inst.eecs.berkeley.edu/~cs61c

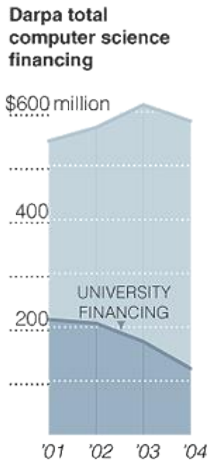
CS61C : Machine Structures

Lecture 28 – Single Cycle CPU Control II

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

DARPA $s drying up…ouch! ⇒

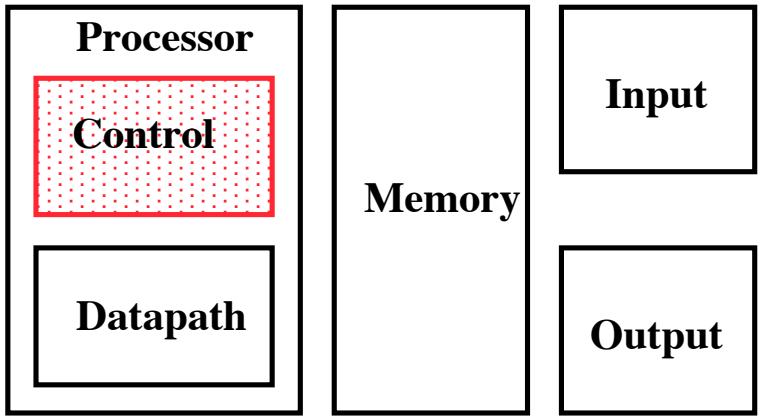


"I'm worried and depressed," – David Patterson, president of the ACM. There is a significant shift of $ from "Blue Sky" research to military contractors. This is a significant shift, mostly for the worse.

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

Review: Single cycle datapath

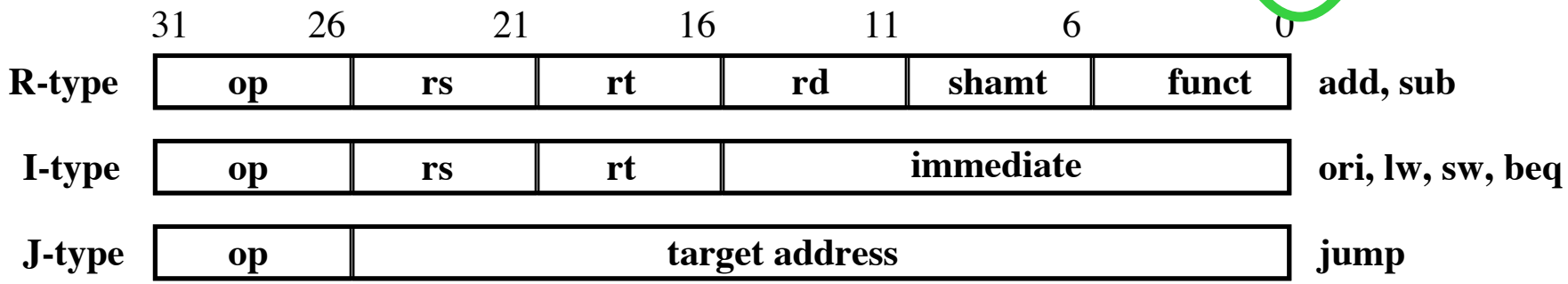
- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Assemble datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that 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)

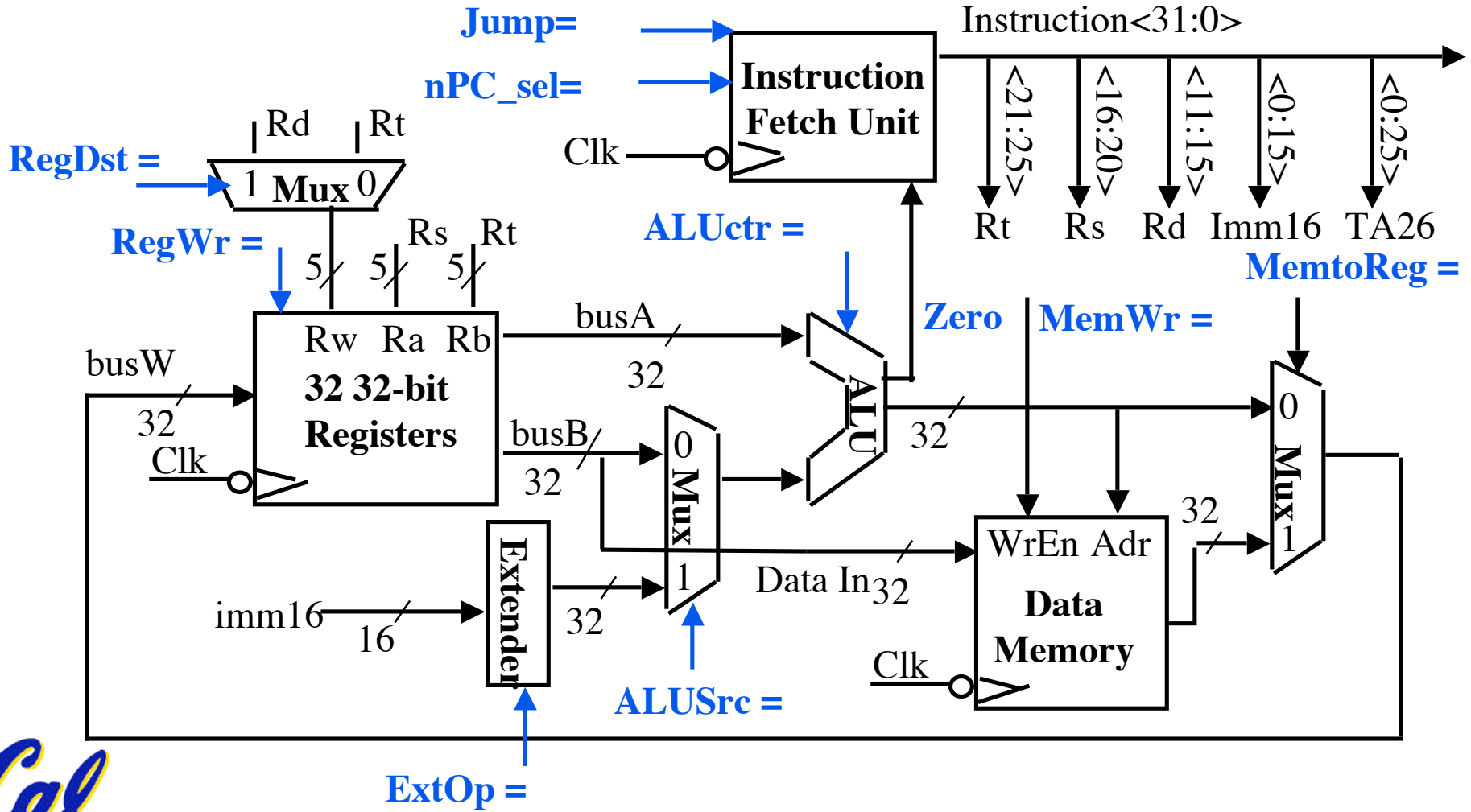
See Appendix A

	func	10 0000	10 0010	We Don't Care :-)				
	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
MemtoReg		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	Add	Subtract	x



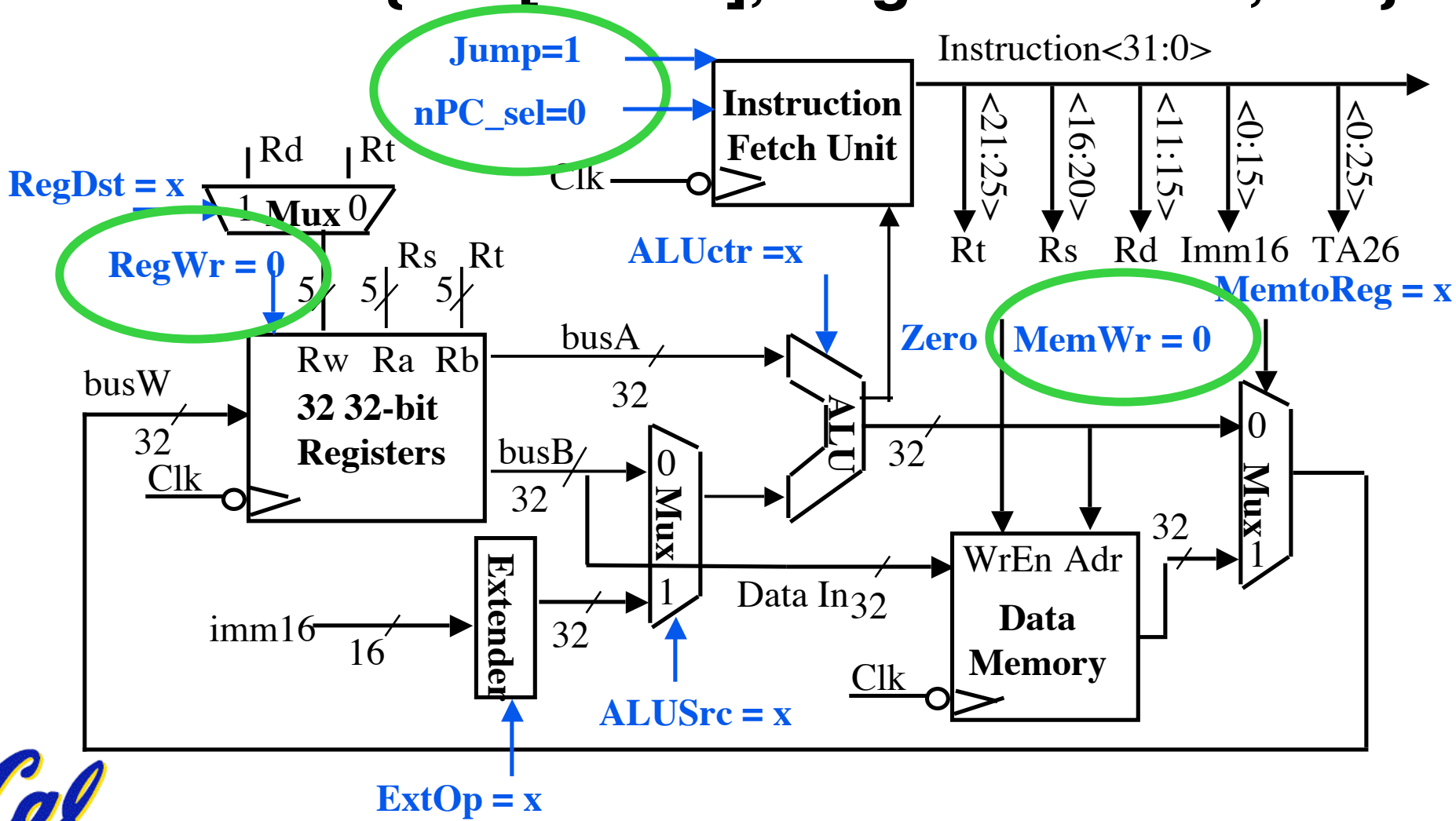
The Single Cycle Datapath during Jump

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



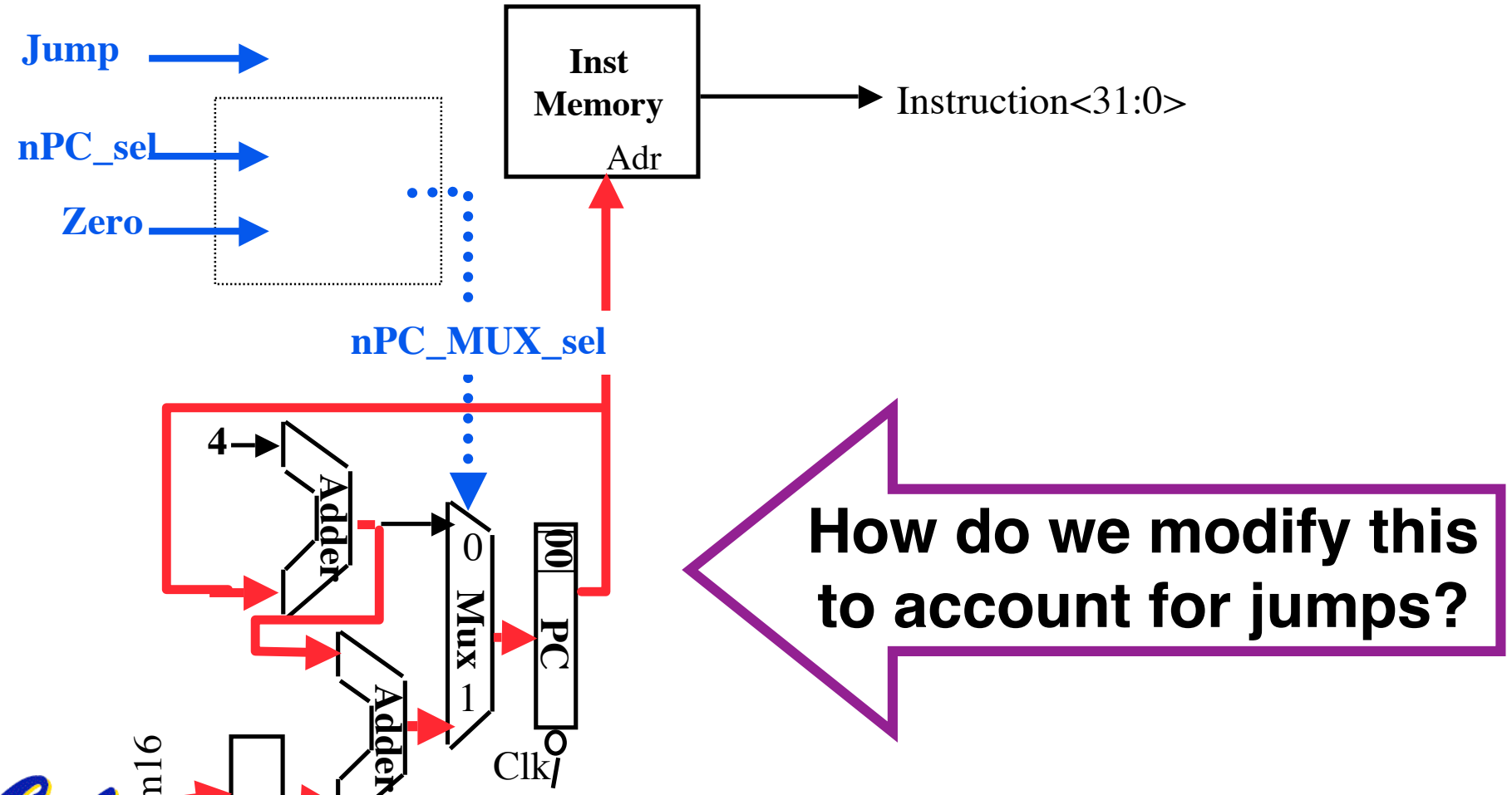
The Single Cycle Datapath during Jump

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



Instruction Fetch Unit at the End of Jump

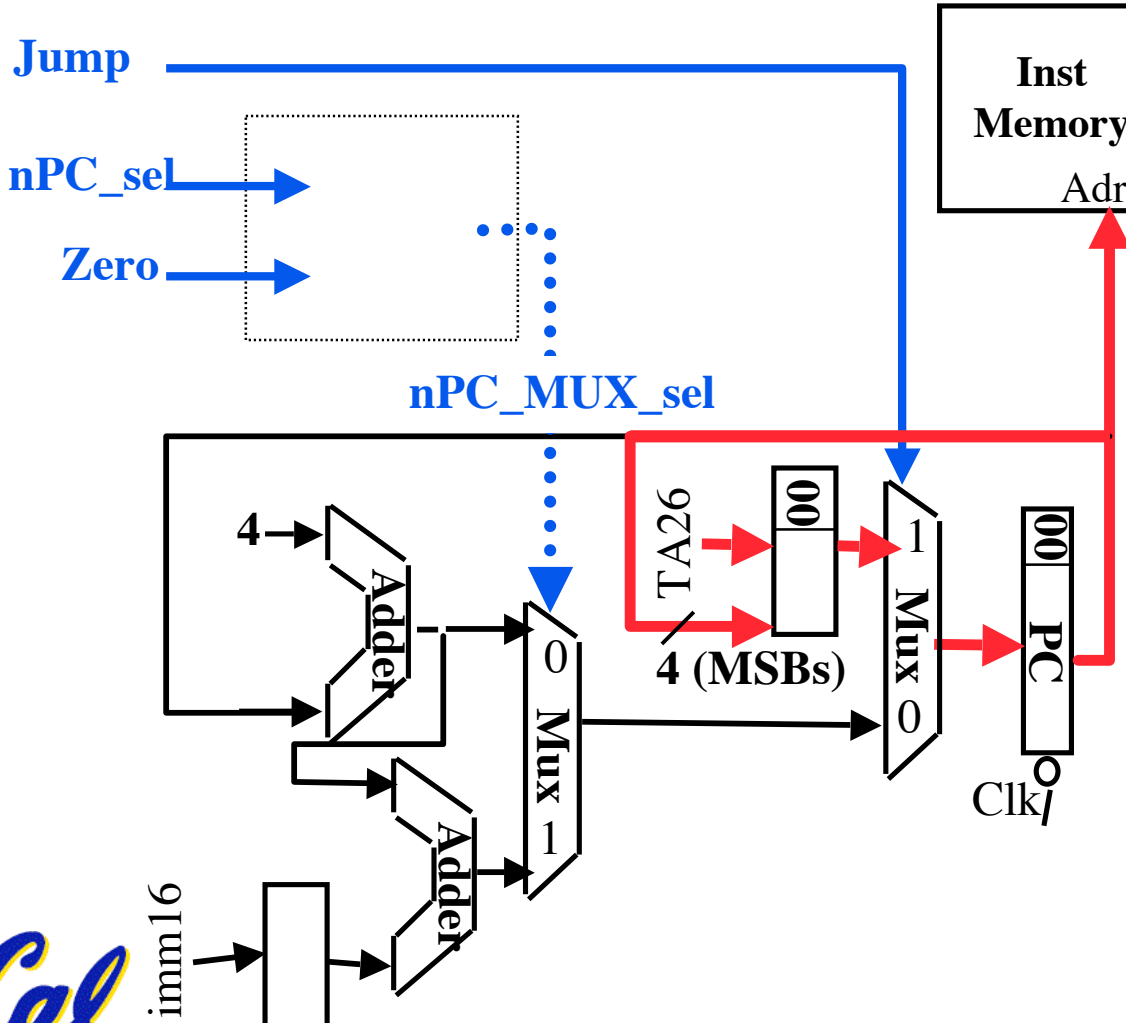
- New PC = { PC[31..28], target address, 00 }



How do we modify this to account for jumps?

Instruction Fetch Unit at the End of Jump

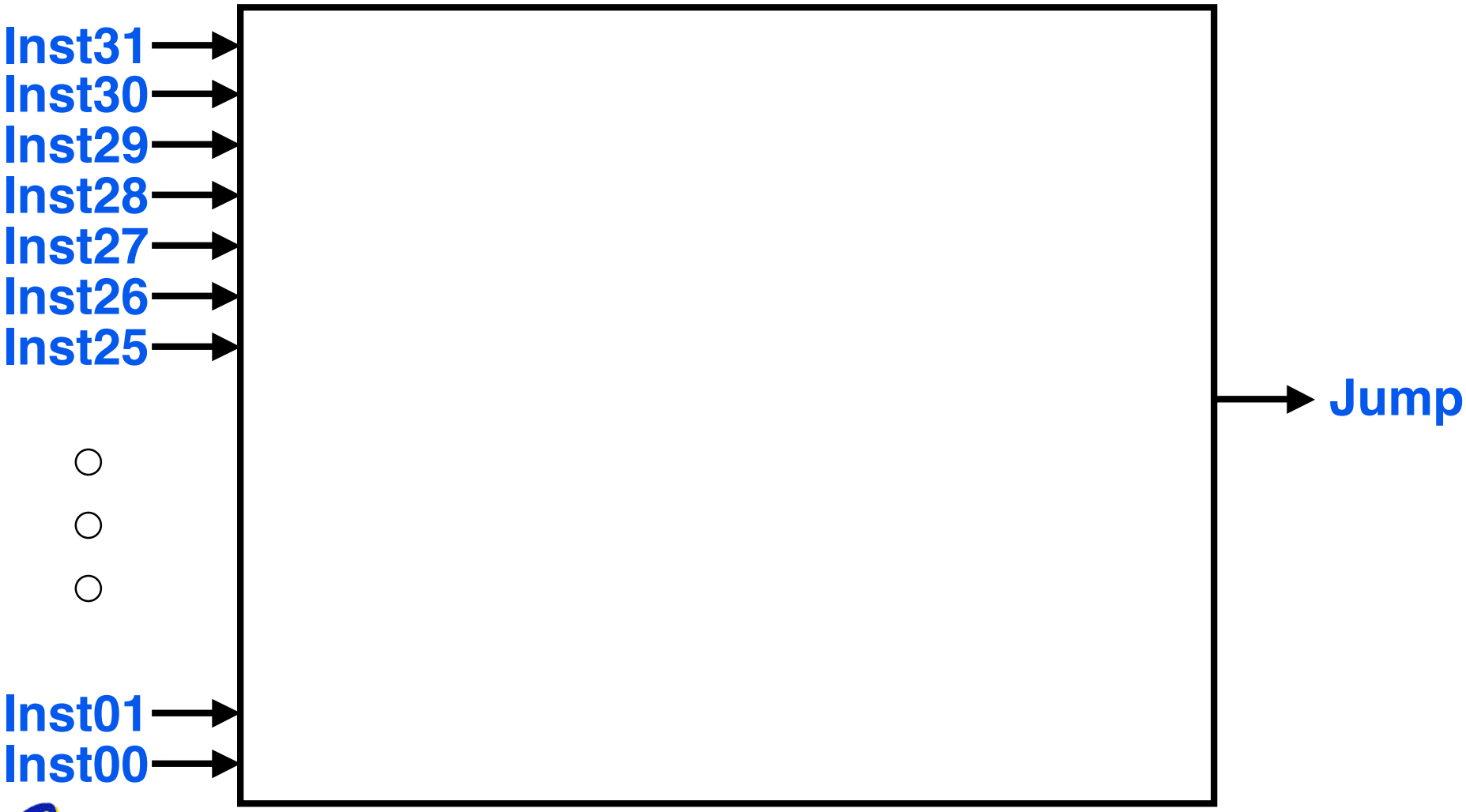
- New PC = { PC[31..28], target address, 00 }



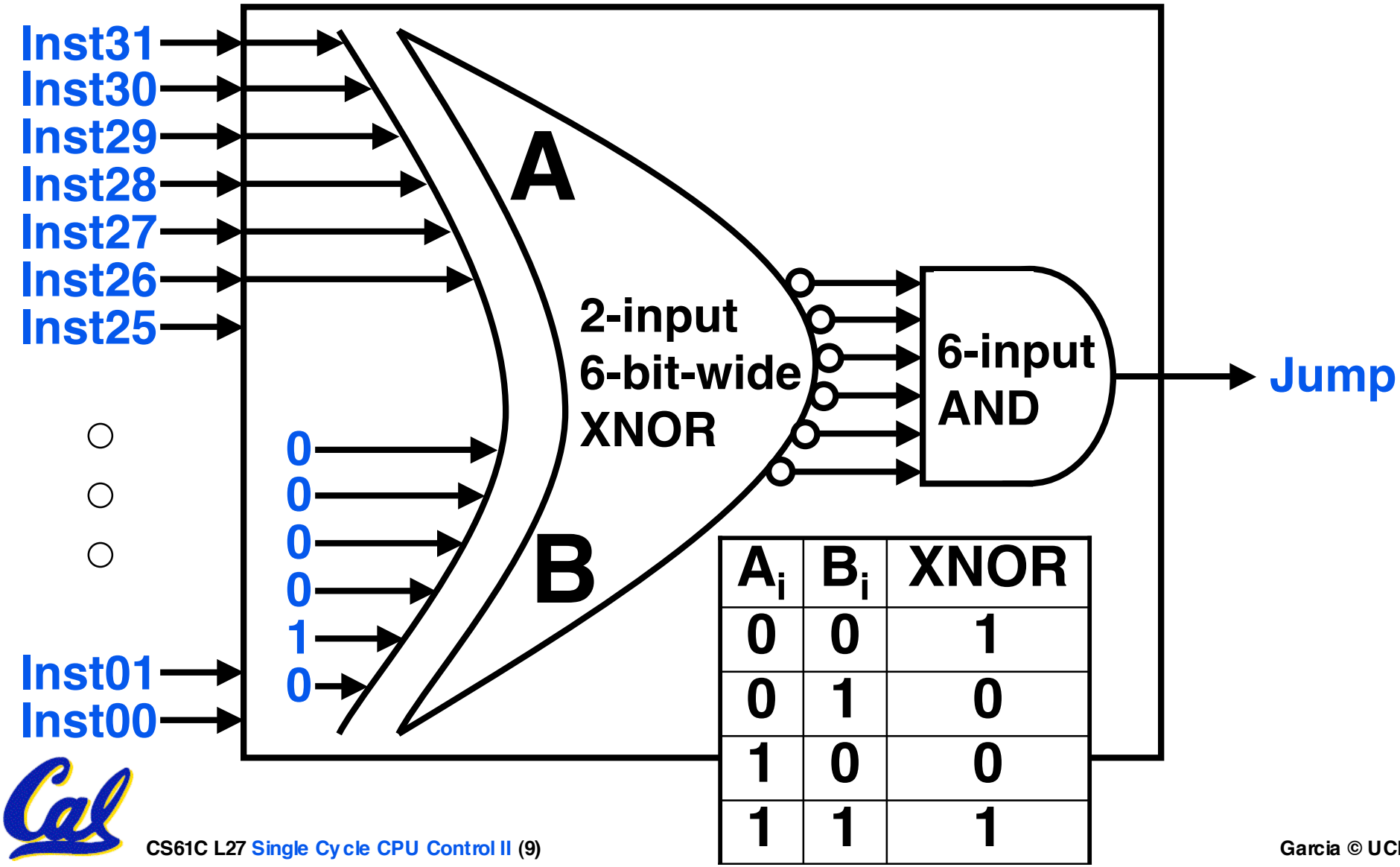
Query

- Can Zero still get asserted?
- Does nPC_sel need to be 0?
  - If not, what?

Build CL to implement Jump on paper now



Build CL to implement Jump on paper now



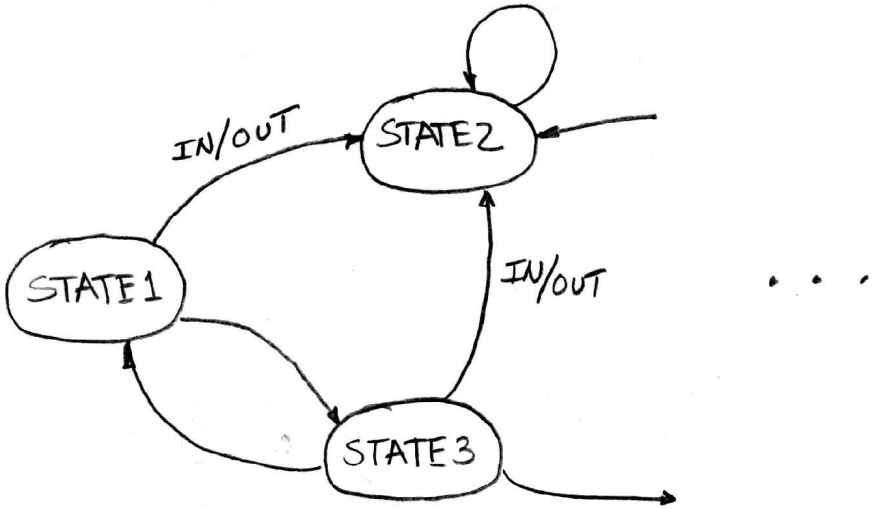
A_i	B_i	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

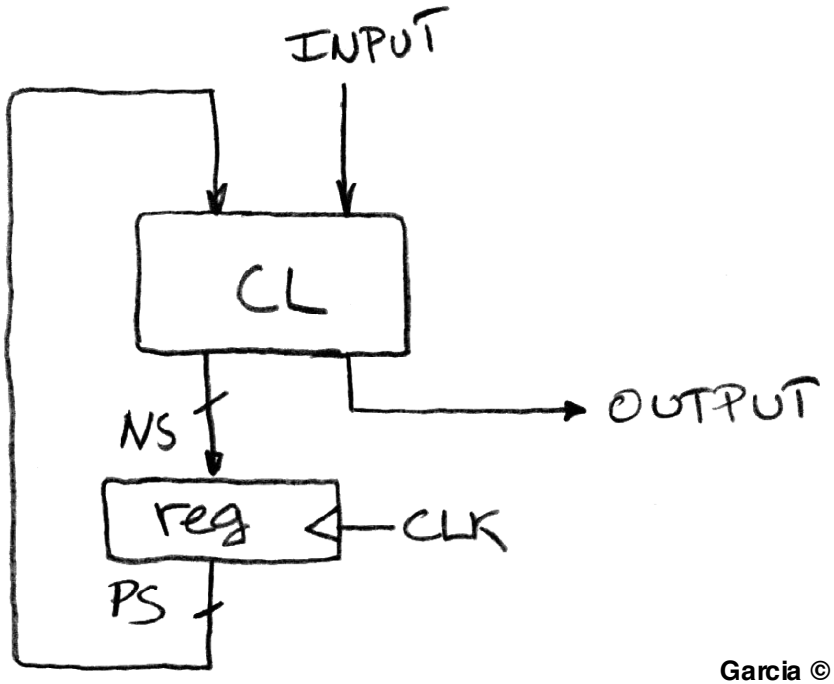
Administrivia

- HW7 out today, due in a week

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!

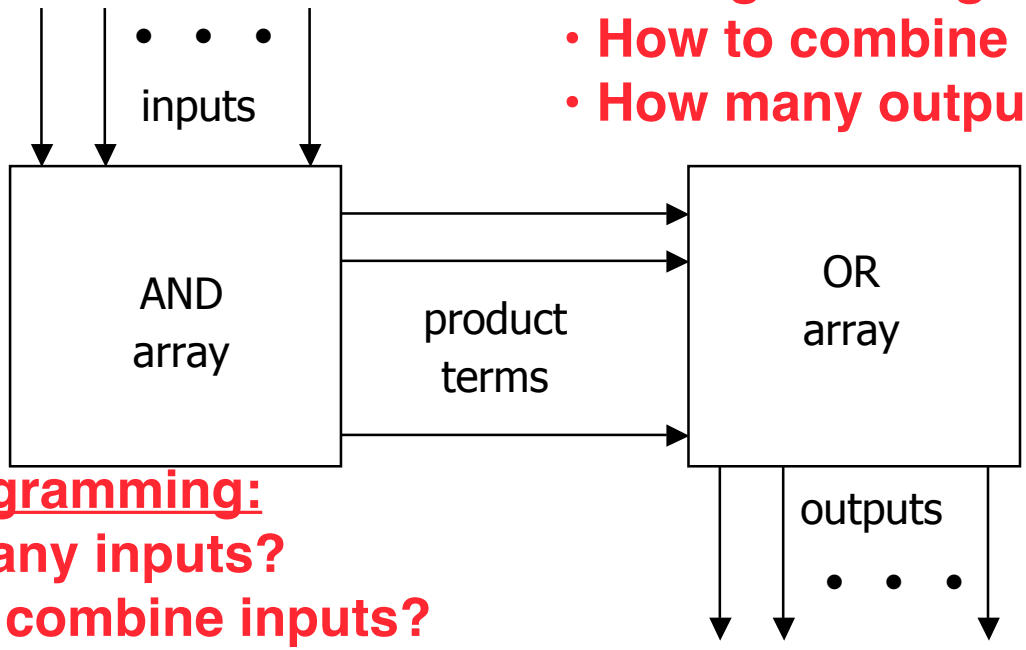
- They define
  - How output signals respond to input signals and previous state.
  - How we change states depending on input signals and previous state
- The output signals could be our familiar control signals
  - Some control signals may only depend on CL, not on state at all…
- We could implement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



Or Programming:
- **How to combine product terms?**
- **How many outputs?**

And Programming:
- **How many inputs?**
- **How to combine inputs?**
- **How many product terms?**

Enabling Concept

- **Shared product terms among outputs**

example:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

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

personality matrix

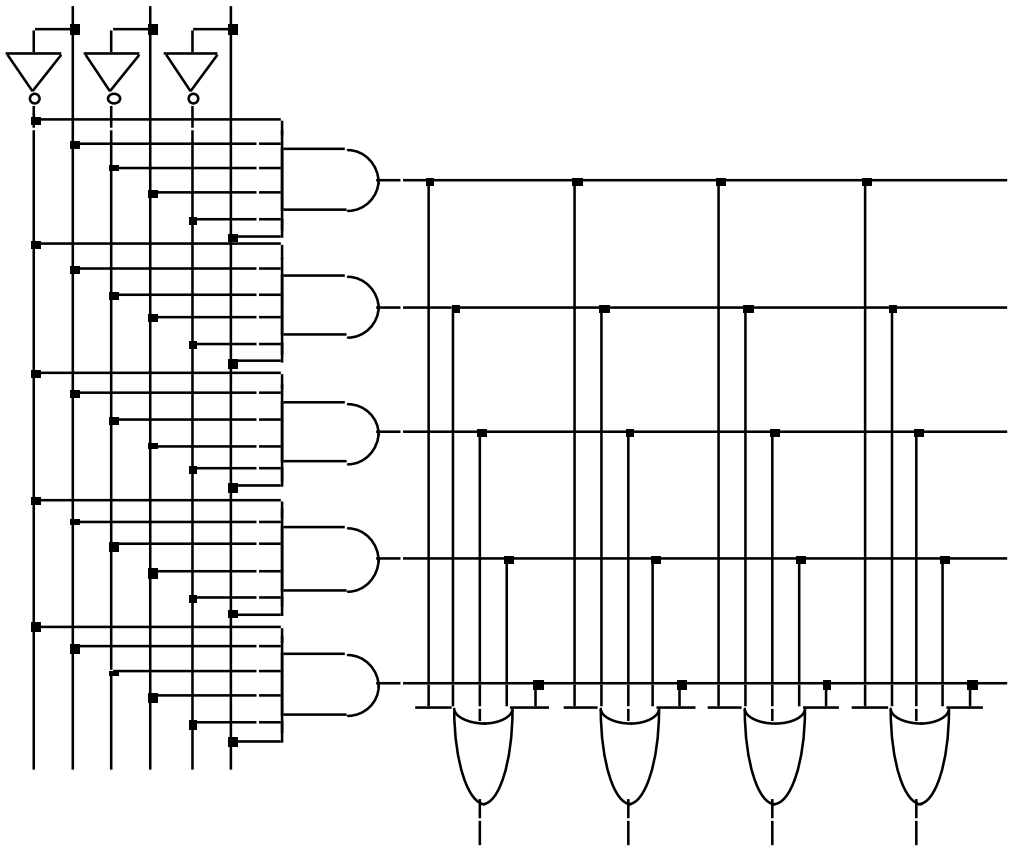
Product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

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

reuse of terms;
5 product terms

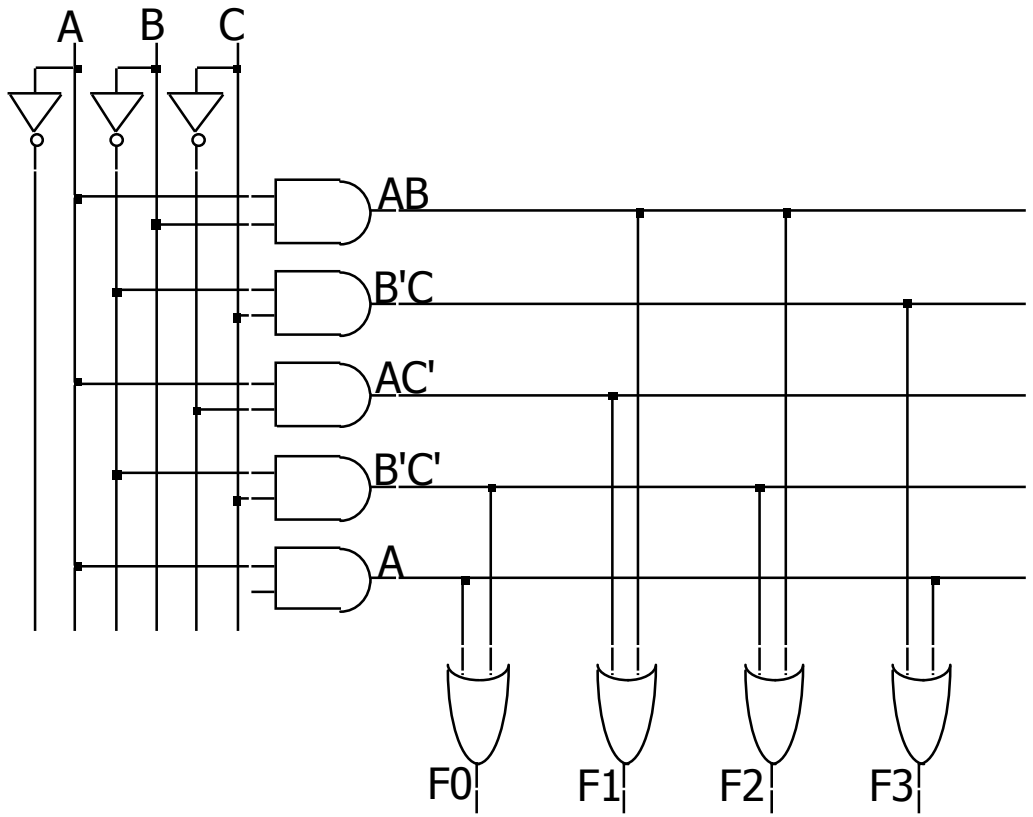
Before Programming

- All possible connections available before "programming"



After Programming

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



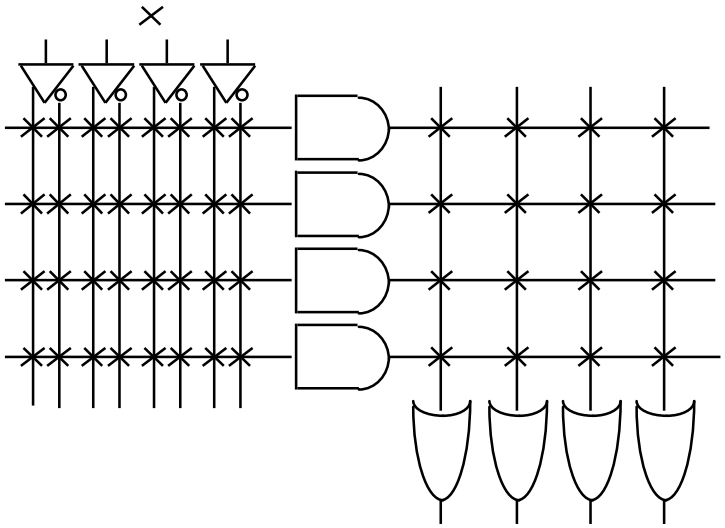
Alternate Representation

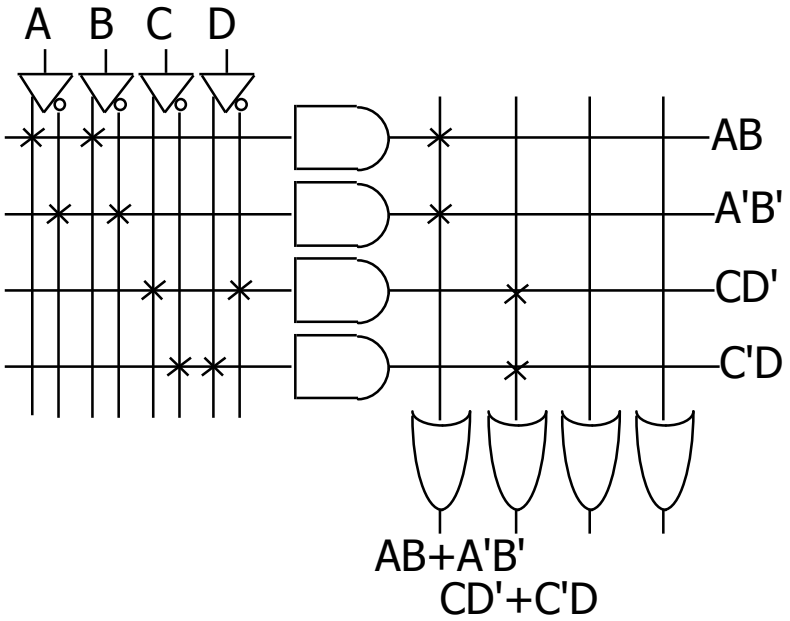
- **Short-hand notation--don't have to draw all the wires**
  - **X Signifies a connection is present and perpendicular signal is an input to gate**

notation for implementing

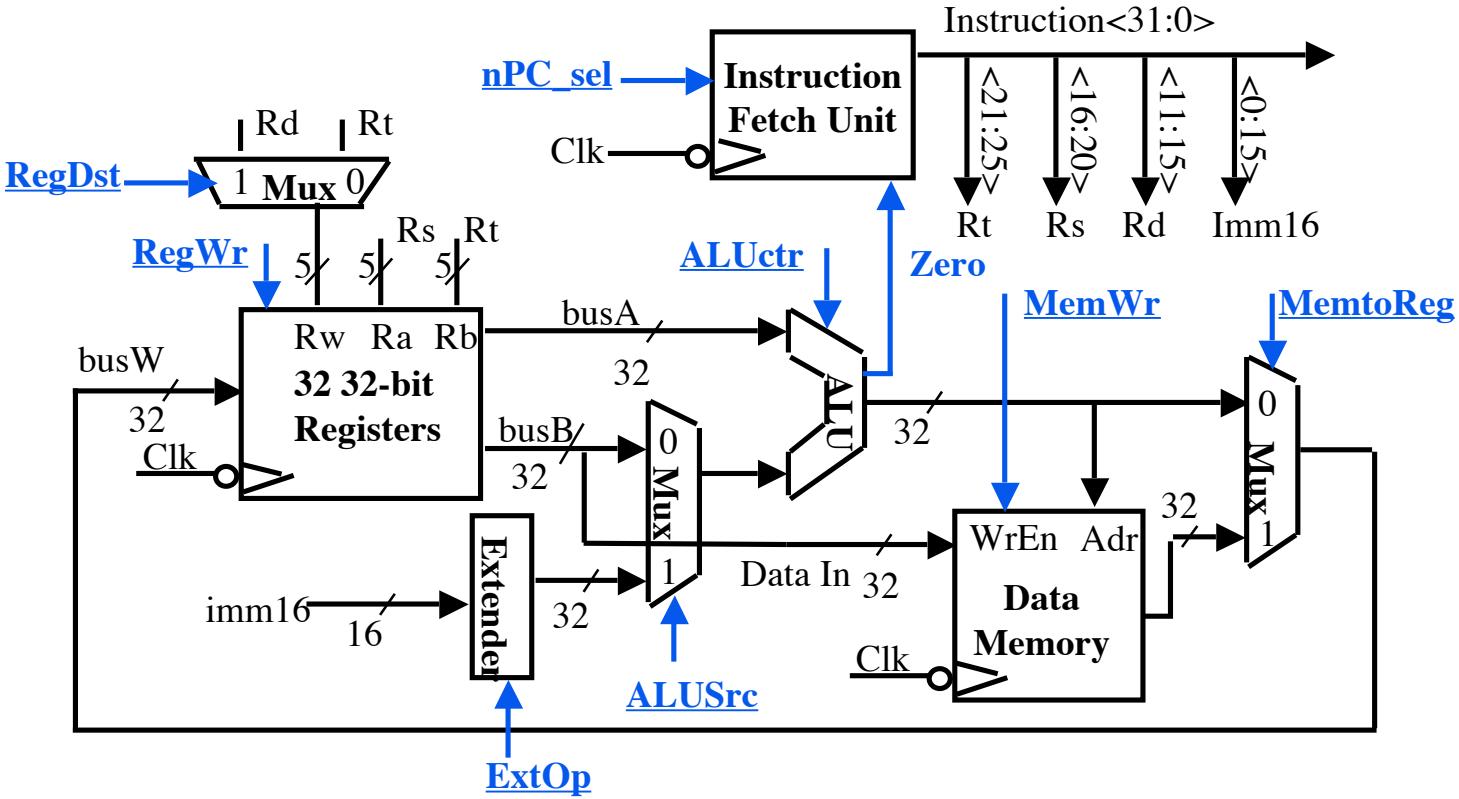
F0 = A B + A' B'

F1 = C D' + C' D





Peer Instruction



A. MemToReg='x' & ALUctr='sub'. SUB or BEQ?

B. ALUctr='add'. Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?

C. "Don't Care" signals are useful because we can simplify our PLA personality matrix. F / T?

	ABC
1:	SRF
2:	SRT
3:	SEF
4:	SET
5:	BRF
6:	BRT
7:	BEF
8:	BET

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
  - Operations always on registers/immediates