inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 27 – Single Cycle CPU Control I

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

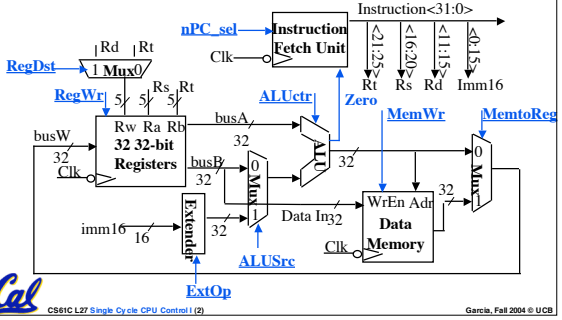
Brain-computer interfaces! ⇒

Paralyzed people can now control artificial limbs! There are two brain connection techniques, implants requiring significant surgery (& brain inflammation) and the non-invasive "swimming cap". You adapt to either.

www.guardian.co.uk/life/feature/story/0,13026,1448140,00.html

Summary: A Single Cycle Datapath

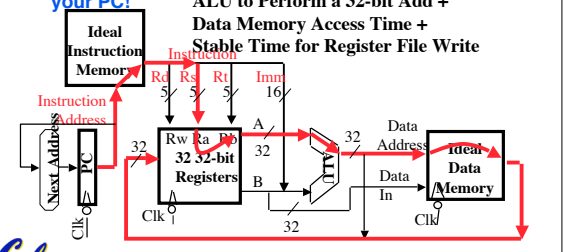
- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals



An Abstract View of the Critical Path

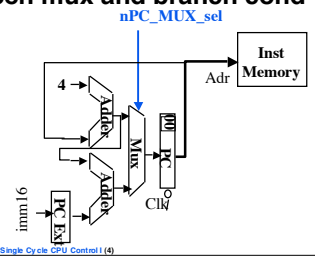
- This affects how much you can overclock your PC!

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



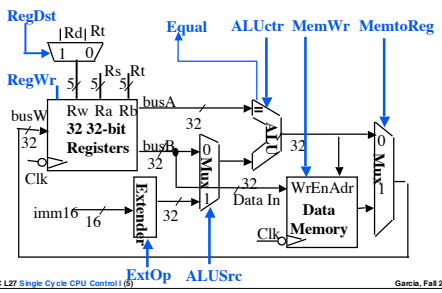
Recap: Meaning of the Control Signals

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



Recap: Meaning of the Control Signals

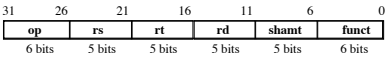
- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "add", "sub", "or"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



Administrivia

- Steven & Andy moved today's OH to 11-noon before lecture

RTL: The Add Instruction

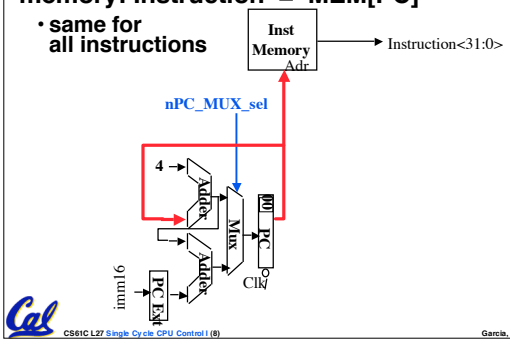


`add rd, rs, rt`

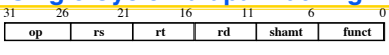
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

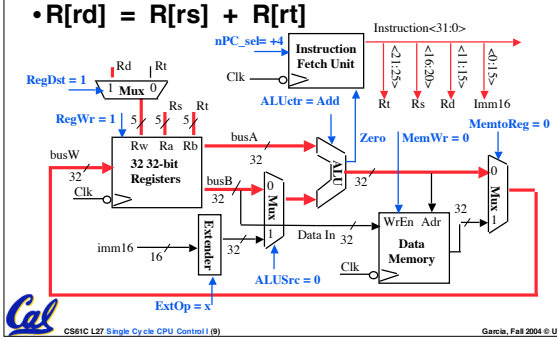
- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions



The Single Cycle Datapath during Add

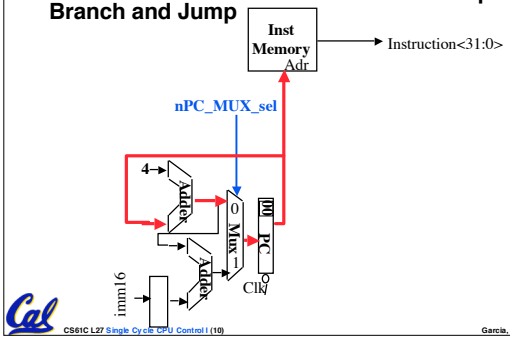


- R[rd] = R[rs] + R[rt]

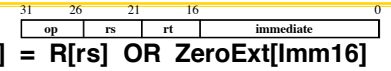


Instruction Fetch Unit at the End of Add

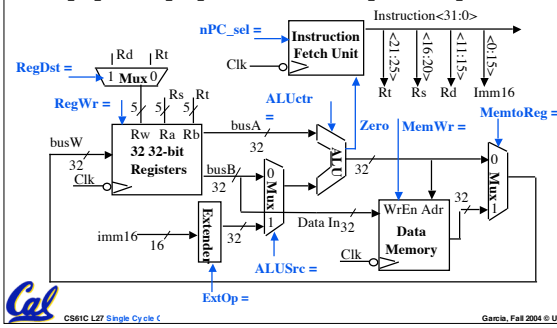
- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump



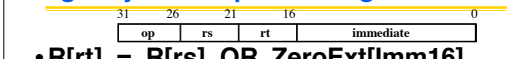
Single Cycle Datapath during Or Immediate?



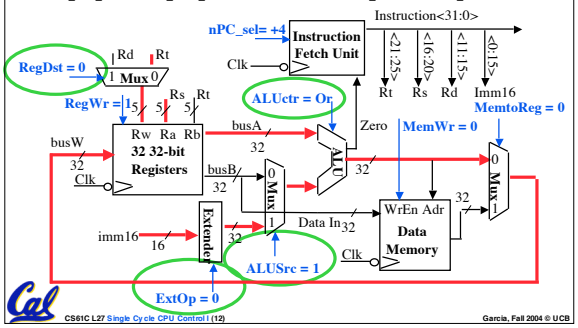
- R[rt] = R[rs] OR ZeroExt[Imm16]



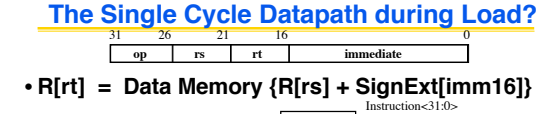
Single Cycle Datapath during Or Immediate?



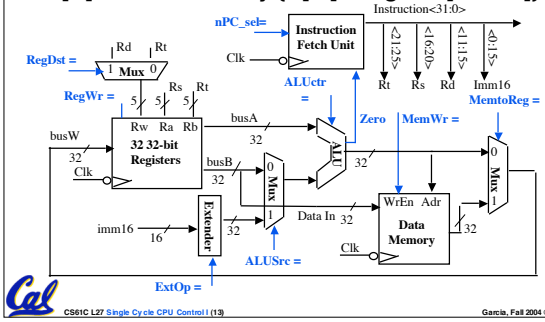
- R[rt] = R[rs] OR ZeroExt[Imm16]



The Single Cycle Datapath during Load?

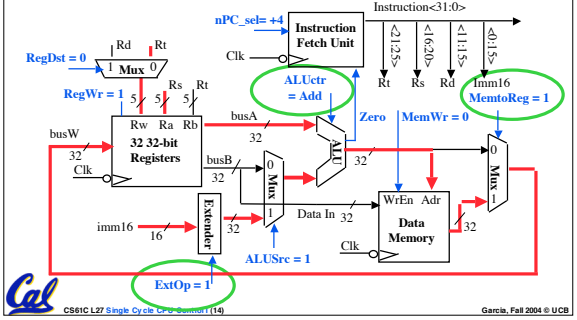


• R[rt] = Data Memory {R[rs] + SignExt[imm16]}

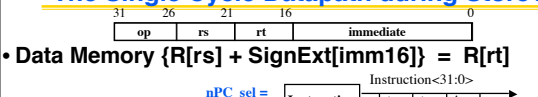


The Single Cycle Datapath during Load

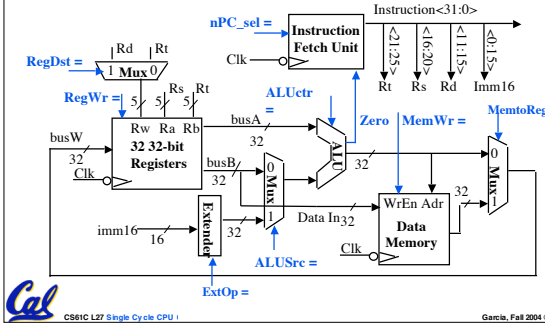
• R[rt] = Data Memory {R[rs] + SignExt[imm16]}



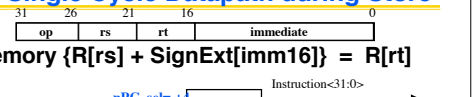
The Single Cycle Datapath during Store?



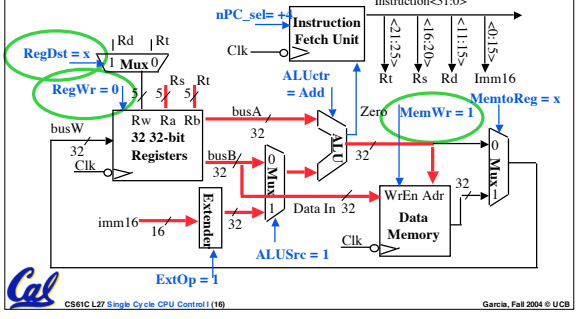
• Data Memory {R[rs] + SignExt[imm16]} = R[rt]



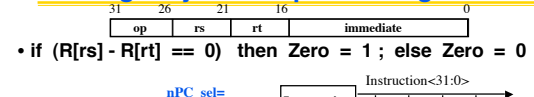
The Single Cycle Datapath during Store



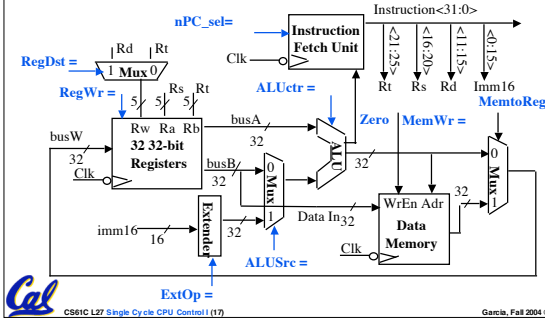
• Data Memory {R[rs] + SignExt[imm16]} = R[rt]



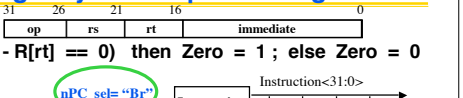
The Single Cycle Datapath during Branch?



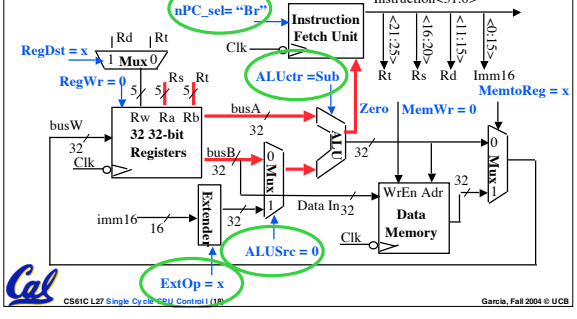
• if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



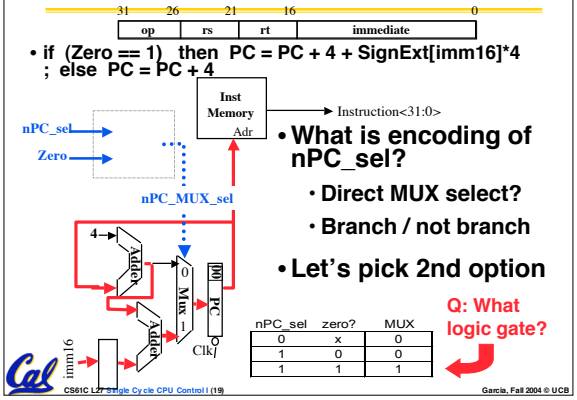
The Single Cycle Datapath during Branch



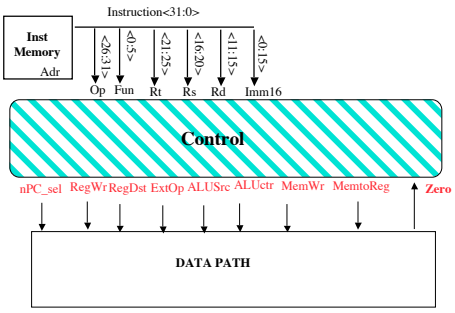
• if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Instruction Fetch Unit at the End of Branch



Step 4: Given Datapath: RTL -> Control

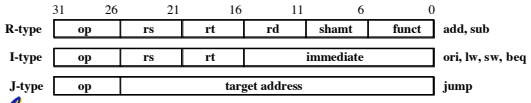


A Summary of the Control Signals (1/2)

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



Peer Instruction

A. Our ALU is a synchronous device

B. We could have used tri-state devices instead of a MUX to feed busW, the register write data line

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

Summary: 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