inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 29 – Single Cycle CPU Control II

2004-11-05

Andrew Schultz

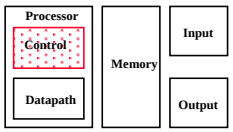
inst.eecs.berkeley.edu/~cs61c-tb

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

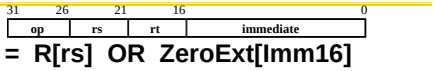
http://www.sgi.com/company_info/newsroom/press_releases/2004/november/jaeri.html

Review: Single cycle datapath

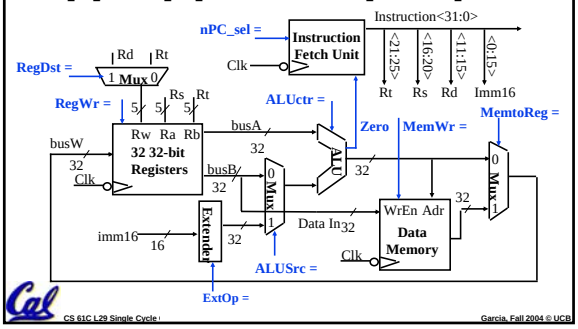
- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Assemble datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that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



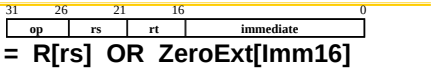
Single Cycle Datapath during Or Immediate?



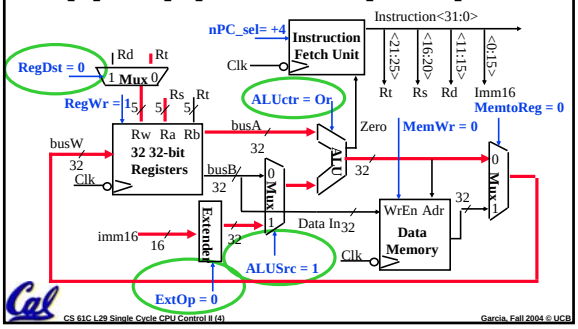
- R[rt] = R[rs] OR ZeroExt[Imm16]



Single Cycle Datapath during Or Immediate?

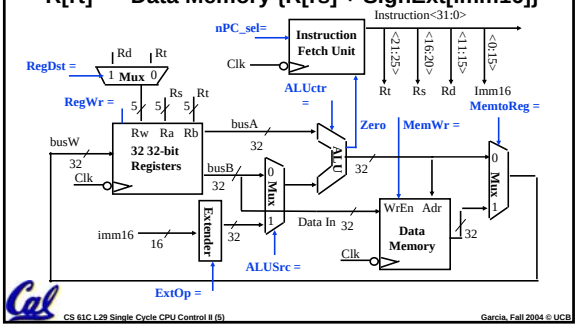


- R[rt] = R[rs] OR ZeroExt[Imm16]



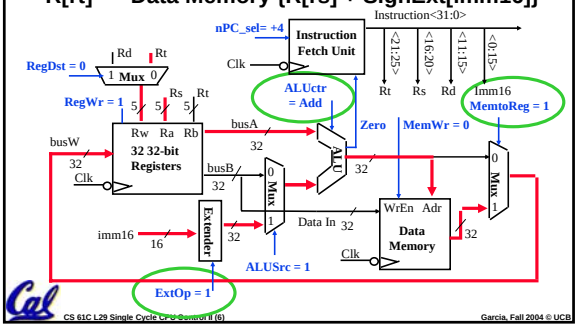
The Single Cycle Datapath during Load?

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

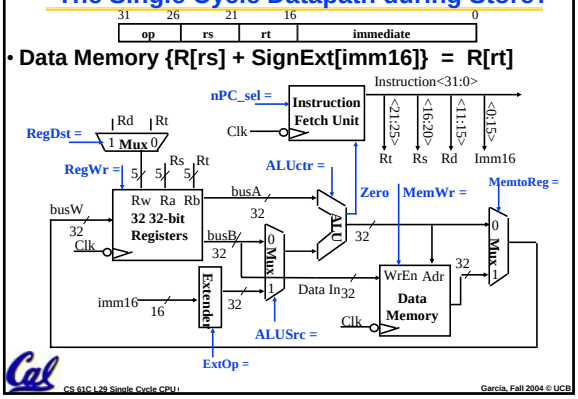


The Single Cycle Datapath during Load

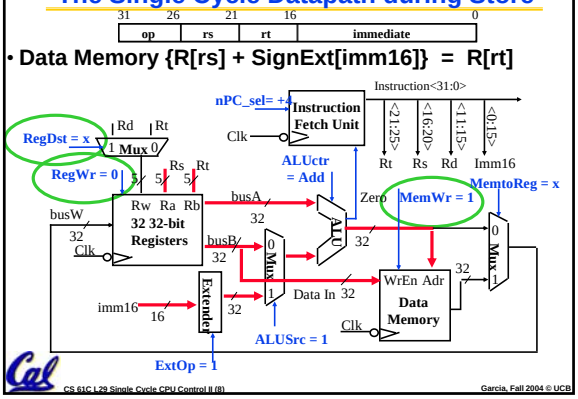
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



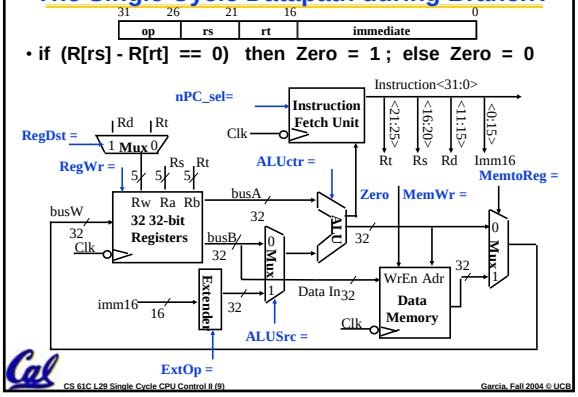
The Single Cycle Datapath during Store?



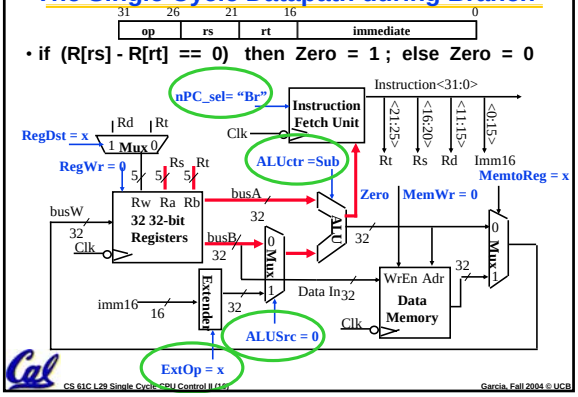
The Single Cycle Datapath during Store



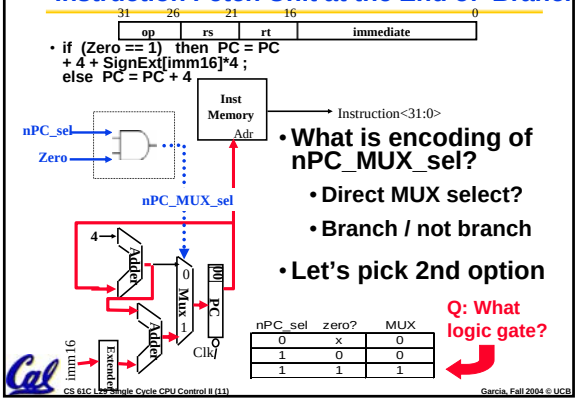
The Single Cycle Datapath during Branch?



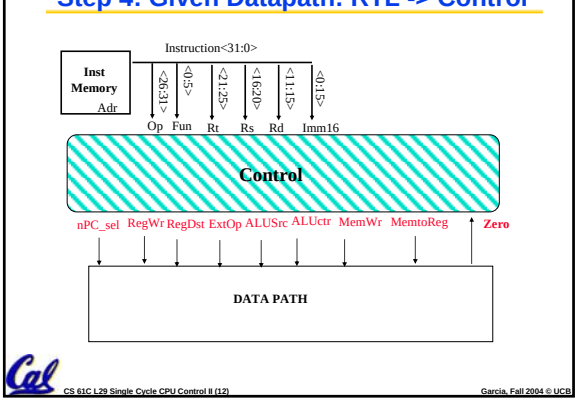
The Single Cycle Datapath during Branch



Instruction Fetch Unit at the End of Branch



Step 4: Given Datapath: RTL -> Control

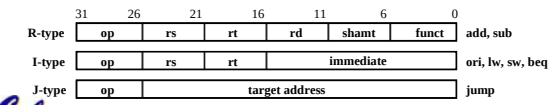


A Summary of the Control Signals (1/2)

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

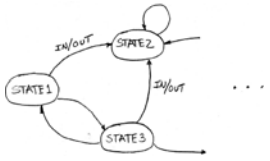


Administrivia

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

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!

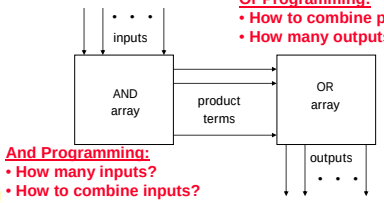
- **They define**
  - **How output signals respond to input signals and previous state.**
  - **How we change states depending on input signals and previous state**
- **The output signals could be our familiar control signals**
  - **Some control signals may only depend on CL, not on state at all…**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

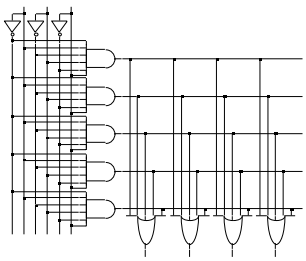
Product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

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

reuse of terms; 5 product terms

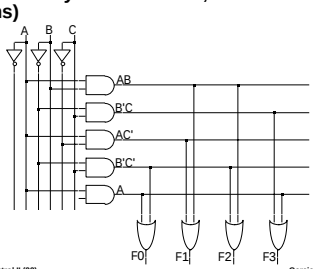
Before Programming

- All possible connections available before "programming"



After Programming

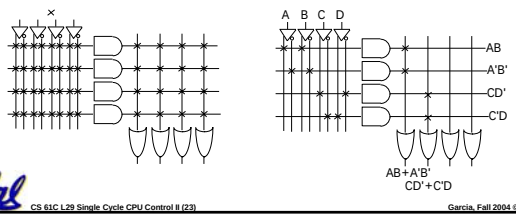
- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)



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

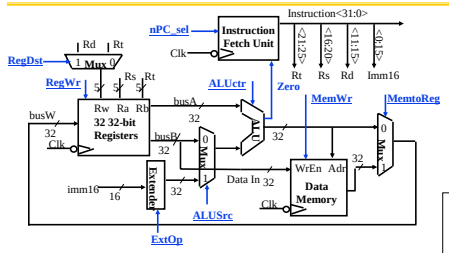
notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D



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

Peer Instruction



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

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?

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