

**Lecture 28 –
 Single Cycle CPU Control II**



Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

DARPA \$s drying up...ouch! ⇒

“I’m worried and depressed,”

– David Patterson, president of the ACM.

There is a significant shift of \$ from “Blue Sky” research to military contractors. This is

a significant shift, mostly for the worse.

www.nytimes.com/2005/04/02/technology/02darpa.html?_r=1



Review: Single cycle datapath

◦ **5 steps to design a processor**

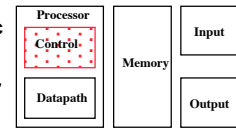
1. Analyze instruction set => datapath requirements
2. **Select** set of datapath components & establish clock methodology
3. **Assemble** datapath meeting the requirements
4. **Analyze** implementation of each instruction to determine setting of control points that effects the register transfer.
5.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

◦ **Control is the hard part**

◦ **MIPS makes that easier**

- Instructions same size
- Source registers always in same place
- Immediates same size, location

Operations always on registers/immediates



A Summary of the Control Signals (2/2)

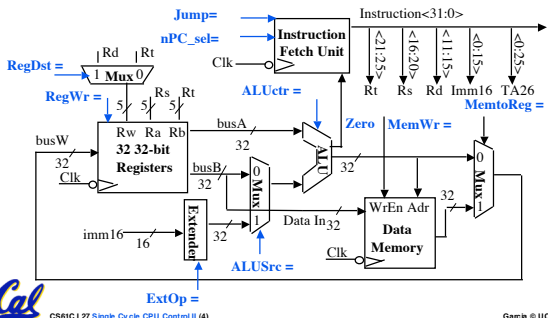
See Appendix A	func	00 0000	10 0010	We Don't Care :-)				00 0010
op		00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		add	sub	ori	lw	sw	beq	jump
RegDst		1	1	0	0	x	x	x
ALUSrc		0	0	1	1	1	0	x
MemoReg		0	0	0	1	x	x	x
RegWrite		1	1	1	1	0	0	0
MemWrite		0	0	0	0	1	0	0
nPCsel		0	0	0	0	0	1	0
Jump		0	0	0	0	0	0	1
ExtOp		x	x	0	1	1	x	x
ALUctr<2:0>		Add	Subtract	Or	Add	Subtract	x	x

R-type	op	rs	rt	rd	shamt	funct	add, sub	
I-type	op	rs	rt	immediate			ori, lw, sw, beq	
J-type	op	target address						jump



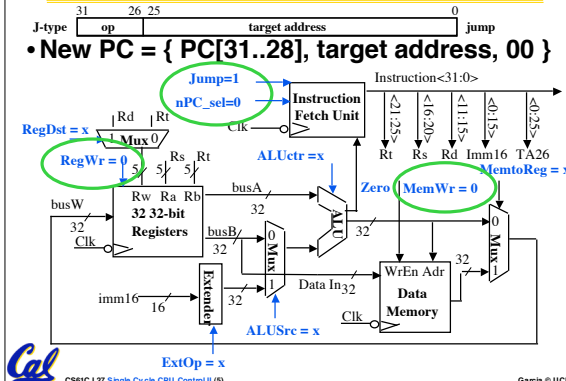
The Single Cycle Datapath during Jump

• **New PC = { PC[31..28], target address, 00 }**



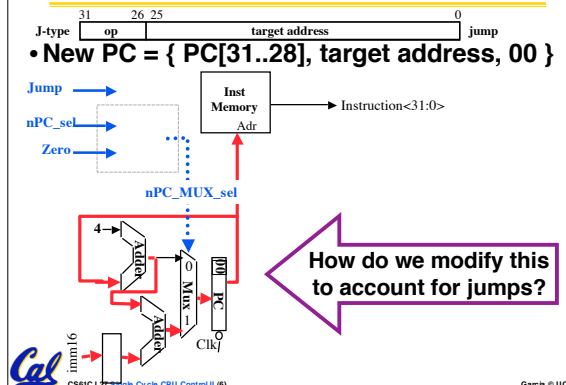
The Single Cycle Datapath during Jump

• **New PC = { PC[31..28], target address, 00 }**



Instruction Fetch Unit at the End of Jump

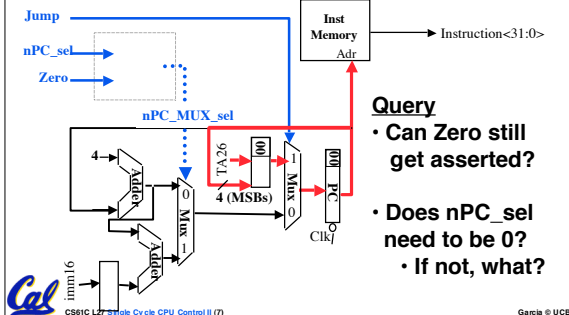
• **New PC = { PC[31..28], target address, 00 }**



How do we modify this to account for jumps?

Instruction Fetch Unit at the End of Jump

J-type op target address jump
 • New PC = { PC[31..28], target address, 00 }

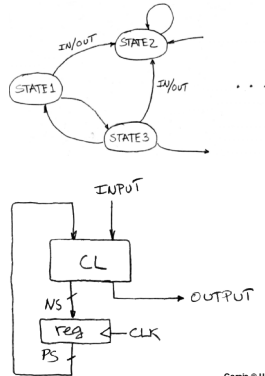


Build CL to implement Jump on paper now



Review: Finite State Machine (FSM)

- **States** represent possible output values.
- **Transitions** represent changes between states based on inputs.
- **Implement** with CL and clocked register feedback.



Finite State Machines extremely useful!

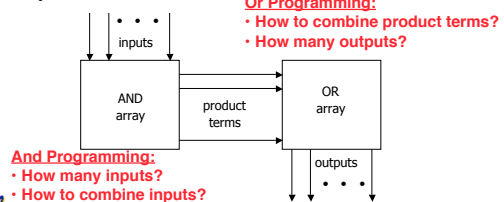
- They define
 - How **output signals** respond to input signals and previous state.
 - How we **change states** depending on input signals and previous state
- The output signals could be our familiar **control signals**
 - Some control signals **may only depend on CL, not on state at all...**
- We could implement very detailed FSMs w/**Programmable Logic Arrays**

Taking advantage of sum-of-products

- Since sum-of-products is a convenient notation and way to think about design, offer hardware building blocks that match that notation
- One example is **Programmable Logic Arrays (PLAs)**
- Designed so that can select (program) ands, ors, complements **after** you get the chip
 - Late in design process, fix errors, figure out what to do later, ...

Programmable Logic Arrays

- Pre-fabricated building block of many AND/OR gates
 - “Programmed” or “Personalized” by making or breaking connections among gates
 - Programmable array block diagram for sum of products form



Enabling Concept

• Shared product terms among outputs

example:

$$\begin{aligned}
 F_0 &= A + B'C \\
 F_1 &= A'C + AB \\
 F_2 &= B'C + AB \\
 F_3 &= B'C + A
 \end{aligned}$$

input side: 3 inputs
 1 = uncomplemented in term
 0 = complemented in term
 - = does not participate

personality matrix

Product term	inputs	outputs
term	A B C	F0 F1 F2 F3
AB	1 1 -	0 1 1 0
B'C	- 0 1	0 0 0 1
AC'	1 - 0	0 1 0 0
B'C'	- 0 0	1 0 1 0
A	1 - -	1 0 0 1

output side: 4 outputs
 1 = term connected to output
 0 = no connection to output

reuse of terms;
 5 product terms

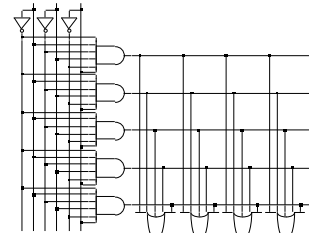


CS61C L27 Single Cycle CPU Control II (19)

Garcia © UCB

Before Programming

• All possible connections available before "programming"



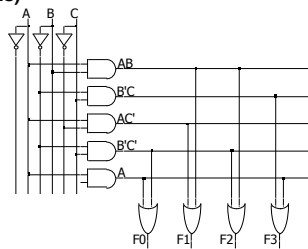
CS61C L27 Single Cycle CPU Control II (18)

Garcia © UCB

After Programming

• Unwanted connections are "blown"

- Fuse (normally connected, break unwanted ones)
- Anti-fuse (normally disconnected, make wanted connections)



CS61C L27 Single Cycle CPU Control II (17)

Garcia © UCB

Alternate Representation

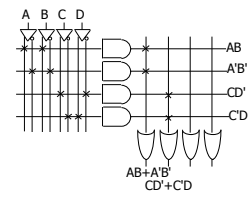
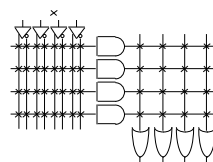
• Short-hand notation--don't have to draw all the wires

- X Signifies a connection is present and perpendicular signal is an input to gate

notation for implementing

$$F_0 = AB + A'B'$$

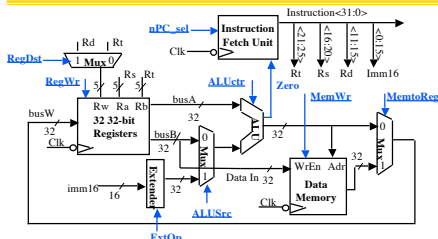
$$F_1 = C'D' + C'D$$



CS61C L27 Single Cycle CPU Control II (18)

Garcia © UCB

Peer Instruction



- MemToReg='x' & ALUctr='sub'. **SUB** or **BEQ**?
- ALUctr='add'. Which 1 signal is different for all 3 of: ADD, LW, & SW? **RegDst** or **ExtOp**?
- "Don't Care" signals are useful because we can simplify our PLA personality matrix. **F / T**?

ABC
1: S R F
2: S R T
3: S E F
4: S E T
5: B R F
6: B R T
7: B E F
8: B E T



CS61C L27 Single Cycle CPU Control II (19)

Garcia © UCB

And in Conclusion... Single cycle control

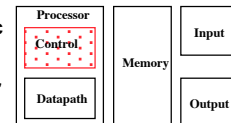
• 5 steps to design a processor

1. Analyze instruction set => datapath requirements
2. **Select** set of datapath components & establish clock 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

• Control is the hard part

• MIPS makes that easier

- Instructions same size
- Source registers always in same place
- immediates same size, location



CS61C L27 Single Cycle CPU Control II (20)

Garcia © UCB

Operations always on registers/immediates