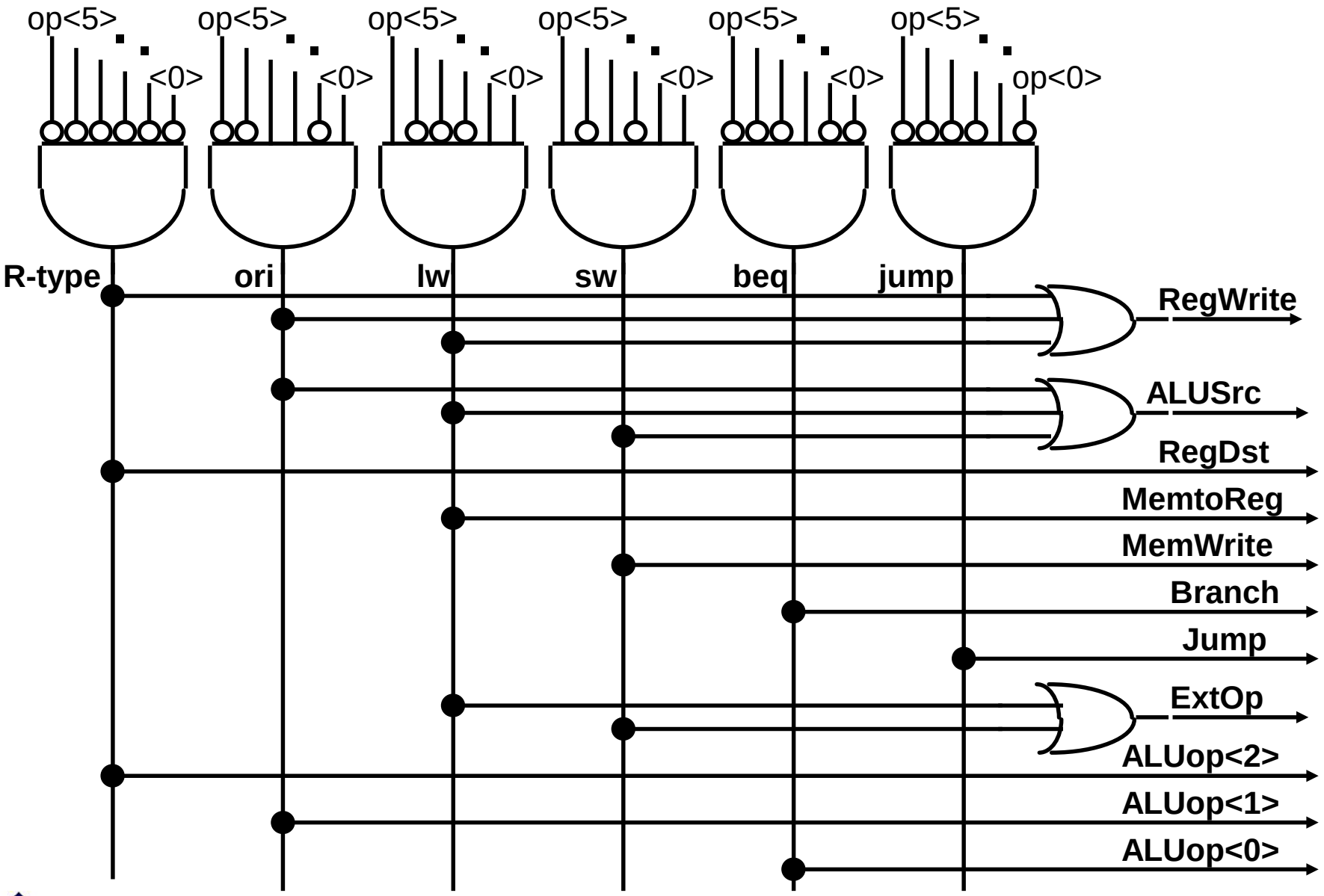
Overview of Control Design



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

* Slide is courtesy of Dave Patterson

Example: PLA Implementation Control



* Slide is courtesy of Dave Patterson

Micro-programming

- Control is the hard part of processor design
  - ➔ Datapath is fairly regular and well-organized
  - ➔ Memory is highly regular
  - ➔ Control is irregular and global
- Micro-programming:

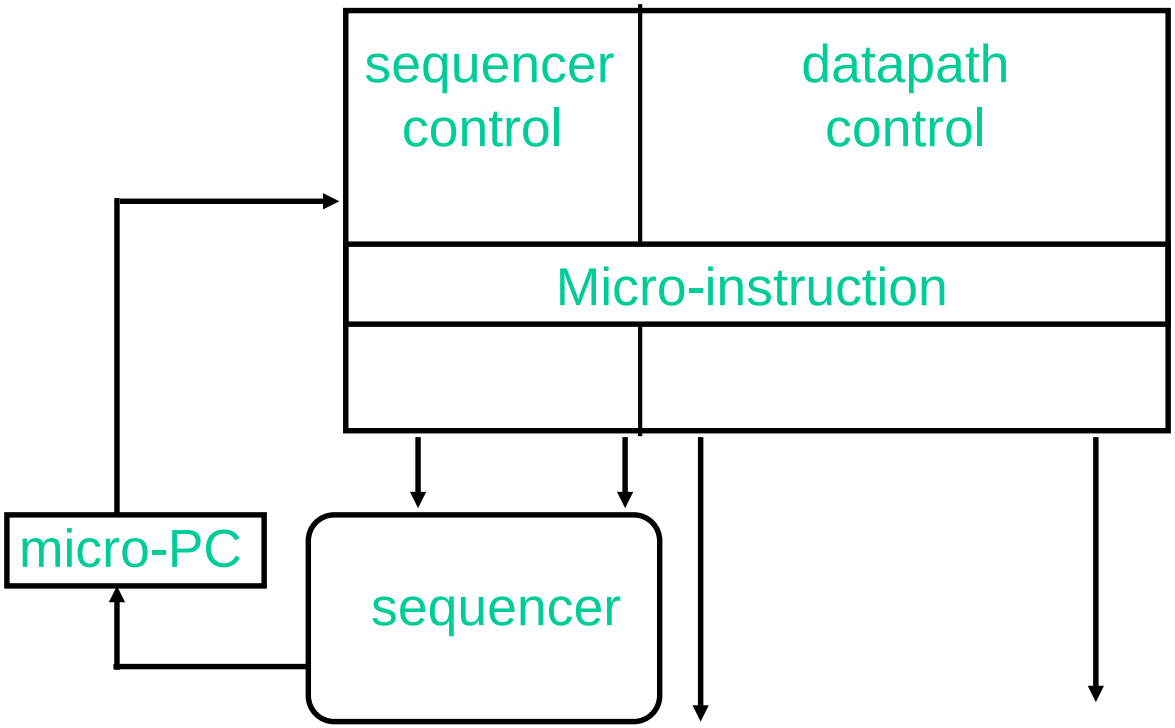
  A Particular Strategy for Implementing the Control Unit of a processor by "programming" at the level of register transfer operations
- Micro-architecture:

  Logical structure and functional capabilities of the hardware as seen by the micro-programmer
- *Historical Note:*
  - ➔ IBM 360 Series first to distinguish between architecture & organization
  - ➔ Same instruction set across wide range of implementations, each with different cost/performance

* Slide is courtesy of Dave Patterson

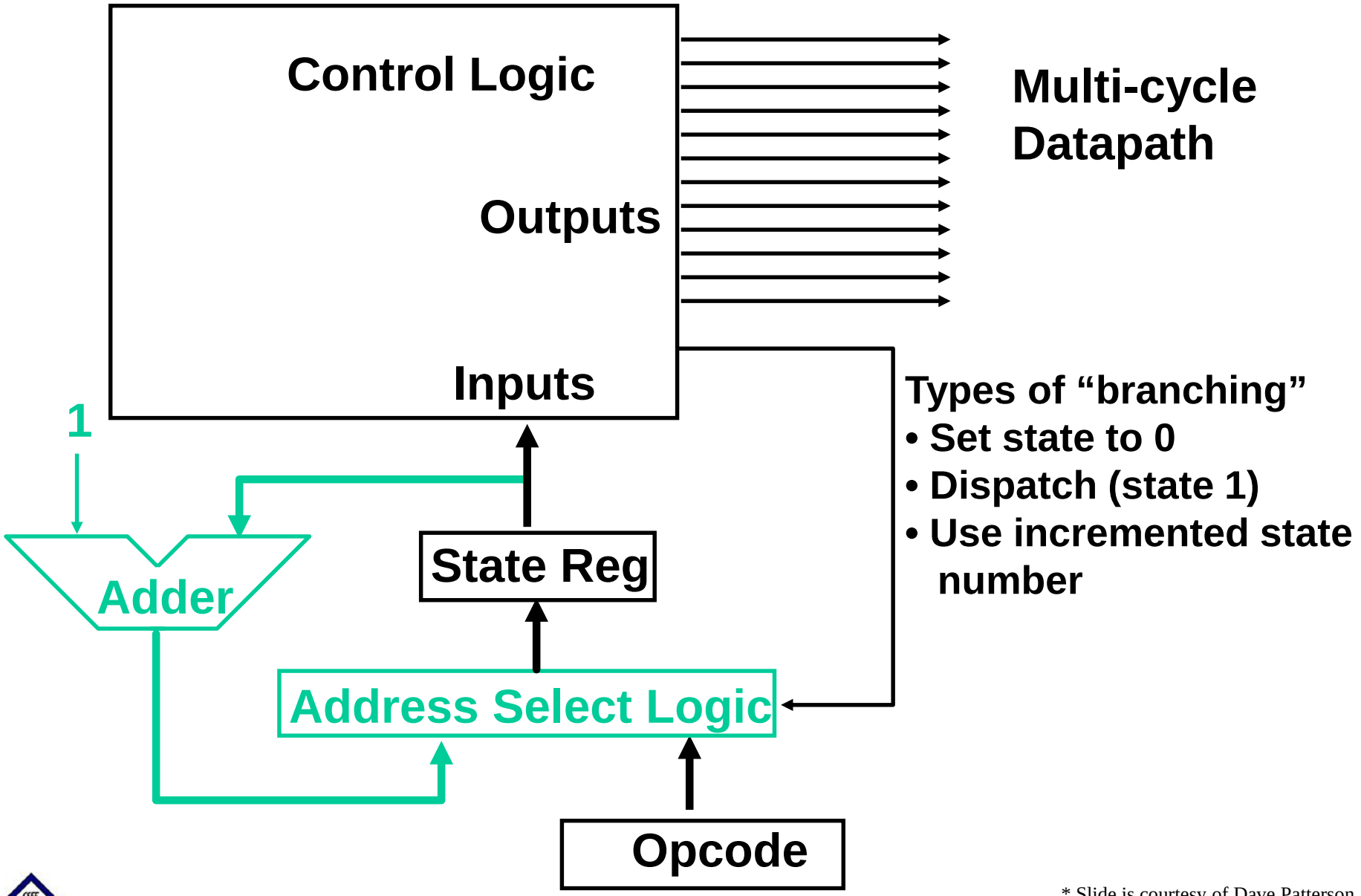
Micro-programmed Controller Design

- The state diagrams that define the controller for an instruction set processor are highly structured
- Use this structure to construct a simple “micro-sequencer”
- Control reduces to programming this very simple device
  - ➔ micro-programming



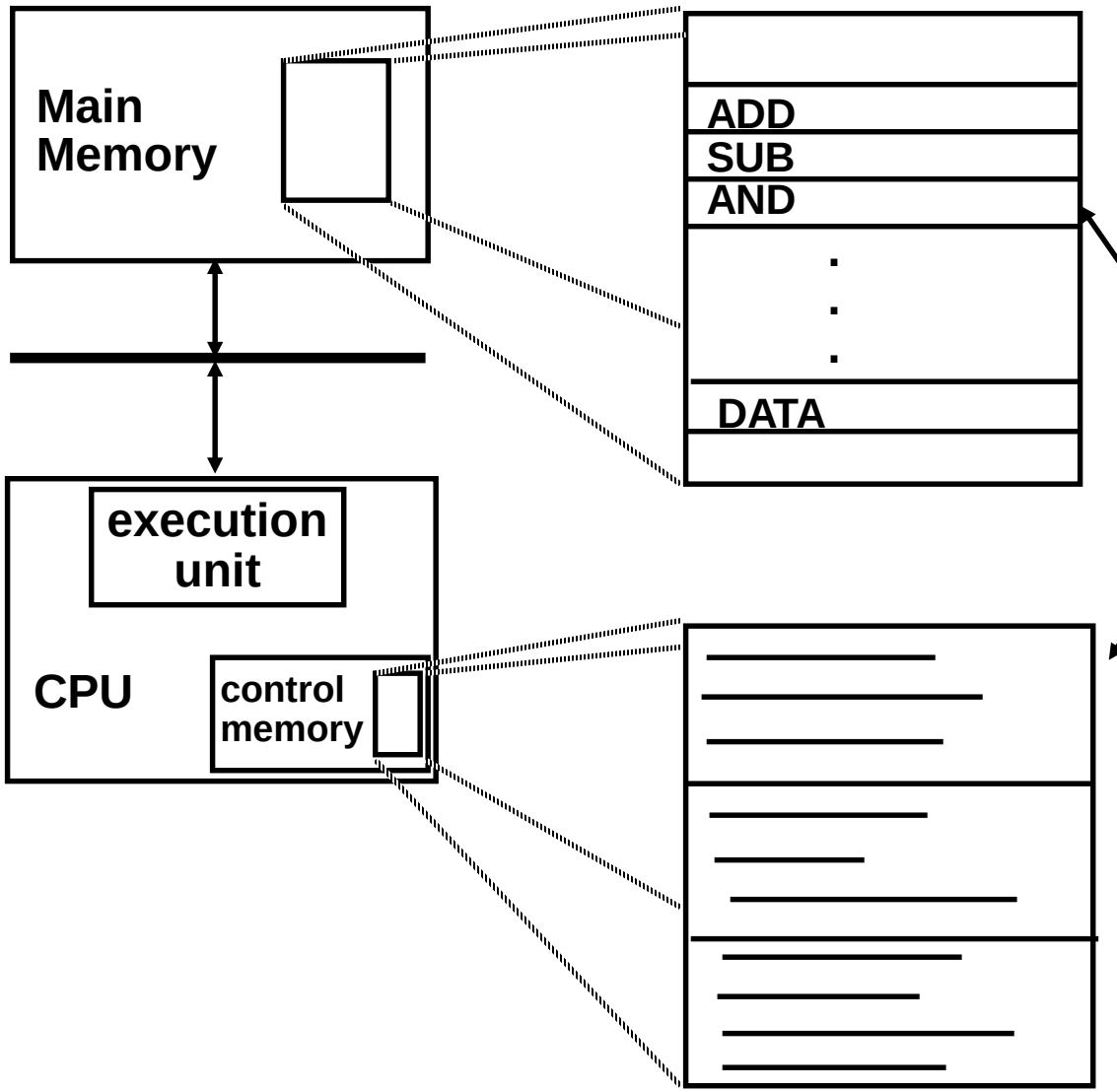
* Slide is courtesy of Dave Patterson

Sequencer-based control unit



* Slide is courtesy of Dave Patterson

“Micro-instruction” Interpretation



User program plus Data

this can change!

one of these is mapped into one of these

AND micro-sequence

e.g., Fetch
Calc. Operand Addr.
Fetch Operand(s)
Calculate
Save Answer(s)

* Slide is courtesy of Dave Patterson

Variations on Micro-programming

- "*Horizontal*" Micro-code
  - ➔ control field for each control point in the machine
  - ◭ more control over the potential parallelism of operations in the datapath
  - ◥ uses up lots of control store

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

- "*Vertical*" Micro-code
  - ➔ compact micro-instruction format for each class of micro-operation

branch:	μseq-op	μadd
execute:	ALU-op	A,B,R
memory:	mem-op	S, D

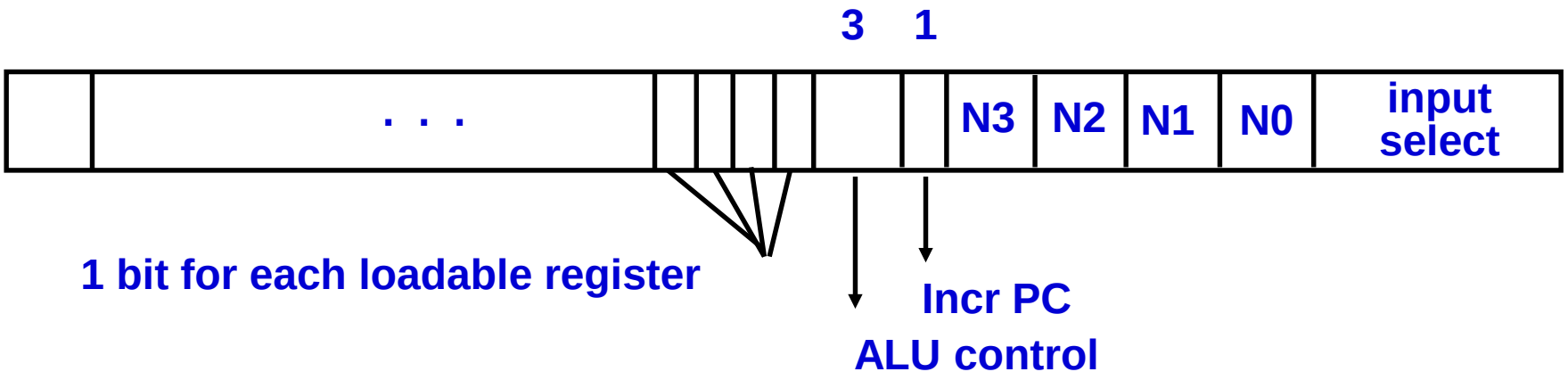
  - ➔ local decode to generate all control points
  - ◭ Easier to program, not very different from programming a RISC machine in assembly language
  - ◥ Extra level of decoding may slow the machine down

* Slide is courtesy of Dave Patterson

Example Of Horizontal Micro-programs

- Depending on bus organization, many potential control combinations simply wrong, i.e., implies transfers that can never happen at the same time
- Example:

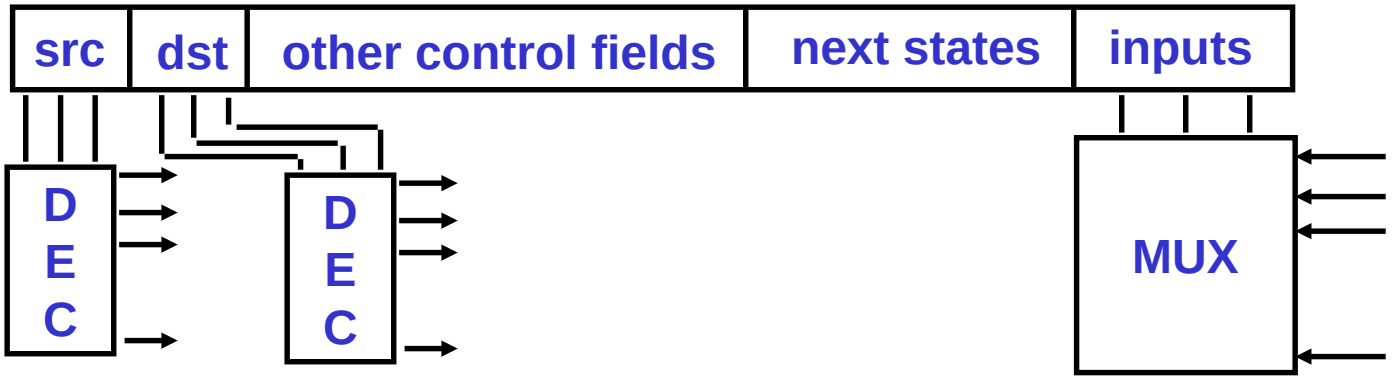
  mem_to_reg and ALU_to_reg should never happen simultaneously;
  ⇒ encode in single bit which is decoded rather than two separate bits
- Makes sense to encode fields to save ROM space



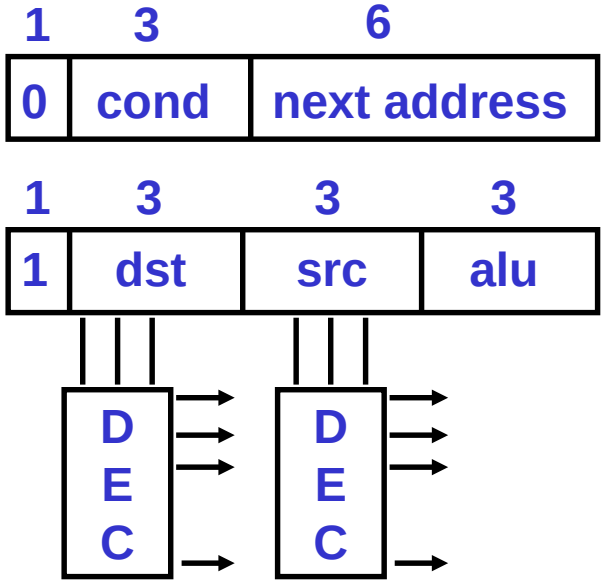
* Slide is courtesy of Dave Patterson

Vertical Micro-programming Format

- ***Single-format Micro-code:***



- ***Multi-format Micro-code:***



Branch Jump

Register transfer Operation

* Slide is courtesy of Dave Patterson

Designing a Micro-instruction Set

1) Start with list of control signals

2) Group together signals when makes sense (vs. random): called “fields”

3) Place fields in some logical order

(e.g., ALU operation & operands first and micro-instruction sequencing last)

4) Create a symbolic legend for the micro-instruction format, showing name of field values and how they set control signals

➔ Use computers to design computers

5) To minimize the width, encode operations that will never be used at the same time

* Slide is courtesy of Dave Patterson

1&2) list signals, grouped into fields

Single Bit Control

Signal name	Effect when deasserted	Effect when asserted
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
RegDst	Reg. dest. no. = rt	Reg. dest. no. = rd
TargetWrite	None	Target reg. = ALU
MemRead	None	Memory at address is read
MemWrite	None	Memory at address is written
IorD	Memory address = PC	Memory address = ALU
IRWrite	None	IR = Memory
PCWrite	None	PC = PCSource
PCWriteCond	None	IF ALUzero then PC = PCSource

Multiple Bit Control

Signal name	Value	Effect
ALUOp	00	ALU adds
	01	ALU subtracts
	10	ALU does function code
	11	ALU does logical OR
ALUSelB	000	2nd ALU input = Reg[rt]
	001	2nd ALU input = 4
	010	2nd ALU input = sign extended IR[15-0]
	011	2nd ALU input = sign extended, shift left 2 IR[15-0]
	100	2nd ALU input = zero extended IR[15-0]
PCSource	00	PC = ALU
	01	PC = Target
	10	PC = PC+4[29-26] : IR[25–0] << 2

* Slide is courtesy of Dave Patterson

3) Encoded fields

Field Name	*Width*		*Control Signals Set*
	wide	narrow	
ALU Control	4	2	ALUOp
SRC1	2	1	ALUSelA
SRC2	5	3	ALUSelB
ALU Destination	6	4	RegWrite, MemtoReg, RegDst, TargetWr.
Memory	4	3	MemRead, MemWrite, IorD
Memory Register	1	1	IRWrite
PCWrite Control	5	4	PCWrite, PCWriteCond, PCSource
Sequencing	3	2	AddrCtl
Total width	30	20	bits

* Slide is courtesy of Dave Patterson

4) 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]
ALU destination	Target	Target = ALUout
	rd	Reg[rd] = ALUout
Memory	Read PC	Read memory using PC
	Read ALU	Read memory using ALU output
	Write ALU	Write memory using ALU output
Memory register	IR	IR = Mem
	Write rt	Reg[rt] = Mem
	Read rt	Mem = Reg[rt]
	…….	

* Slide is courtesy of Dave Patterson

5) Create Micro-program

Label	*ALU*	*SRC1*	*SRC2*	*ALUDest.*	*Memory*	*Mem. Reg.*	*PC Write*	*Sequencing*
Fetch	Add	PC	4		Read PC	IR	ALU	Seq
	Add	PC	Extshft	Target				Dispatch
LW	Add	rs	Extend					Seq
					Read ALU	Write rt		Fetch
SW	Add	rs	Extend					Seq
					Write ALU	Read rt		Fetch
Rtype	Func	rs	rt					Seq
				rd				Fetch
BEQ1	Subt.	rs	rt				Target–cond.	Fetch
JUMP1							jump address	Fetch
ORI	Or	rs	Extend0					Seq
				rd				Fetch

Micro-programming Pros and Cons

Advantages:

- 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
- Scalable for 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

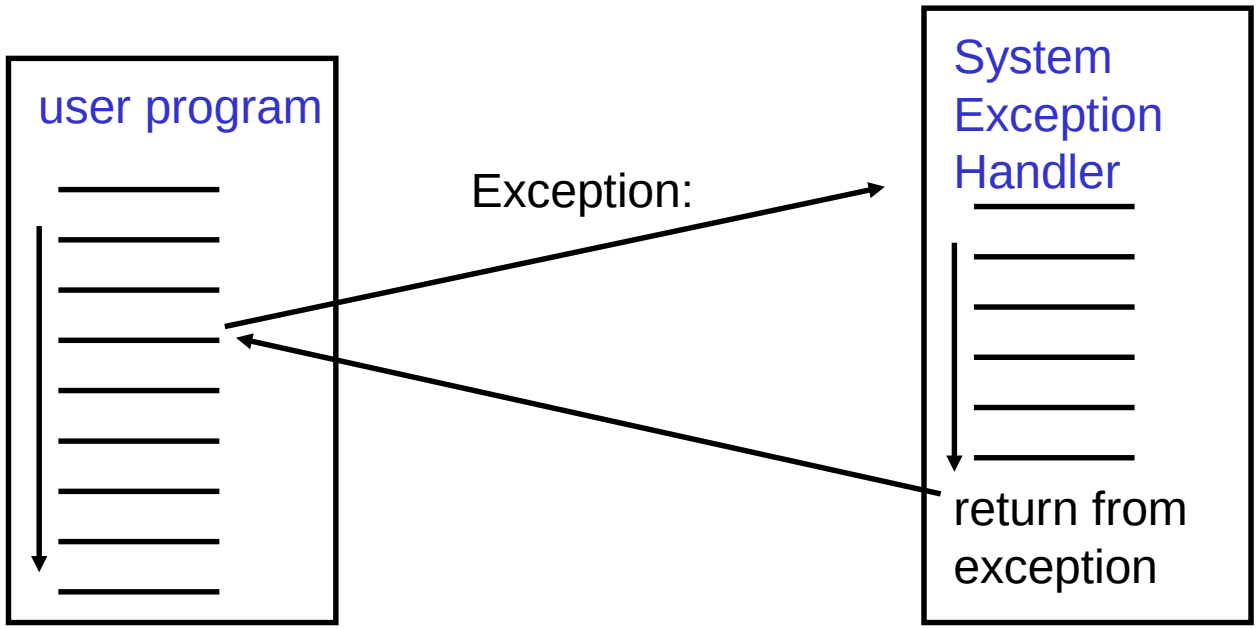
Disadvantages

- Costly to implement
- Slow

* Slide is courtesy of Dave Patterson

Exceptions

- Exception = unplanned control transfer
  - ➔ system takes action to handle the exception
    - must record the address of the offending instruction
  - ➔ returns control to user
  - ➔ must save & restore user state
- Allows construction of a “user virtual machine”



normal control flow:
sequential, jumps, branches, calls, returns

* Slide is courtesy of Dave Patterson

Types of Exceptions

- ***Interrupts***
  - ➔ caused by external events
  - ➔ asynchronous to program execution
  - ➔ may be handled between instructions
  - ➔ simply suspend and resume user program
- ***Traps***
  - ➔ caused by internal events
    - exceptional conditions (overflow)
    - errors (parity)
    - faults (non-resident page)
  - ➔ synchronous to program execution
  - ➔ condition must be remedied by the handler
  - ➔ instruction may be retried or simulated and program continued or program may be aborted

* Slide is courtesy of Dave Patterson

MIPS Convention

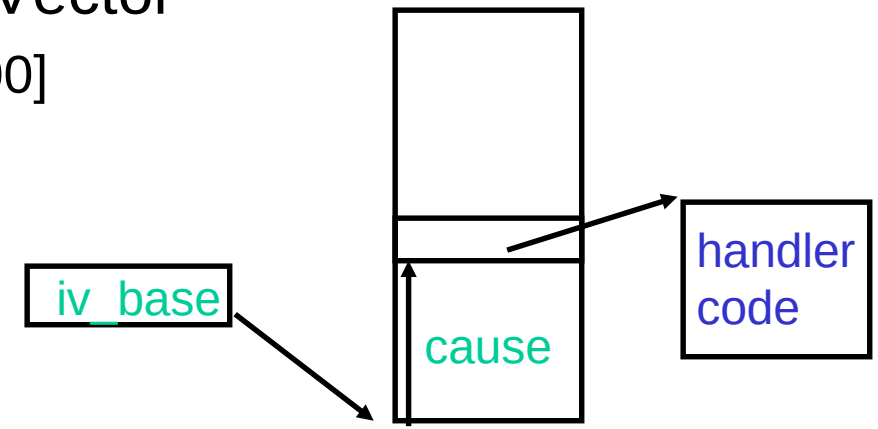
- Exception means any unexpected change in control flow, without distinguishing internal or external
- Use the term interrupt only when the event is externally caused
- MIPS architecture defines the instruction as having no effect if the instruction causes an exception

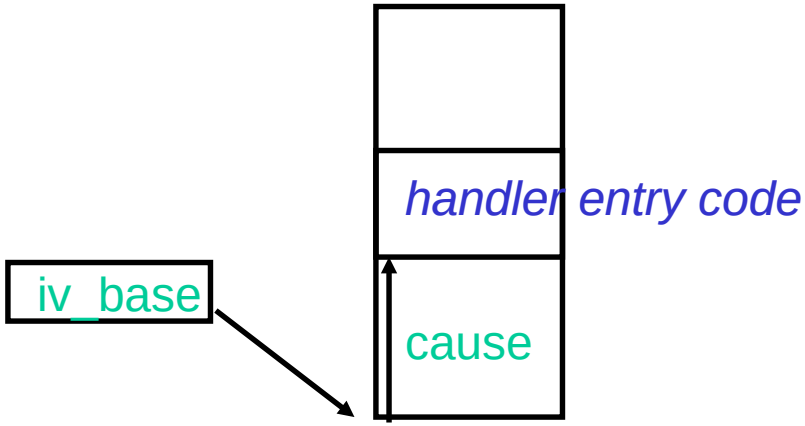
Type of event	*From where?*	*MIPS terminology*
I/O device request	External	Interrupt
Invoke OS from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or Interrupt

* Slide is courtesy of Dave Patterson

Addressing Exception Handlers

- Traditional Approach: Interrupt Vector
  - ➔ PC ⇐ MEM[IV_base + cause || 00]
  - ➔ 370, 68000, Vax, 80x86, . . .
- RISC Handler Table
  - ➔ PC ⇐ IT_base + cause || 0000
  - ➔ saves state and jumps
  - ➔ Sparc, PA, M88K, . . .
- MIPS Approach: fixed entry
  - ➔ PC ⇐ EXC_addr
  - ➔ Actually very small table
    - RESET entry
    - TLB
    - other





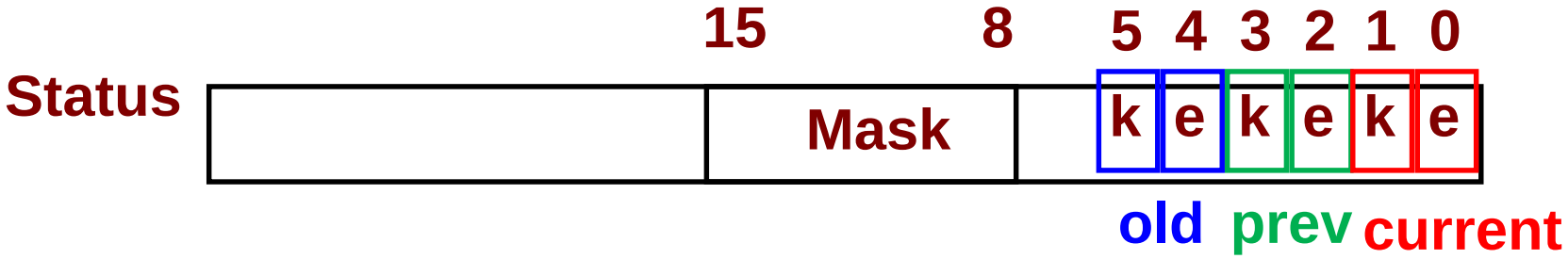
* Slide is courtesy of Dave Patterson

MIPS support for Exceptions

- EPC: a 32-bit register used to hold the address of the affected instruction
- Cause: a register used to record the cause of the exception
- BadVAddr: register contained memory address at which memory reference occurred
- Status: interrupt mask and enable bits
- Control signals to write EPC , Cause, BadVAddr, and Status
- Be able to write exception address into PC, increase MUX to add as input $01000000\ 00000000\ 00000000\ 01000000_{two}$ ($8000\ 0080_{hex}$)
- May have to undo PC = PC + 4, since want EPC to point to offending instruction (not its successor); PC = PC - 4

* Slide is courtesy of Dave Patterson

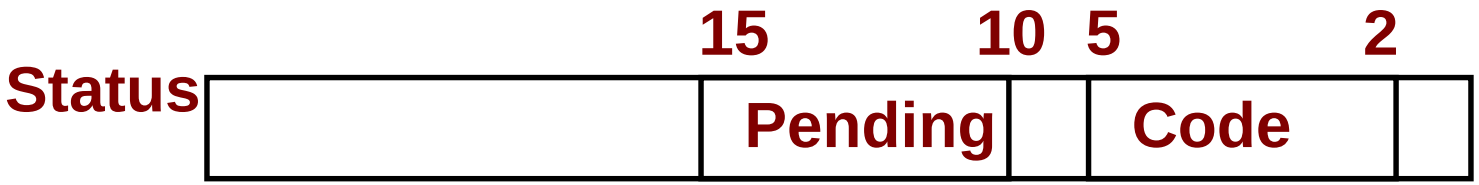
MIPS Status register



- Mask = 1 bit for each of 5 hardware and 3 software interrupt levels
  - ➔ 1 => enables interrupts
  - ➔ 0 => disables interrupts
- k = kernel/user (important to know setting after interrupt is serviced)
  - ➔ 0 => was in the kernel when interrupt occurred
  - ➔ 1 => was running user mode
- e = interrupt enable
  - ➔ 0 => interrupts were disabled
  - ➔ 1 => interrupts were enabled
- When interrupt occurs, 6 LSB shifted left 2 bits, setting 2 LSB to 0
  - ➔ run in kernel mode with interrupts disabled
  - ➔ Enable handling nested exceptions (when allowed)

* Slide is courtesy of Dave Patterson

MIPS Cause register



- Pending interrupt 5 H/W levels: bit set if interrupt occurs but not yet serviced
  - ➔ handles cases when more than one interrupt occurs at the same time, or while records interrupt requests when interrupts are disabled
- Exception Code encodes reasons for interrupt
  - ➔ 0 (INT) ⇒ external interrupt
  - ➔ 4 (ADDRL) ⇒ address error exception (load or instr fetch)
  - ➔ 5 (ADDRS) ⇒ address error exception (store)
  - ➔ 6 (IBUS) ⇒ bus error on instruction fetch
  - ➔ 7 (DBUS) ⇒ bus error on data fetch
  - ➔ 8 (Syscall) ⇒ Syscall exception
  - ➔ 9 (BKPT) ⇒ Breakpoint exception
  - ➔ 10 (RI) ⇒ Reserved Instruction exception
  - ➔ 12 (OVF) ⇒ Arithmetic overflow exception

* Slide is courtesy of Dave Patterson

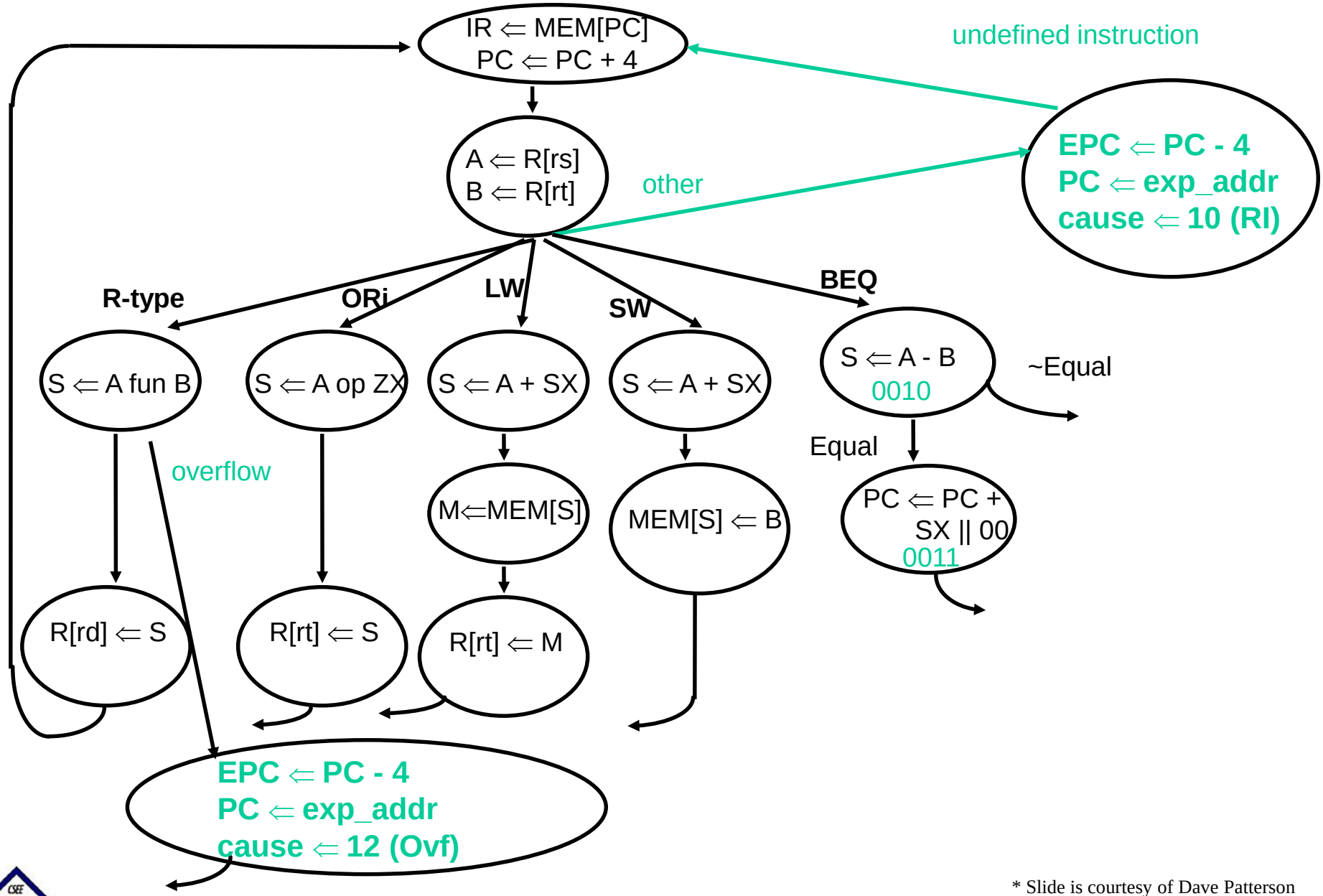
Detecting Exceptions

- Undefined Instruction–detected when no next state is defined from state 1 for the op value.
  - ➔ We handle this exception by defining the next state value for all op values other than lw, sw, 0 (R-type), jmp, beq, and ori as new state
  - ➔ Shown symbolically using “other” to indicate that the op field does not match any of the op-codes that label arcs out of state 1
- Arithmetic overflow: logic is included in the ALU to detect overflow, and a signal called *Overflow* is provided as an output from the ALU
- Challenge in designing control of a real machine is to handle different interactions between instructions and other exception-causing events such that control logic remains small and fast

Complex interactions makes control unit the most challenging aspect of h/w design

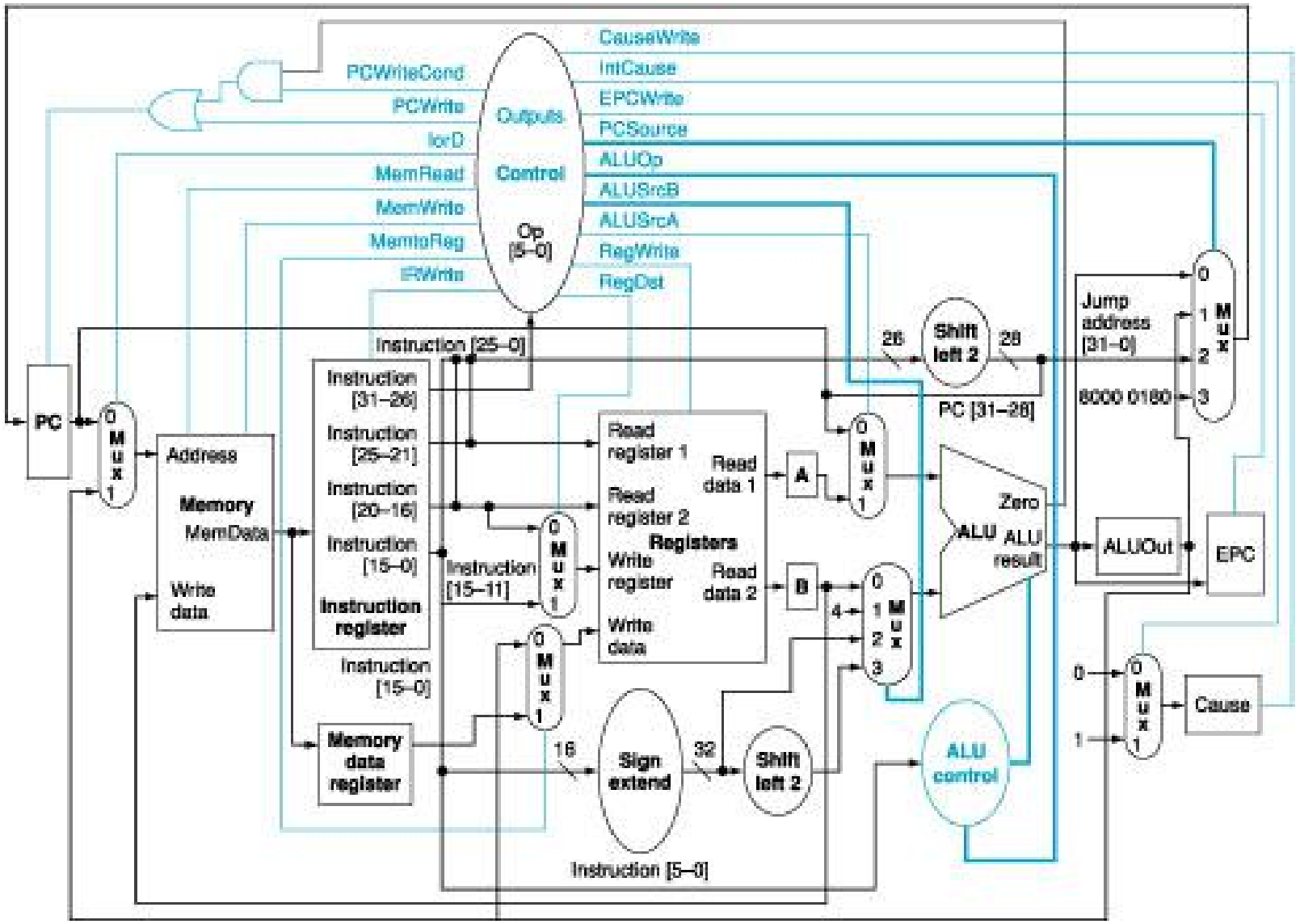
* Slide is courtesy of Dave Patterson

Modification to the Control Specification



* Slide is courtesy of Dave Patterson

Extended Datapath/Contol



Conclusion

❑ *Summary*

➔ Micro-programmed control
- PLA versus ROM based control unit design
- Horizontal versus vertical micro-coding
- Designing a micro-instruction set

➔ Processor exceptions
- Exceptions are the hardest part of control
- MIPS interrupts and exceptions support
- Detecting exceptions by the control unit

❑ *Next Lecture*

➔ An overview of Pipelining

➔ A pipelined datapath

➔ Pipelined control

Read sections D.5 in 5th d Ed., or C.12 & D.4 in 4th Ed. of textbook