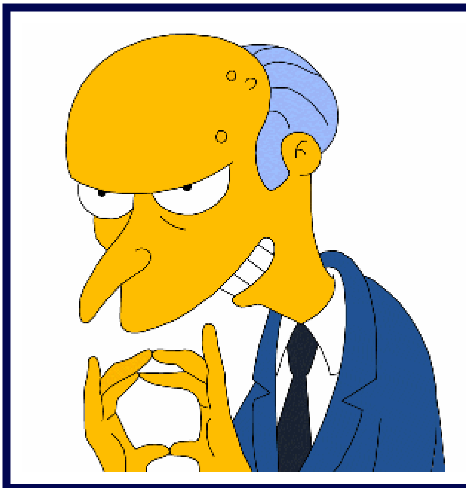
inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 29 – Single Cycle CPU Control II



2004-11-05

Andrew Schultz

inst.eecs.berkeley.edu/~cs61c-tb

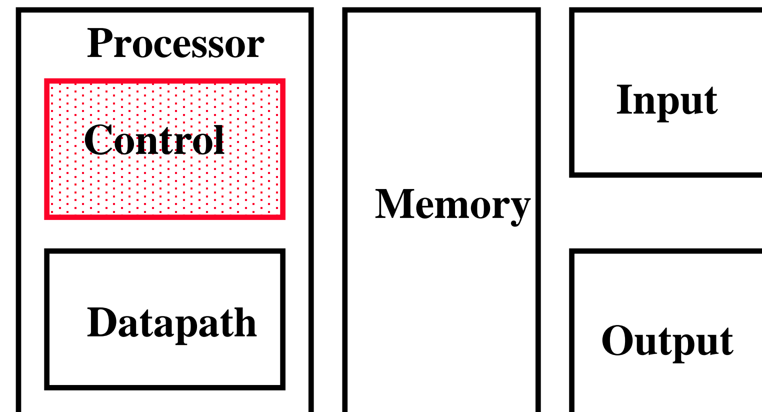
13TB of Memory ⇒ Soon after delivering a 10,240 processor supercomputer to NASA, SGI delivers a 2,048 node system to Japan with the worlds largest memory capacity, 13TB

http://www.sgi.com/company_info/newsroom/press_releases/2004/november/jaeri.html

Garcia, Fall 2004 © UCB

Review: Single cycle datapath

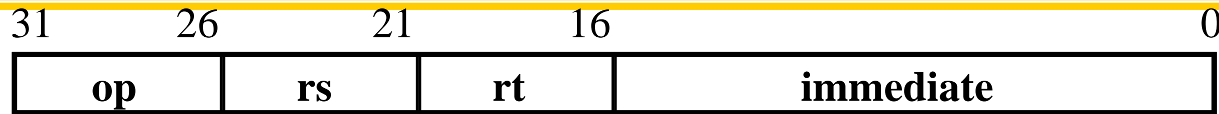
- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Assemble datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that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

Garcia, Fall 2004 © UCB

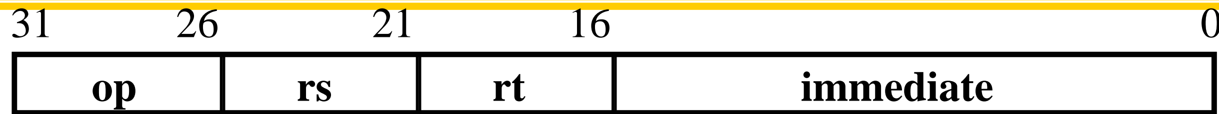
Single Cycle Datapath during Or Immediate?



- R[rt] = R[rs] OR ZeroExt[Imm16]

Instruction<31:0>
nPC_sel =
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst =
Rd
Rt
1 Mux 0
RegWr =
5
5
5
Rs
Rt
ALUctr =
Zero
MemWr =
MemtoReg =
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busA
32
busB
32
ALU
32
0
Mux
1
Extender
imm16
16
32
Data In
32
ALUSrc =
WrEn Adr
Data Memory
Clk
32
0
Mux
1
ExtOp =

Single Cycle Datapath during Or Immediate?

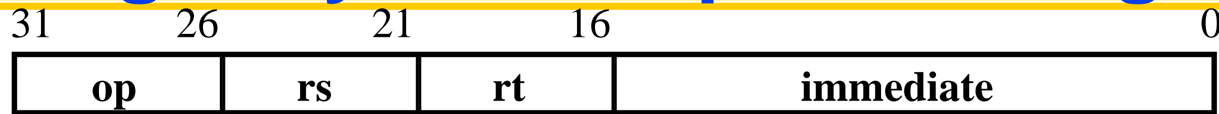


- R[rt] = R[rs] OR ZeroExt[Imm16]

Instruction<31:0>
nPC_sel= +4
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt Rs Rd Imm16
RegDst = 0
Rd Rt
1 Mux 0
RegWr = 1
5 5 5
Rs Rt
ALUctr = Or
MemtoReg = 0
Zero
MemWr = 0
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0 Mux 1
ALU
32
0 Mux 1
32
WrEn Adr
Data Memory
Data In 32
imm16
16
Extender
32
ALUSrc = 1
Clk
ExtOp = 0

Garcia, Fall 2004 © UCB

The Single Cycle Datapath during Load?

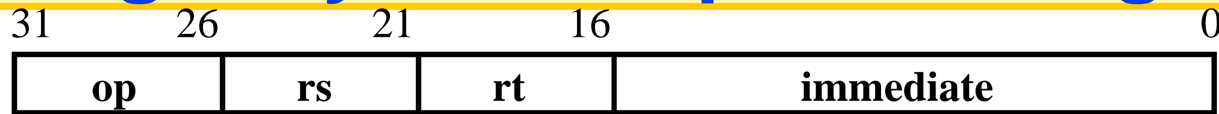


- R[rt] = Data Memory {R[rs] + SignExt[imm16]}

Instruction<31:0>
nPC_sel=
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst =
Rd
Rt
1 Mux 0
RegWr =
5
5
5
Rs
Rt
ALUctr =
Zero
MemWr =
MemtoReg =
busA
Rw Ra Rb
32 32-bit Registers
32
busW
32
Clk
busB
32
0 Mux 1
ALU
32
0 Mux 1
WrEn Adr
Data In
32
Data Memory
32
Clk
imm16
16
Extender
32
ALUSrc =
ExtOp =

Garcia, Fall 2004 © UCB

The Single Cycle Datapath during Load

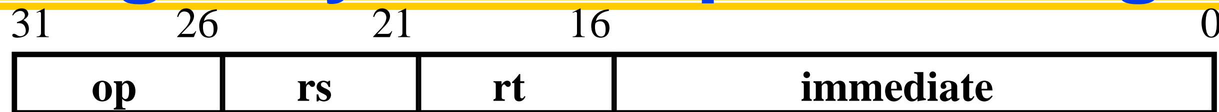


- R[rt] = Data Memory {R[rs] + SignExt[imm16]}

Instruction<31:0>
nPC_sel= +4
Instruction Fetch Unit
Clk
<21:25> <16:20> <11:15> <0:15>
Rt Rs Rd Imm16
RegDst = 0
Rd Rt
1 Mux 0
ALUctr = Add
MemtoReg = 1
RegWr = 1
5 5 5
Rs Rt
Zero
MemWr = 0
busA
busW
Rw Ra Rb
32 32-bit Registers
32
32
ALU
32
Clk
busB
32
0 Mux 1
0 Mux 1
WrEn Adr
Data In 32
Data Memory
imm16
16
Extender
32
32
Clk
ALUSrc = 1
ExtOp = 1

Garcia, Fall 2004 © UCB

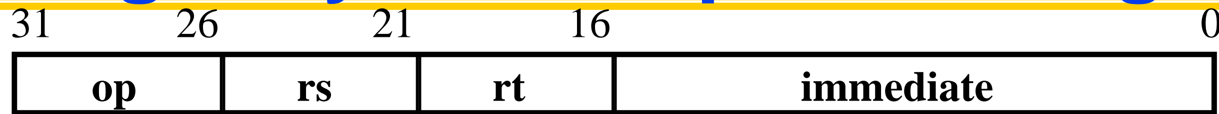
The Single Cycle Datapath during Store?



- Data Memory {R[rs] + SignExt[imm16]} = R[rt]

Instruction<31:0>
nPC_sel =
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst =
Rd
Rt
1 Mux 0
RegWr =
5
5
Rs
5
Rt
ALUctr =
Zero
MemWr =
MemtoReg =
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busA
32
busB
32
ALU
32
0
Mux
1
Extender
imm16
16
32
Data In
32
WrEn Adr
Data Memory
Clk
32
0
Mux
1
ALUSrc =
ExtOp =

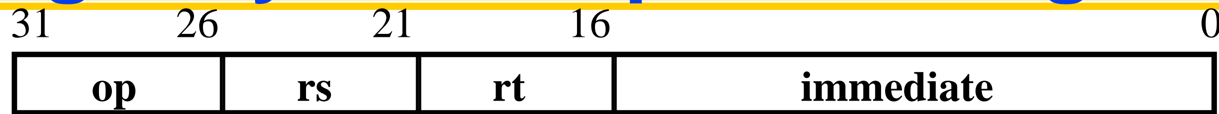
The Single Cycle Datapath during Store



- Data Memory {R[rs] + SignExt[imm16]} = R[rt]

Instruction<31:0>
nPC_sel= +4
Instruction Fetch Unit
Clk
<21:25> <16:20> <11:15> <0:15>
Rt Rs Rd Imm16
RegDst = x
Rd Rt
1 Mux 0
RegWr = 0
5 5 5
Rs Rt
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busA
32
busB
32
ALUctr = Add
ALU
Zero
MemWr = 1
MemtoReg = x
32
0 Mux 1
32
Extender
imm16
16
32
Data In 32
WrEn Adr
Data Memory
Clk
ALUSrc = 1
ExtOp = 1
0
Mux
1

The Single Cycle Datapath during Branch?

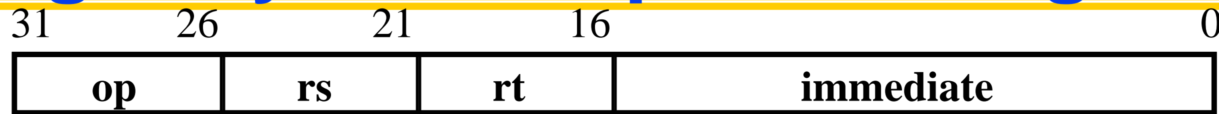


- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0

nPC_sel=
Instruction<31:0>
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt Rs Rd Imm16
RegDst =
Rd Rt
1 Mux 0
RegWr =
Rs Rt
5 5 5
ALUctr =
MemtoReg =
Zero
MemWr =
busA
32
ALU
32
busW
Rw Ra Rb
32 32-bit Registers
32
Clk
busB
32
0 Mux 1
0 Mux 1
32
WrEn Adr
Data In 32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc =
ExtOp =

Garcia, Fall 2004 © UCB

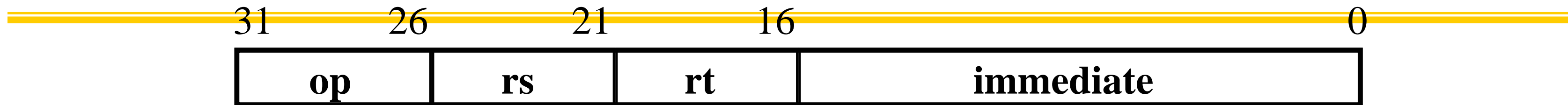
The Single Cycle Datapath during Branch



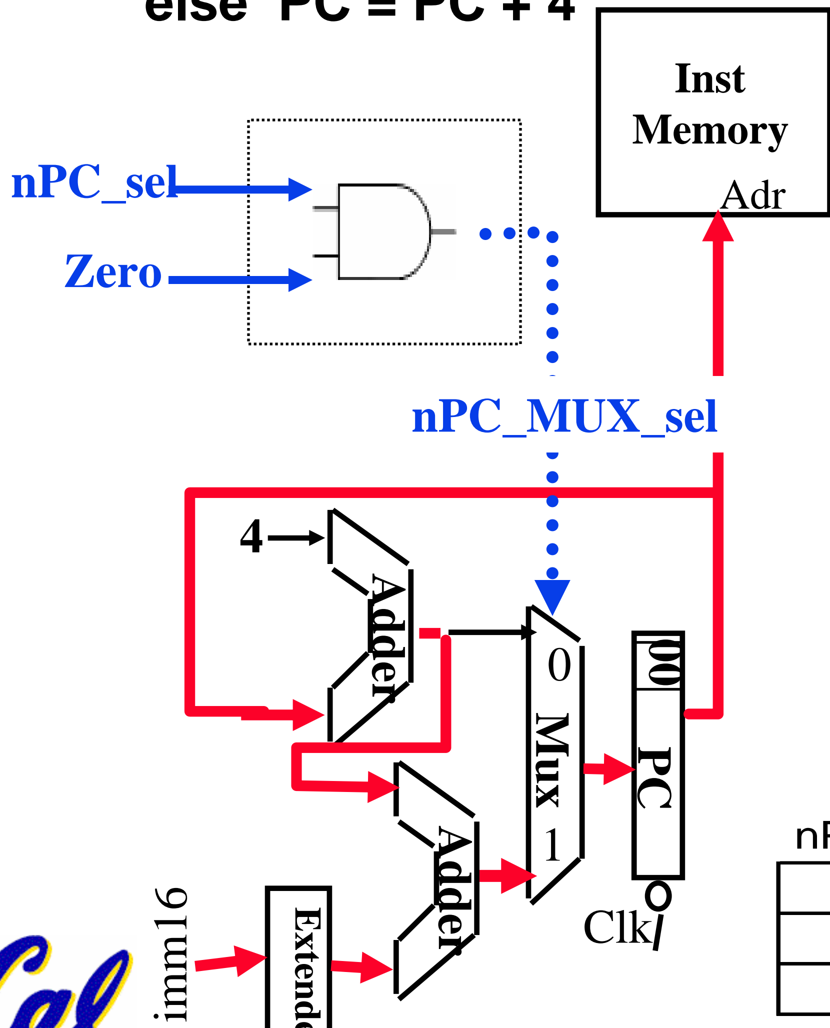
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0

nPC_sel= "Br"
Instruction Fetch Unit
Clk
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = x
Rd
Rt
1 Mux 0
ALUctr =Sub
RegWr = 0
5
5
5
Rs
Rt
MemtoReg = x
Zero
MemWr = 0
busA
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
32
ALU
32
0
Mux
1
busB
32
0
Mux
1
Extender
imm16
16
32
Data In
32
WrEn Adr
Data Memory
Clk
32
ALUSrc = 0
ExtOp = x

Instruction Fetch Unit at the End of Branch



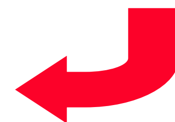
- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4



- What is encoding of nPC_MUX_sel?
  - Direct MUX select?
  - Branch / not branch
- Let's pick 2nd option

nPC_sel	zero?	MUX
0	x	0
1	0	0
1	1	1

Q: What logic gate?



Step 4: Given Datapath: RTL -> Control

Instruction<31:0>

Inst Memory

Adr

Field	Bits
Op	<26:31>
Fun	<0:5>
Rt	<21:25>
Rs	<16:20>
Rd	<11:15>
Imm16	<0:15>

Control

nPC_sel RegWr RegDst ExtOp ALUSrc ALUctr MemWr MemtoReg **Zero**

DATA PATH

Garcia, Fall 2004 © UCB

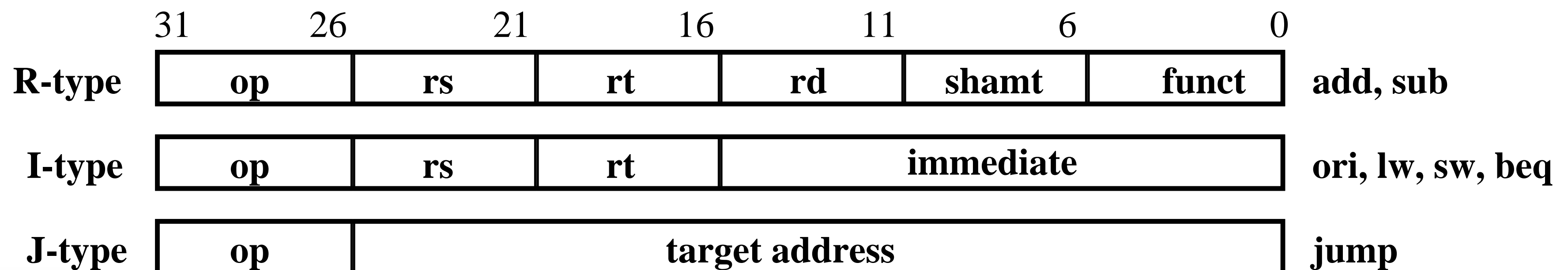
A Summary of the Control Signals (1/2)

inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”	
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “sub”, RegDst = rd, RegWr, nPC_sel = “+4”	
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
	ALUsrc = Im, Extop = “Z”, ALUctr = “or”, RegDst = rt, RegWr, nPC_sel =“+4”	
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemtoReg, RegDst = rt, RegWr,	nPC_sel = “+4”
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemWr, nPC_sel = “+4”	
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	
	nPC_sel = “Br”, ALUctr = “sub”	

A Summary of the Control Signals (2/2)

See Appendix A

	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
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	xxx



Garcia, Fall 2004 © UCB

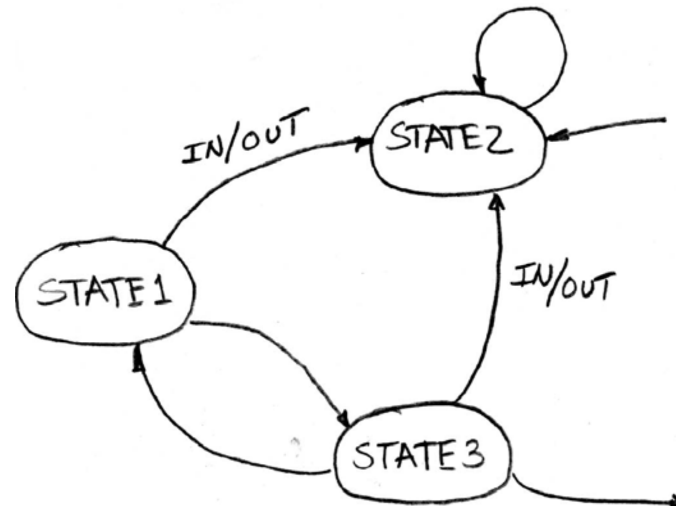
Administrivia

- Final exam time/location set
  - Tuesday, December 14th, 12:30 – 3:30 pm
  - At the Hearst Gym (lucky us!)

Garcia, Fall 2004 © UCB

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.



Garcia, Fall 2004 © UCB

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

Garcia, Fall 2004 © UCB

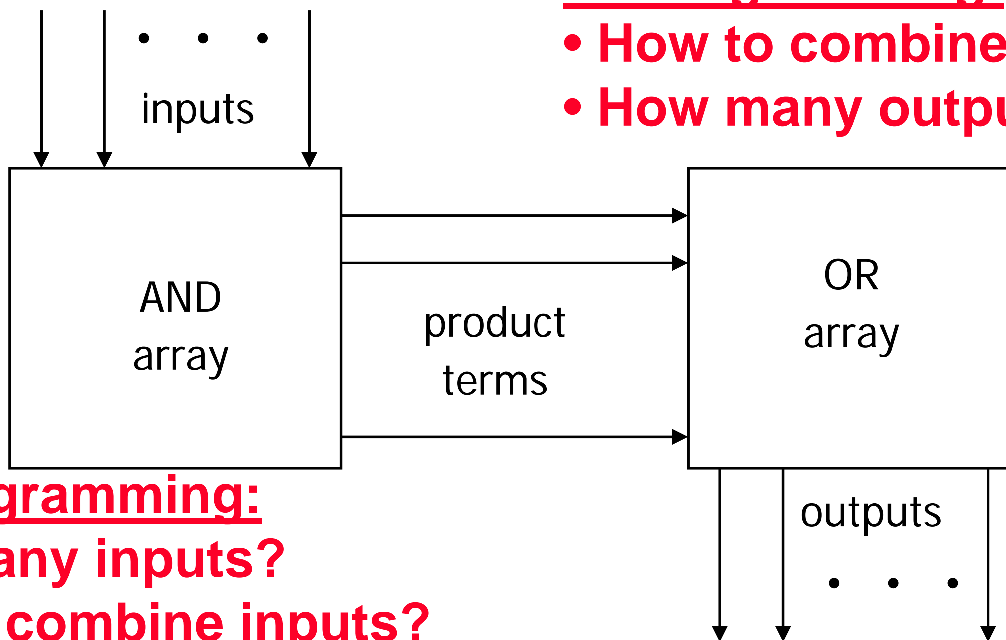
Taking advantage of sum-of-products

- 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, …

Garcia, Fall 2004 © UCB

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?

Garcia, Fall 2004 © UCB

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

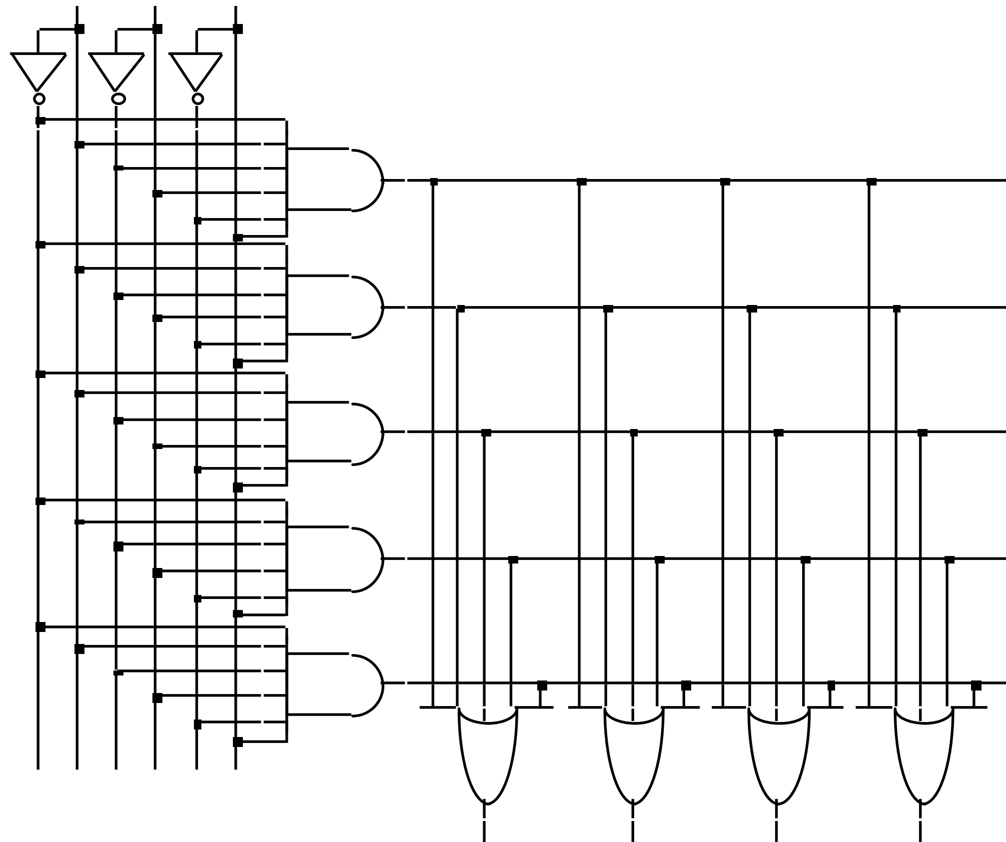
1 = term connected to output
0 = no connection to output

reuse of terms;
5 product terms

Garcia, Fall 2004 © UCB

Before Programming

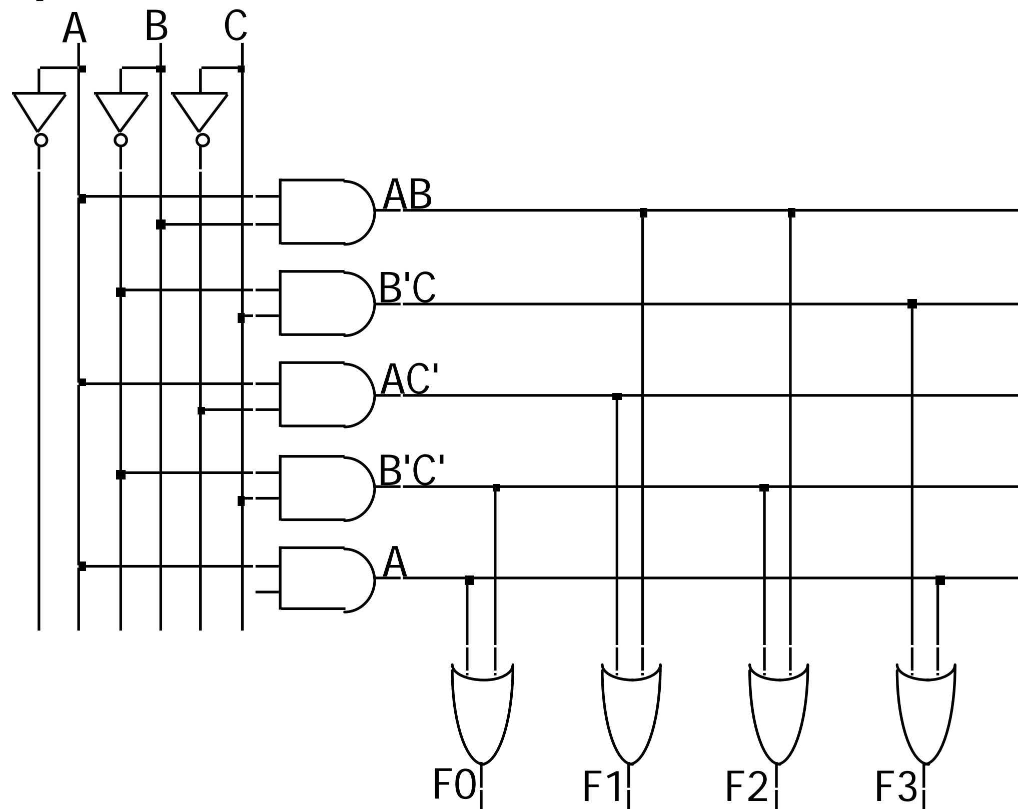
- All possible connections available before "programming"



Garcia, Fall 2004 © UCB

After Programming

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



Garcia, Fall 2004 © UCB

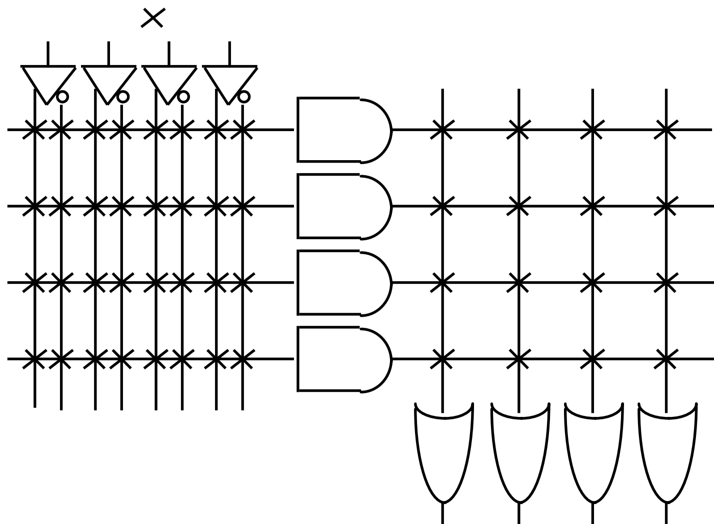
Alternate Representation

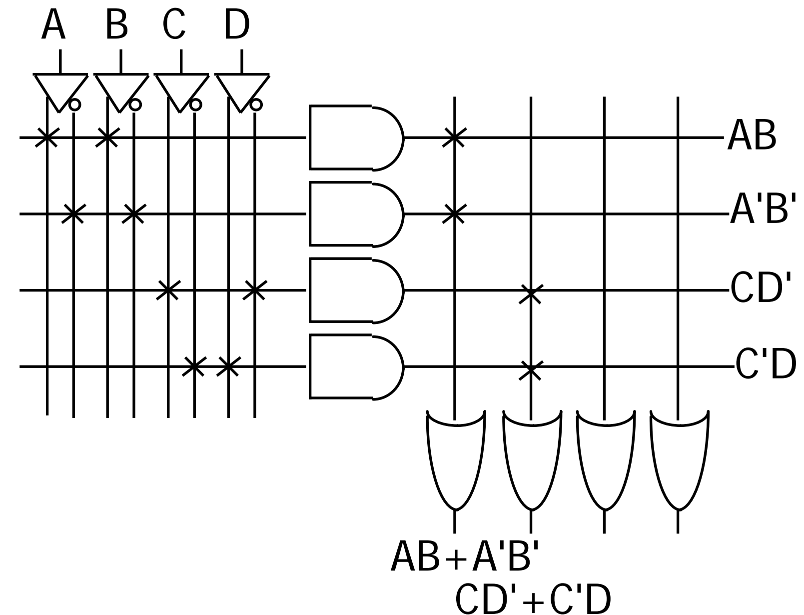
- Short-hand notation--don't have to draw all the wires
  - X Signifies a connection is present and perpendicular signal is an input to gate

notation for implementing

F0 = A B + A' B'

F1 = C D' + C' D





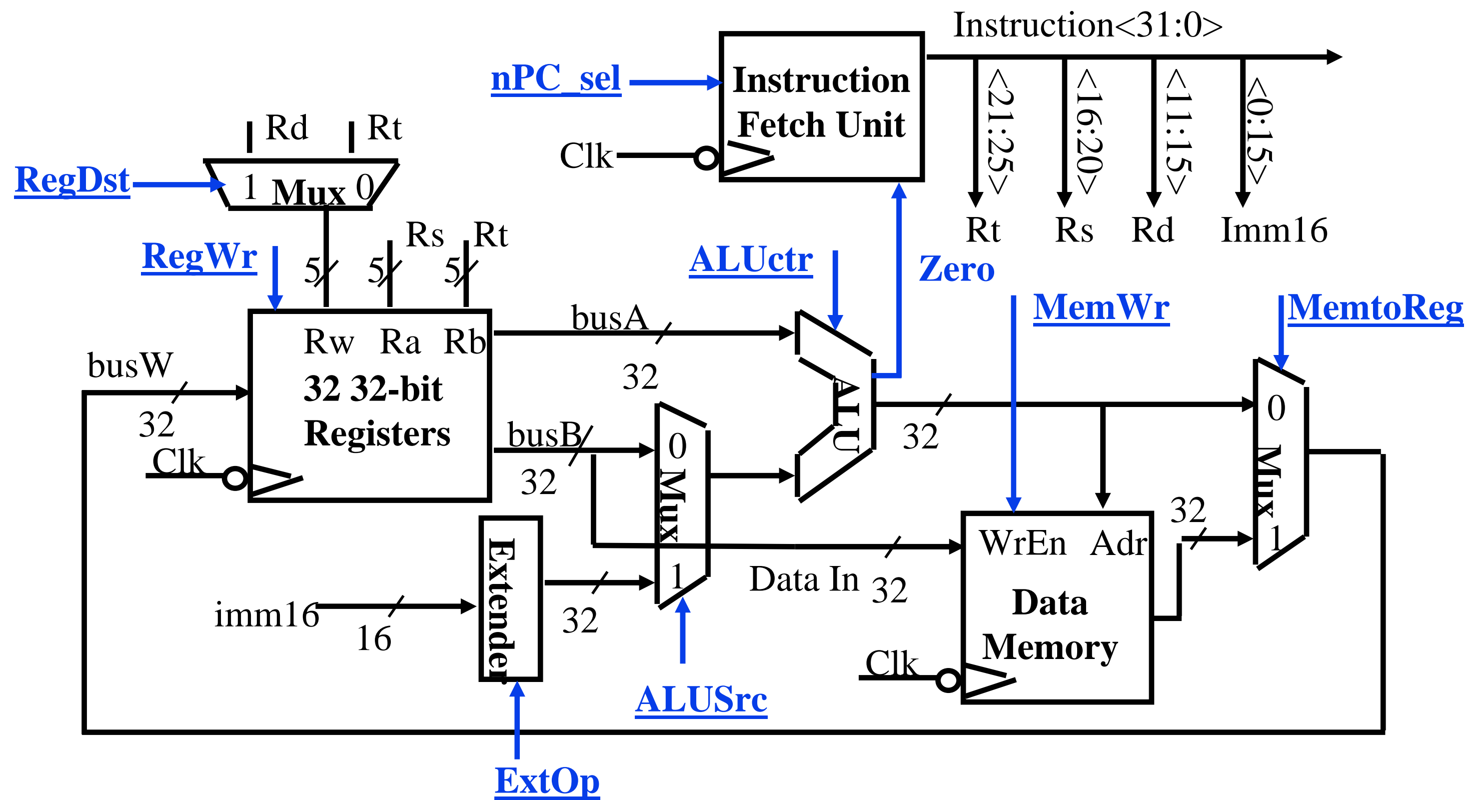
Garcia, Fall 2004 © UCB

Other Programmable Logic Arrays

- There are other types of PLAs which can be reprogrammed on the fly
- The most common is called a Field Programmable Gate Array (FPGA)
- FPGAs are made up of configurable logic blocks (CLBs) and flip-flops which can be programmed by software
- Berkeley has on-going research into reconfigurable computing with FPGAs
    - Check out Brass and BEE2 projects

Garcia, Fall 2004 © UCB

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?

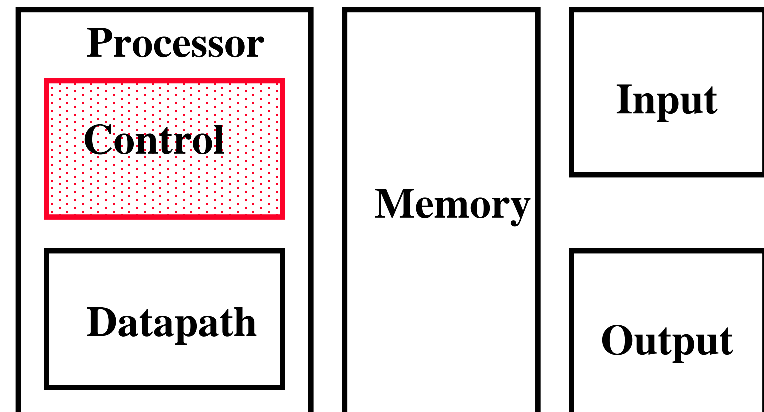
C. “Don’t Care” signals are useful because we can simplify our PLA personality matrix. F / T?

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

Garcia, Fall 2004 © 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
  - Operations always on registers/immediates