inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #18
Single Cycle CPU Control

CPS today!

2005-11-02

There is one handout today at the front and back of the room!

Lecturer PSOE, new dad Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Data Transfer Record! ⇒ A Japanese company (Kansai Electric) claims fiber-optic cables on power-transmitting steel towers have achieved 1 Terabit/sec, 100 times faster than normal.

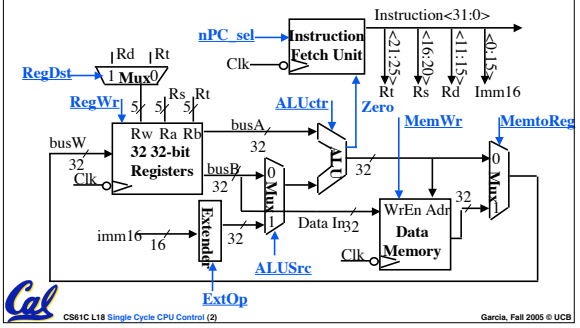
news.softpedia.com/news/Japanese-Claim-New-Fiber-Optic-Transmission-Record-11257.shtml

5:00:00 send 2 hr movie

5:00:01 get 2 hr movie (< 1 second!)

Summary: A Single Cycle Datapath

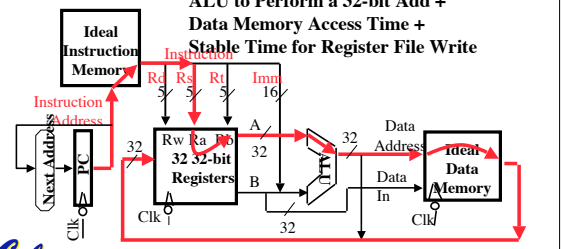
- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals



An Abstract View of the Critical Path

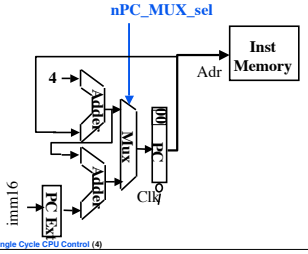
- This affects how much you can overclock your PC!

Critical Path (Load Operation) = Delay clock through PC (FFs) + Instruction Memory's Access Time + Register File's Access Time, + ALU to Perform a 32-bit Add + Data Memory Access Time + Stable Time for Register File Write



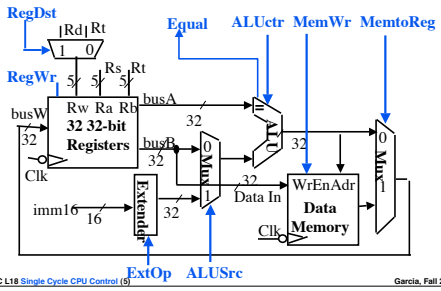
Recap: Meaning of the Control Signals

- nPC_MUX_sel: "n"=next
  - 0 ⇒ PC <– PC + 4
  - 1 ⇒ PC <– PC + 4 + {SignExt(Im16) , 00 }
- Later in lecture: higher-level connection between mux and branch cond

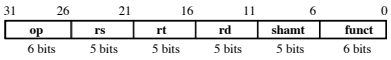


Recap: Meaning of the Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "add", "sub", "or"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



RTL: The Add Instruction

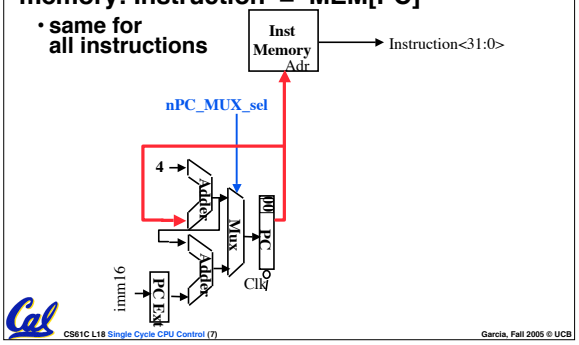


`add rd, rs, rt`

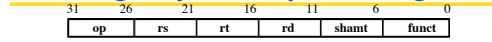
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

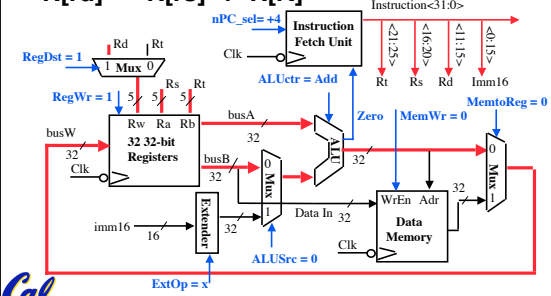
- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions



The Single Cycle Datapath during Add

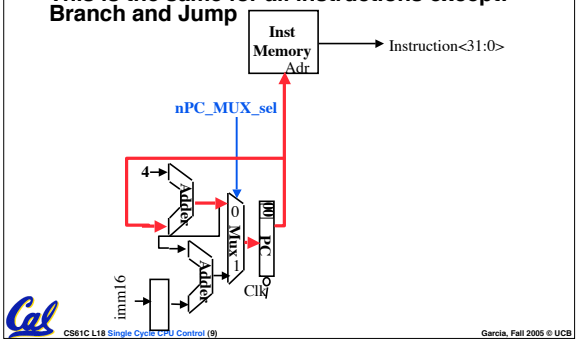


- R[rd] = R[rs] + R[rt]



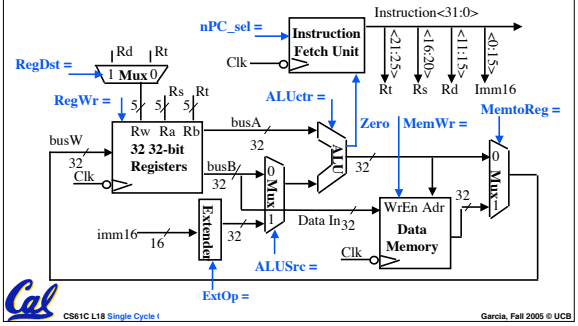
Instruction Fetch Unit at the End of Add

- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump

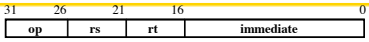


Single Cycle Datapath during Or Immediate?

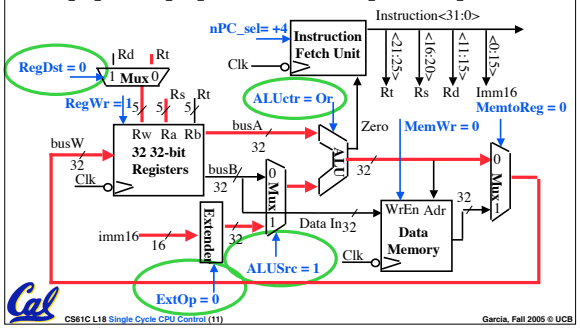
- R[rt] = R[rs] OR ZeroExt[Imm16]



Single Cycle Datapath during Or Immediate?

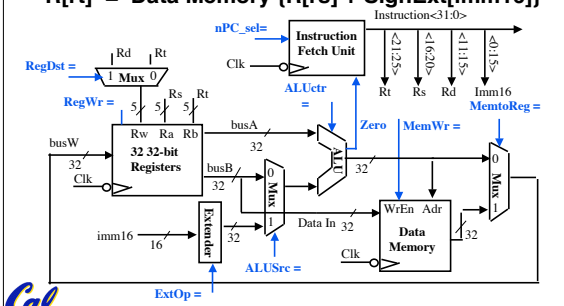


- R[rt] = R[rs] OR ZeroExt[Imm16]

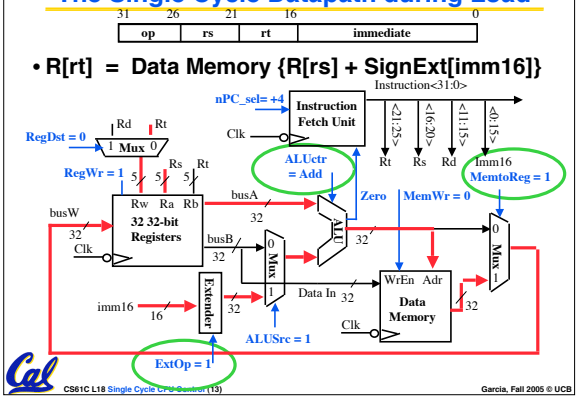


The Single Cycle Datapath during Load?

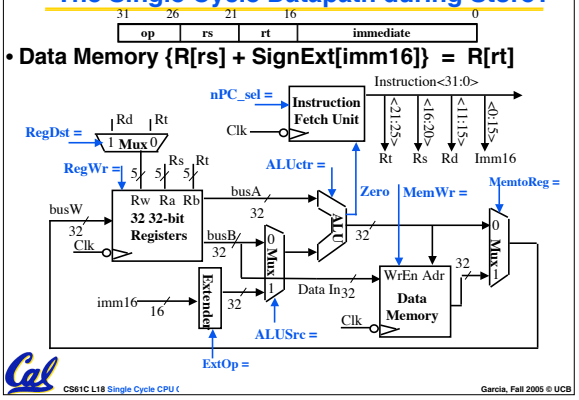
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



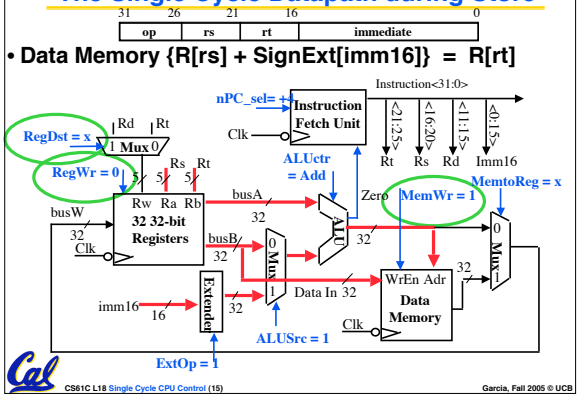
The Single Cycle Datapath during Load



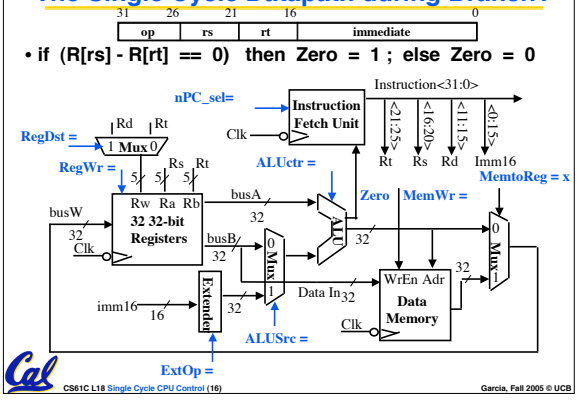
The Single Cycle Datapath during Store?



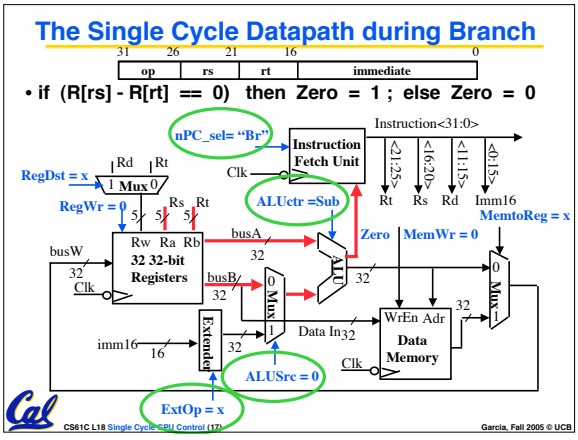
The Single Cycle Datapath during Store



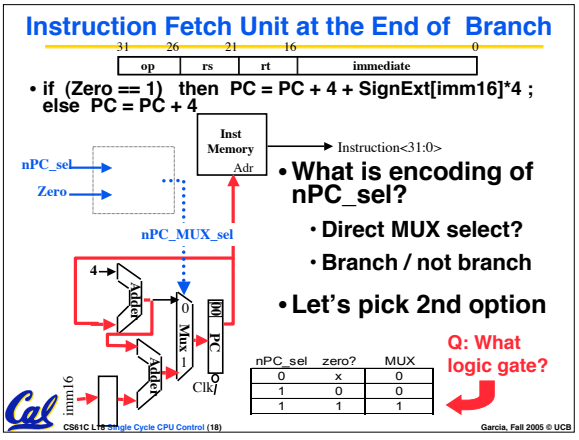
The Single Cycle Datapath during Branch?



The Single Cycle Datapath during Branch



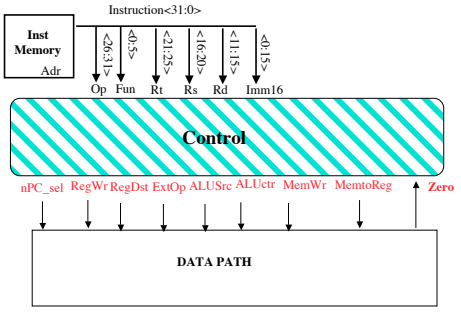
Instruction Fetch Unit at the End of Branch



Administrivia

- Any Administrivia?

Step 4: Given Datapath: RTL ⇒ Control

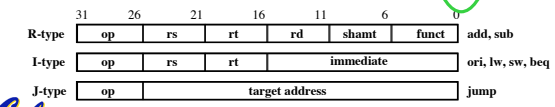


A Summary of the Control Signals (1/2)

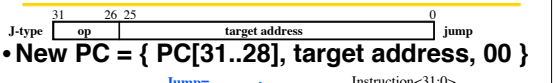
inst	Register Transfer	
ADD	R[rd] <- R[rs] + R[rt];	PC <- PC + 4
	ALUsrc = RegB, ALUctr = "add", RegDst = rd, RegWr, nPC_sel = "+4"	
SUB	R[rd] <- R[rs] – R[rt];	PC <- PC + 4
	ALUsrc = RegB, ALUctr = "sub", RegDst = rd, RegWr, nPC_sel = "+4"	
ORi	R[rt] <- R[rs] + zero_ext(Imm16);	PC <- PC + 4
	ALUsrc = 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)

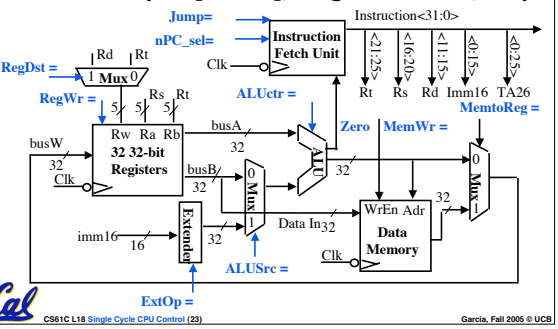
See Appendix A	func	10 0000	10 0010	We Don't Care :-)				
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		add	sub	ori	lw	sw	beq	jump
RegDst		1	1	0	0	x	x	x
ALUSrc		0	0	1	1	1	0	x
MemtoReg		0	0	0	1	x	x	x
RegWrite		1	1	1	1	0	0	0
MemWrite		0	0	0	0	1	0	0
nPCsel		0	0	0	0	0	1	0
Jump		0	0	0	0	0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



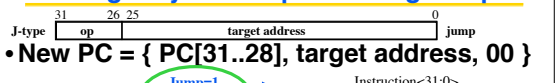
The Single Cycle Datapath during Jump



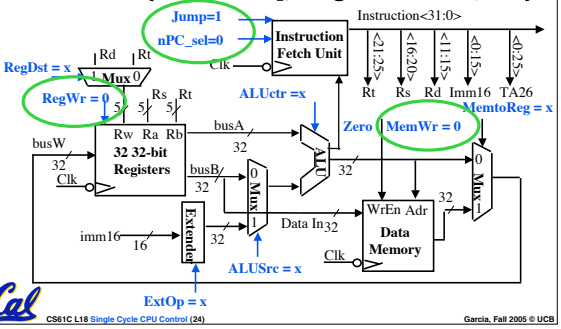
- New PC = { PC[31..28], target address, 00 }



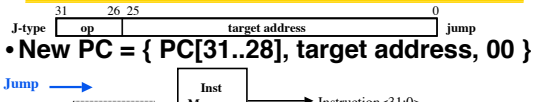
The Single Cycle Datapath during Jump



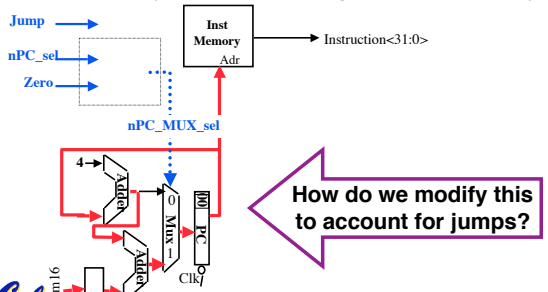
- New PC = { PC[31..28], target address, 00 }



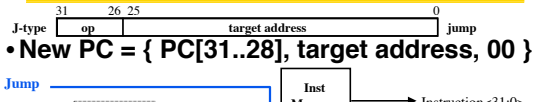
Instruction Fetch Unit at the End of Jump



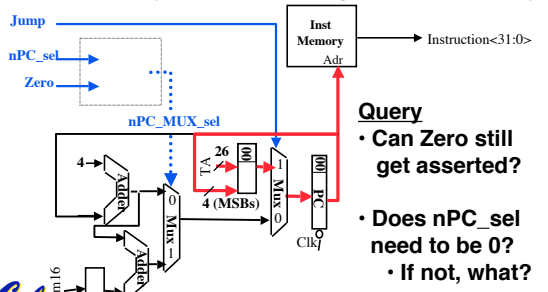
- New PC = { PC[31..28], target address, 00 }



Instruction Fetch Unit at the End of Jump



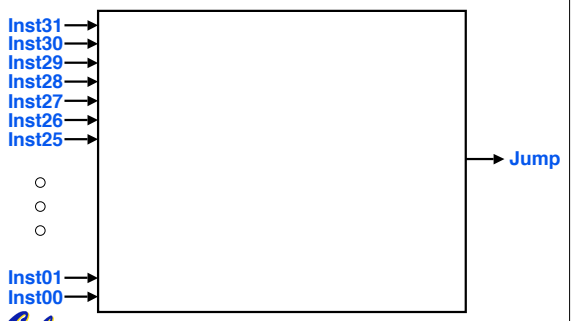
- New PC = { PC[31..28], target address, 00 }



Query

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

Build CL to implement Jump on paper now



Peer Instruction

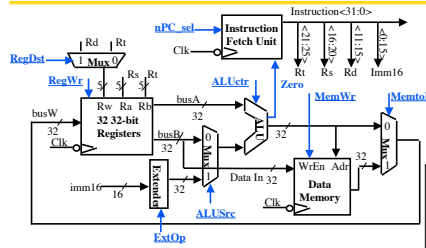
A. Our ALU is a synchronous device

B. We should use the main ALU to compute PC=PC+4

C. The ALU is inactive for memory reads or writes.

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TTF
8:	TTT

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