CS61C : Machine Structures

Lecture 5.2.1

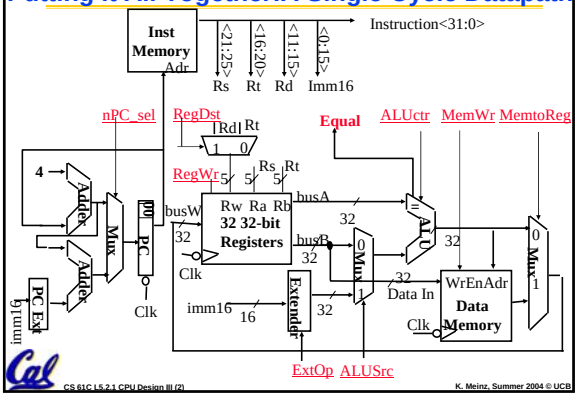
CPU Design III: Control

2004-07-21

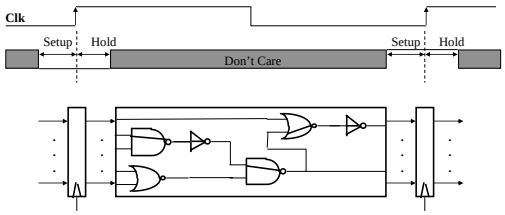
Kurt Meinz

`inst.eecs.berkeley.edu/~cs61c`

Putting it All Together:A Single Cycle Datapath

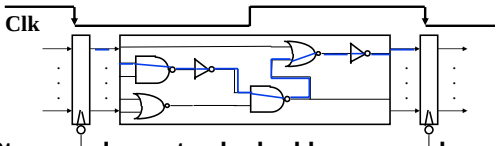


Recall: Clocking Methodology



- All storage elements are clocked by the same clock edge
- **Cycle Time** = CLK-to-Q + **Longest Delay Path** + Setup + Clock Skew

Clocking Methodology

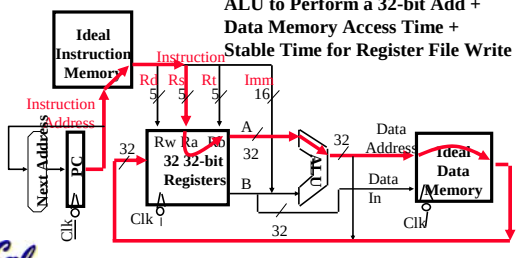


- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF, and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

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

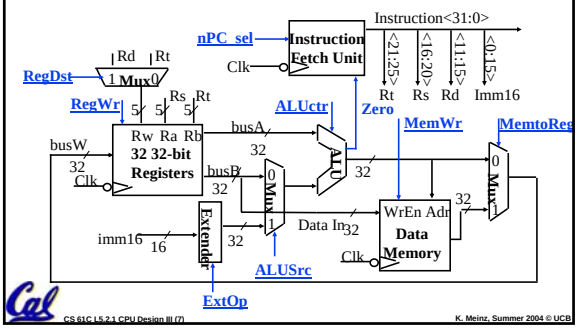


How to Design a Processor: step-by-step

- 1. Analyze instruction set architecture (ISA) => datapath requirements
  - meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
- 2. Select set of datapath components and establish clocking methodology
- 3. Assemble datapath meeting requirements
- 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
- 5. Assemble the control logic (hard part!)

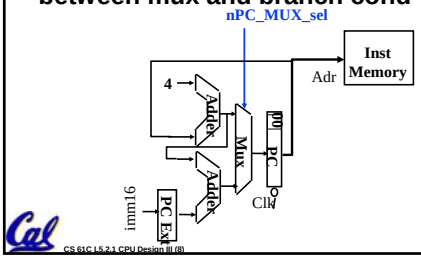
A Single Cycle Datapath

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



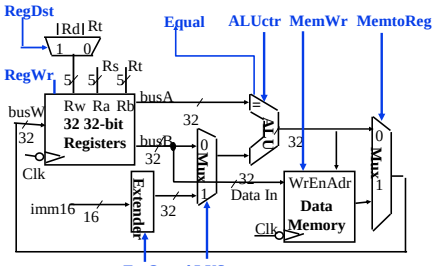
Meaning of the Control Signals

- 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

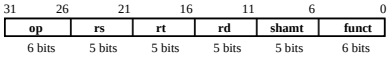


Meaning of the Control Signals

- 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

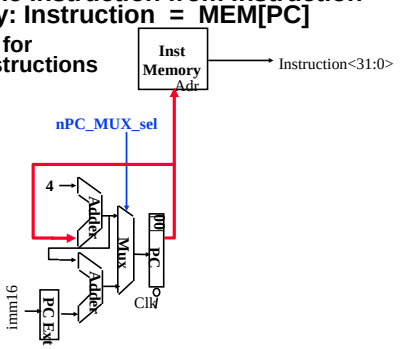


`add rd, rs, rt`

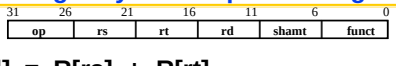
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

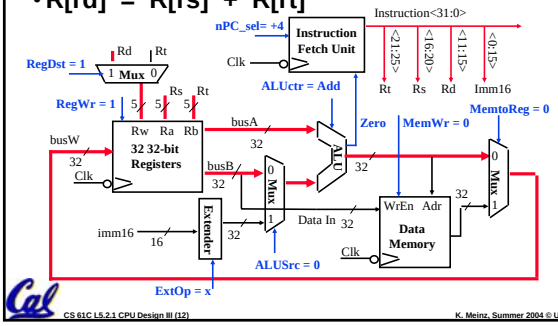
- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions



The Single Cycle Datapath during Add

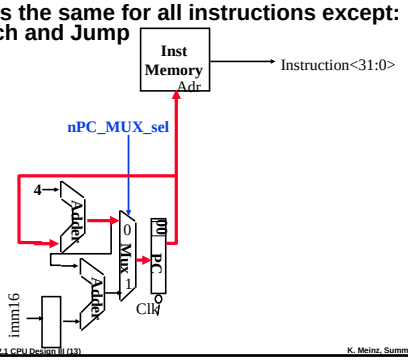


- R[rd] = R[rs] + R[rt]

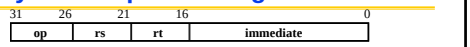


Instruction Fetch Unit at the End of Add

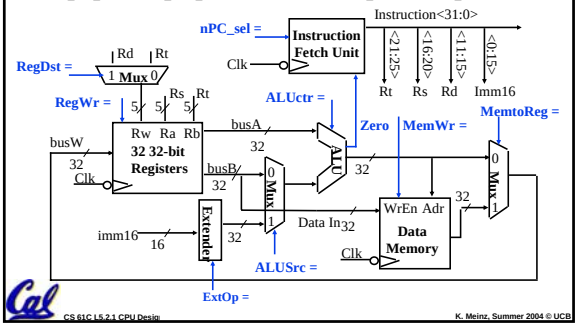
- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump



Single Cycle Datapath during Or Immediate?

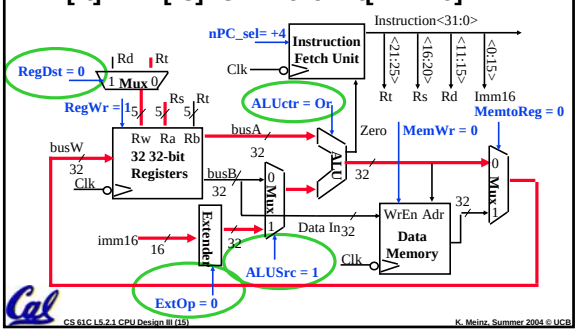


- R[rt] = R[rs] OR ZeroExt[Imm16]

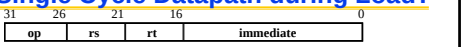


Single Cycle Datapath during Or Immediate

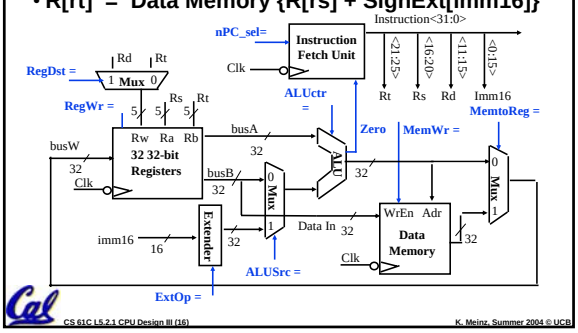
- R[rt] = R[rs] OR ZeroExt[Imm16]



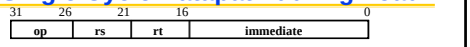
The Single Cycle Datapath during Load?



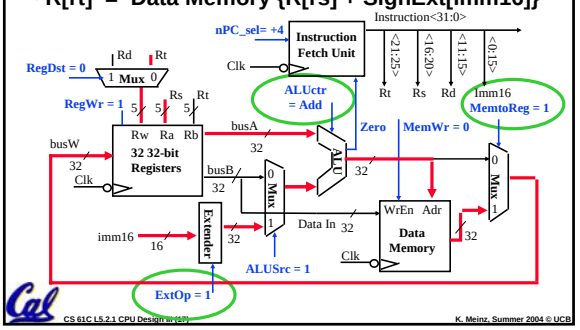
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



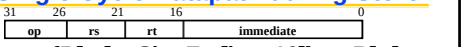
The Single Cycle Datapath during Load



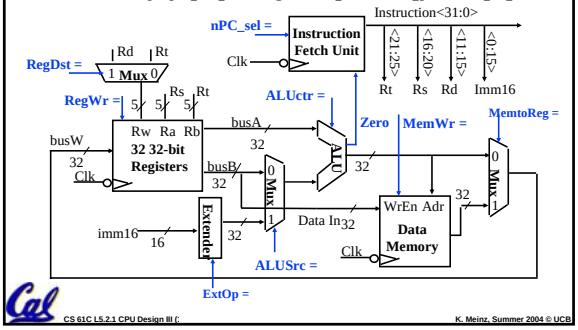
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



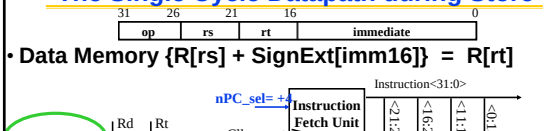
The Single Cycle Datapath during Store?



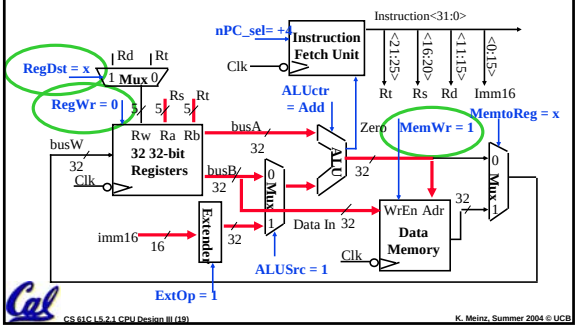
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



The Single Cycle Datapath during Store



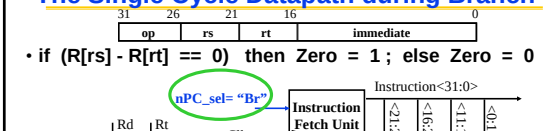
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



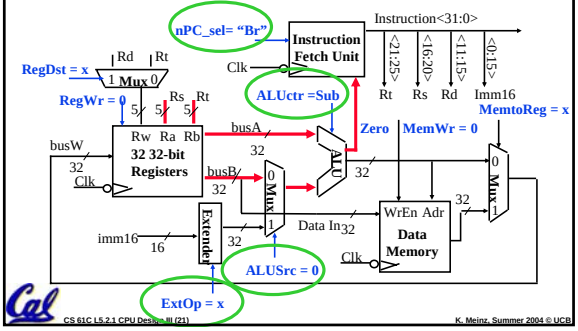
The Single Cycle Datapath during Branch?

- 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

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

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

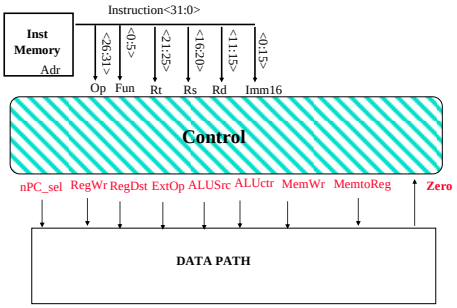
Q: What logic gate?

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

How to Design a Processor: step-by-step

- 1. Analyze instruction set architecture (ISA) => datapath requirements
  - meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
- 2. Select set of datapath components and establish clocking methodology
- 3. Assemble datapath meeting requirements
- 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
- 5. Assemble the control logic (hard part!)

Step 5: Given Datapath: RTL -> Control



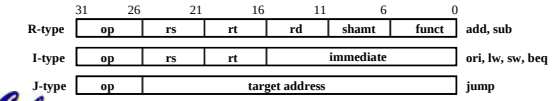
A Summary of the Control Signals (1/2)

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)

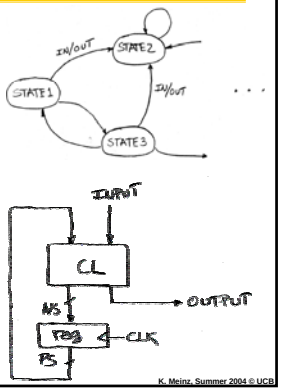
See Appendix A → func / op

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



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!

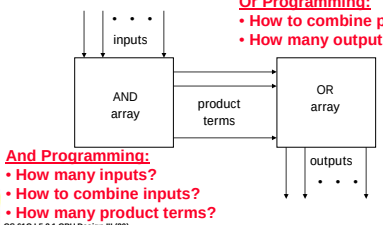
- They define
  - How output signals respond to input signals and previous state.
  - How we change states depending on input signals and previous state
- The output signals could be our familiar control signals
  - Some control signals may only depend on CL, not on state at all…
- We could implement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

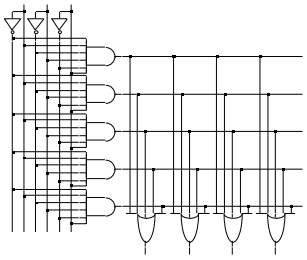
Product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

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

reuse of terms; 5 product terms

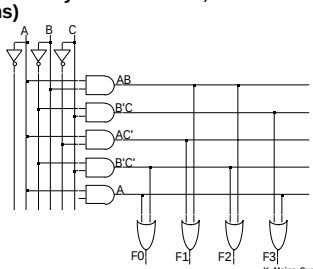
Before Programming

- **All possible connections available before "programming"**



After Programming

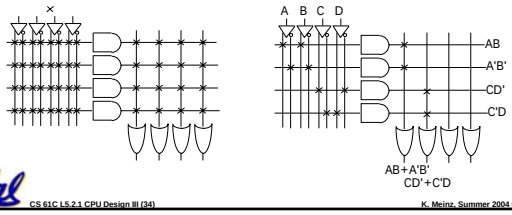
- **Unwanted connections are "blown"**
  - **Fuse (normally connected, break unwanted ones)**
  - **Anti-fuse (normally disconnected, make wanted connections)**



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



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**