CS 61C:
Great Ideas in Computer Architecture
Building Blocks for Datapaths

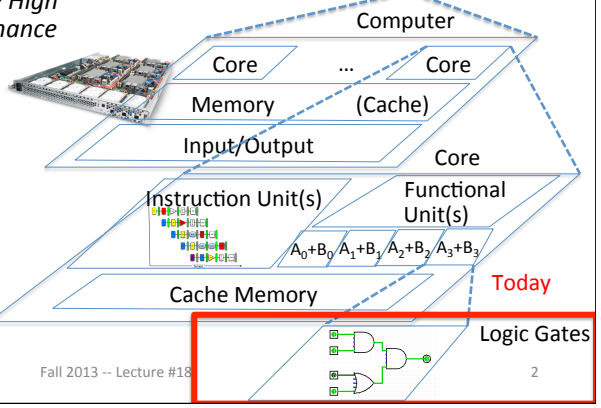
Instructor:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

You are Here!

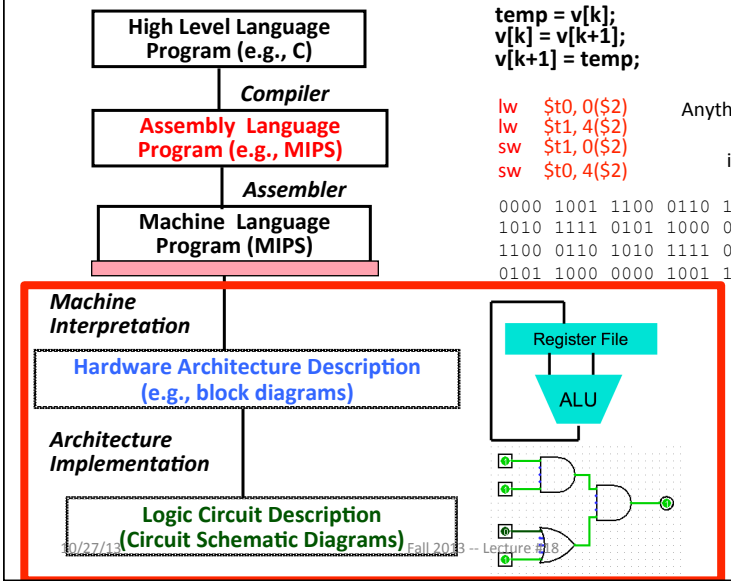
Software *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance



Levels of Representation/ Interpretation



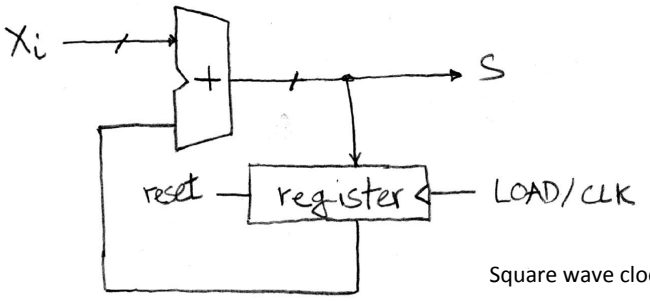
Agenda

- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

Agenda

- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

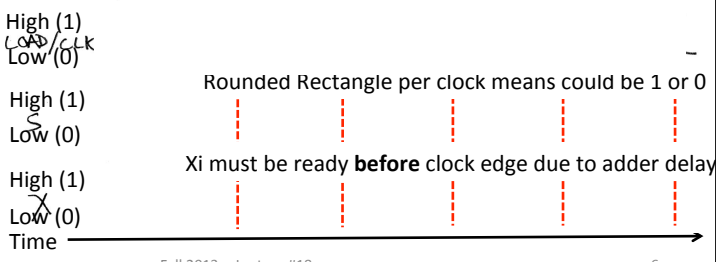
Last Time: Summation Circuit



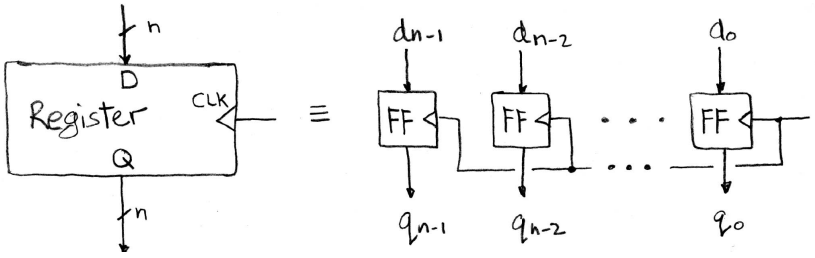
Register is used to hold up the transfer of data to adder

Square wave clock sets when things change

Rough timing …



Register Internals



- n instances of a "Flip-Flop"
- Flip-flop name because the output flips and flops between 0 and 1
- D is "data input", Q is "data output"
- Also called "D-type Flip-Flop"

Camera Analogy Timing Terms

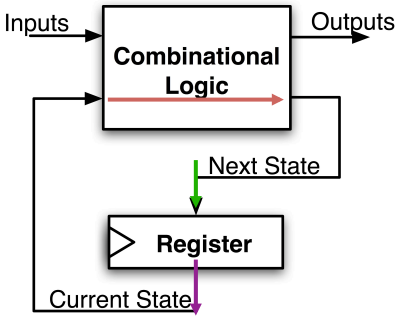
- Want to take a portrait – timing right before and after taking picture
- *Set up time* – don't move since about to take picture (open camera shutter)
- *Hold time* – need to hold still after shutter opens until camera shutter closes
- *Time click to data* – time from open shutter until can see image on output (viewfinder)

Hardware Timing Terms

- Setup Time: when the input must be stable *before* the edge of the CLK
- Hold Time: when the input must be stable *after* the edge of the CLK
- "CLK-to-Q" Delay: how long it takes the output to change, measured from the edge of the CLK

FSM Maximum Clock Frequency

- What is the maximum frequency of this circuit?

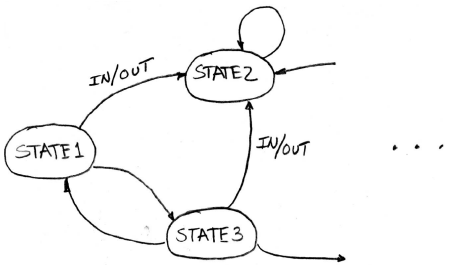


Hint:
Frequency = 1/Period

Max Delay = Setup Time + CLK-to-Q Delay + CL Delay

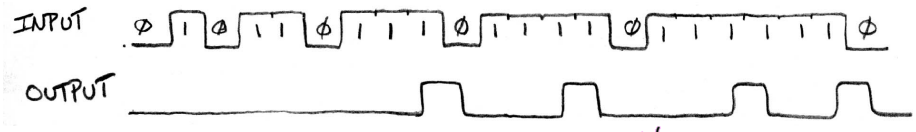
Another Great (Theory) Idea: Finite State Machines (FSM)

- You may have seen FSMs in other classes (e.g., CS70)
- Same basic idea
- Function can be represented with a "state transition diagram"
- With combinational logic and registers, any FSM can be implemented in hardware

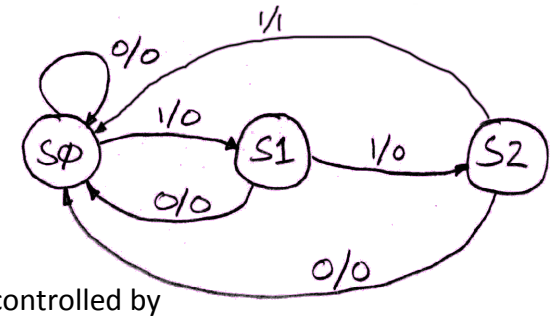


Example: 3 Ones FSM

FSM to detect the occurrence of 3 consecutive 1's in the Input



Draw the FSM …



Assume state transitions are controlled by the clock: On each clock cycle the machine checks the inputs and moves to a new state and produces a new output …

Hardware Implementation of FSM

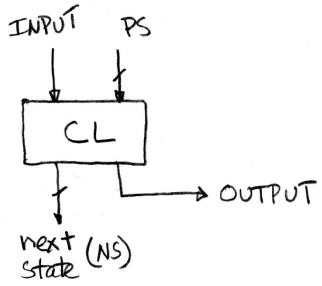
Register needed to hold a representation of the machine's state.
Unique bit pattern for each state.

Combinational logic circuit is used to implement a function maps from *present state (PS)* and *input* to *next state (NS)* and *output*.

The register is used to break the feedback path between Next State (NS) and Prior State (PS), controlled by the clock

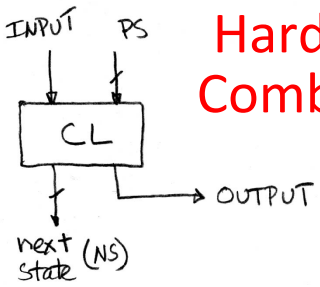
Hardware for FSM: Combinational Logic

Can look at its functional specification, truth table form



Truth table …

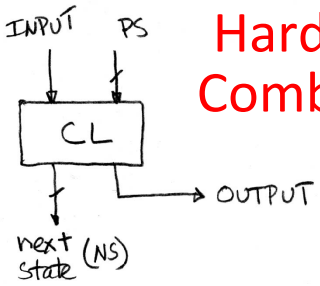
PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



Hardware for FSM: Combinational Logic

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

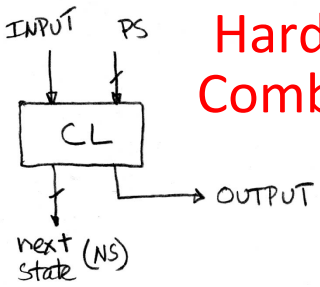


Hardware for FSM: Combinational Logic

Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



Hardware for FSM: Combinational Logic

Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

NS bit 0 is 1

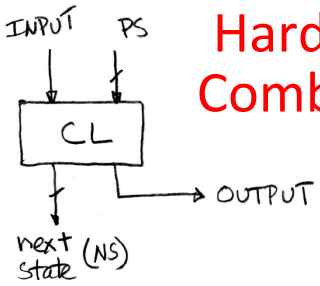
PS	Input
00	1

NS bit 1 is 1

PS	Input
01	1

Output is 1

PS	Input
10	1



Hardware for FSM: Combinational Logic

Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

- NS0 = $\overline{PS1} \bullet \overline{PS0} \bullet Input$
  - NS0 = ~PS1•~PS0•Input
- NS1 = $\overline{PS1} \bullet PS0 \bullet Input$
  - NS1 = ~PS1•PS0•Input
- Output= $PS1 \bullet \overline{PS0} \bullet Input$
  - Output= PS1•~PS0•Input

NS bit 0 is 1

PS	Input
00	1

NS bit 1 is 1

PS	Input
01	1

Output is 1

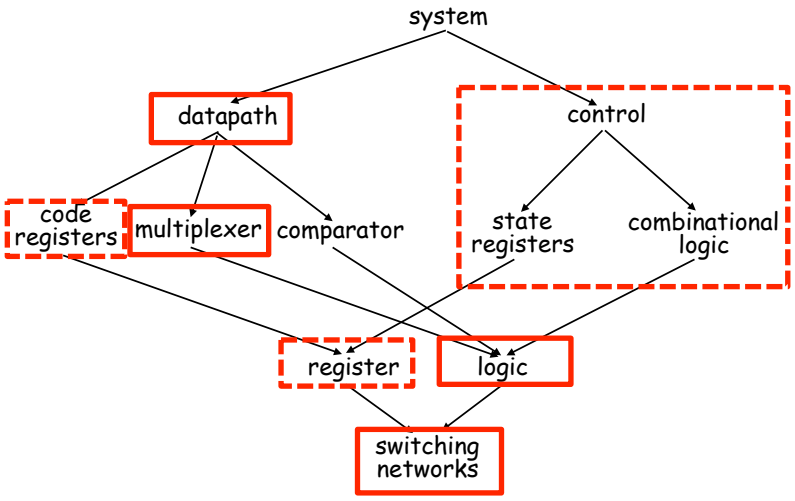
PS	Input
10	1

Administrivia

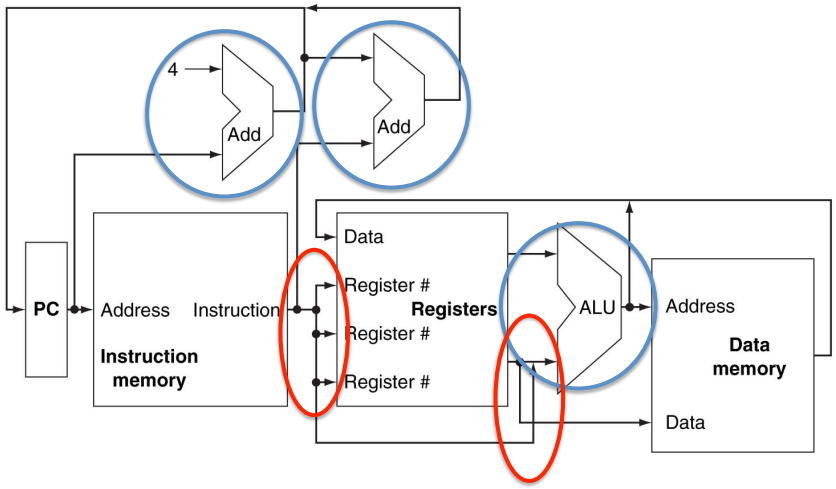
Agenda

- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

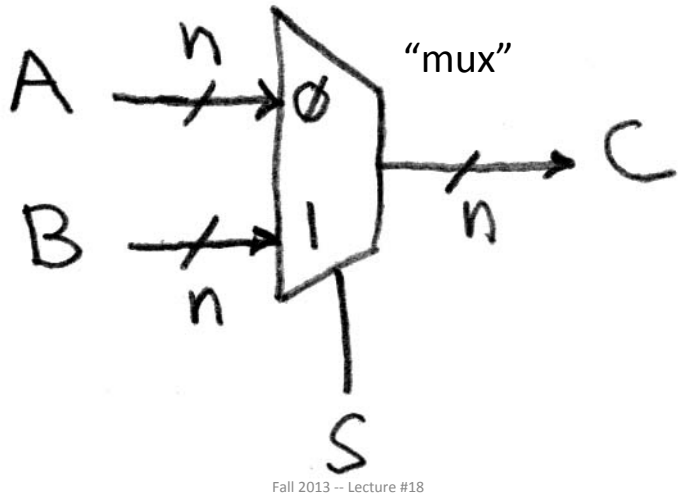
Design Hierarchy



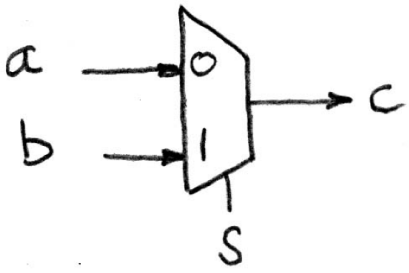
Conceptual MIPS Datapath



Data Multiplexer (e.g., 2-to-1 x n-bit-wide)



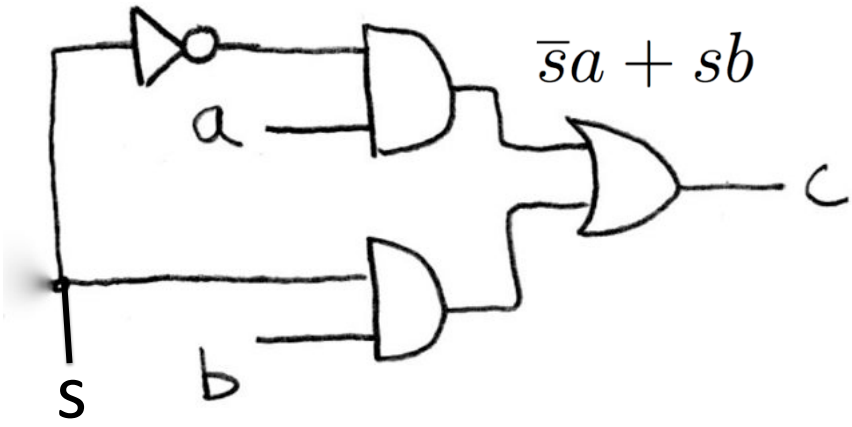
N Instances of 1-bit-Wide Mux



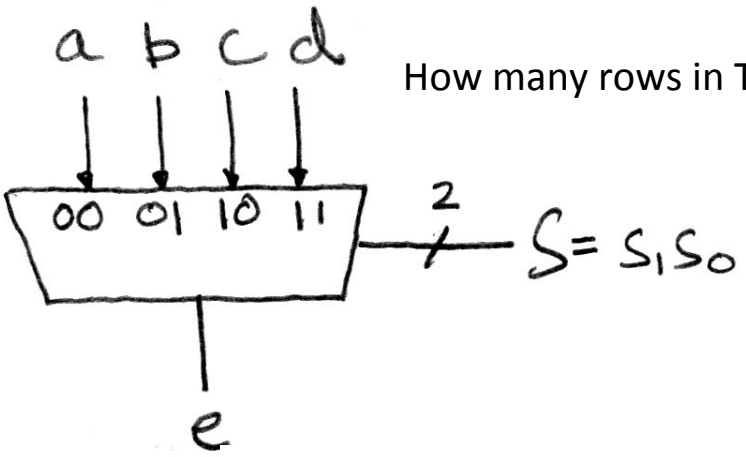
$$
\begin{aligned}
c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\
&= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\
&= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\
&= \bar{s}(a(1) + s((1)b) \\
&= \bar{s}a + sb
\end{aligned}
$$

s	ab	c
0	00	0
0	01	0
0	10	1
0	11	1
1	00	0
1	01	1
1	10	0
1	11	1

How Do We Build a 1-bit-Wide Mux (in Logisim)?



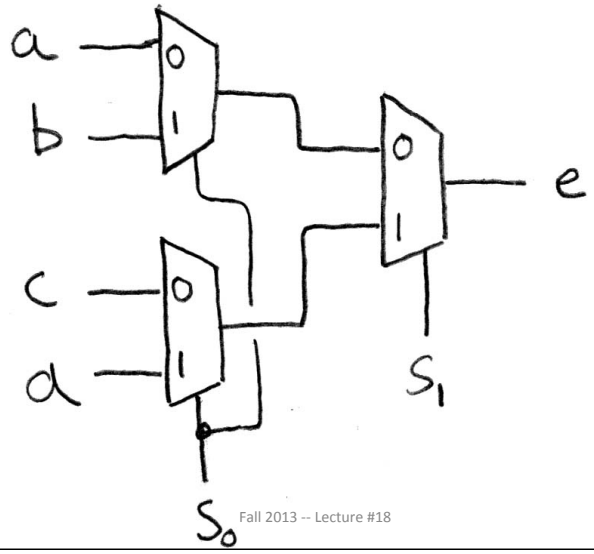
4-to-1 Multiplexer



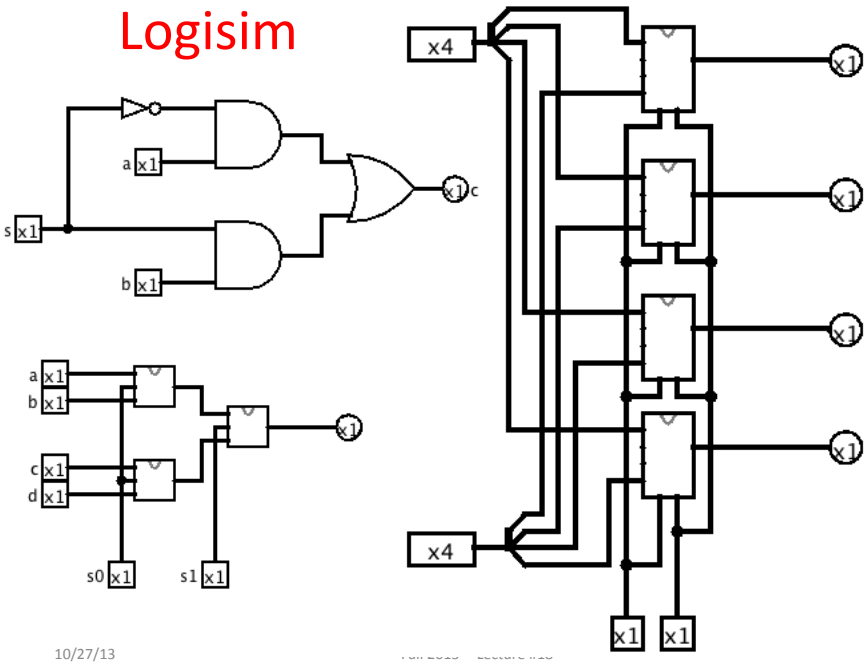
How many rows in TT?

$$e = \overline{s_1}\,\overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$

Alternative Hierarchical Approach (in Logisim)



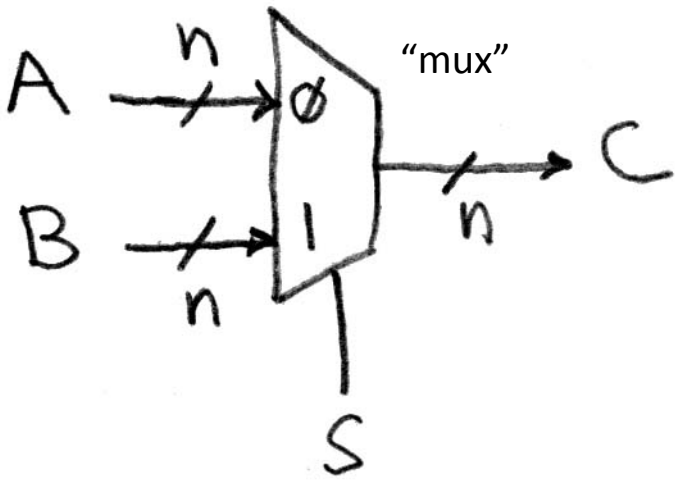
Logisim



Subcircuits

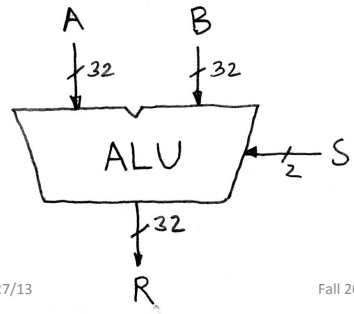
- Subcircuit: Logisim equivalent of procedure or method
  - Every project is a hierarchy of subcircuits

N-bit-wide Data Multiplexer (in Logisim + tunnel)



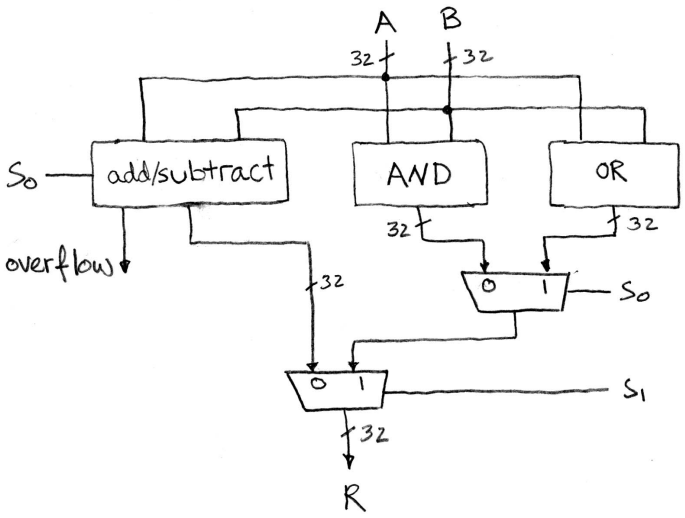
Arithmetic and Logic Unit

- Most processors contain a special logic block called "Arithmetic and Logic Unit" (ALU)
- We'll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR



when S=00, R=A+B
when S=01, R=A-B
when S=10, R=A AND B
when S=11, R=A OR B

Simple ALU



Adder/Subtractor: One-bit adder Least Significant Bit

$$\begin{array}{cccc|c} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_0 = a_0 \text{ XOR } b_0$$
$$c_1 = a_0 \text{ AND } b_0$$

Adder/Subtractor: One-bit adder (1/2)

…

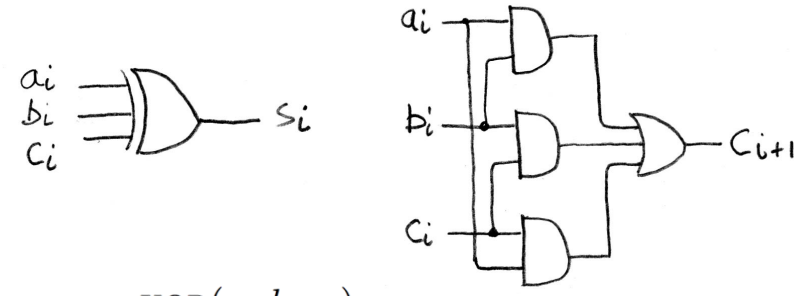
$$\begin{array}{cccc|c} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i =$$
$$c_{i+1} =$$

Adder/Subtractor: One-bit Adder (2/2)

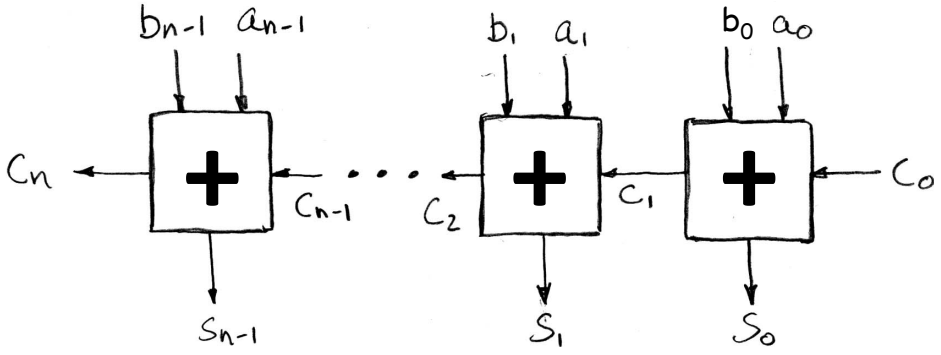
…



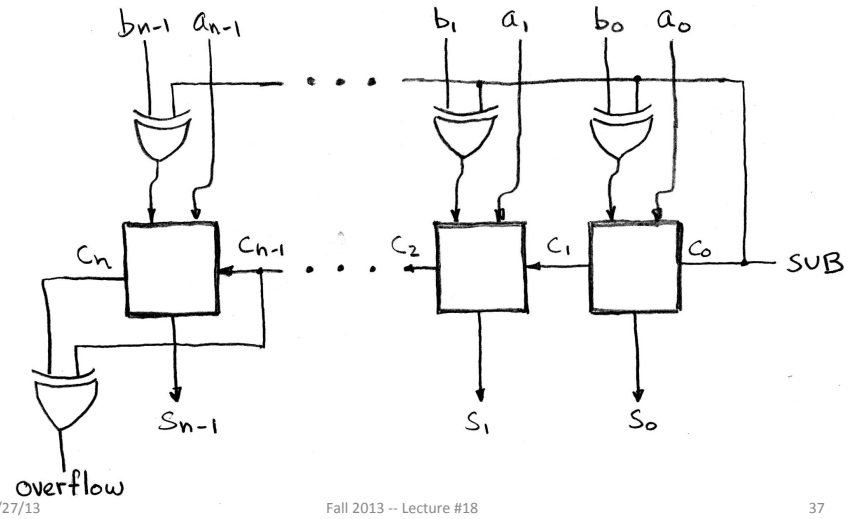
$$s_i = \text{XOR}(a_i, b_i, c_i)$$
$$c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

N x 1-bit Adders ⇒ 1 N-bit Adder

Connect Carry Out i-1 to Carry in i:



Twos Complement Adder/Subtractor



Critical Path

- When setting clock period in synchronous systems, must allow for worst case
- Path through combinational logic that is worst case called "critical path"
  - Can be estimated by number of "gate delays": Number of gates must go through in worst case
- Idea: Doesn't matter if speedup other paths if don't improve the critical path
- What might critical path of ALU?

Agenda

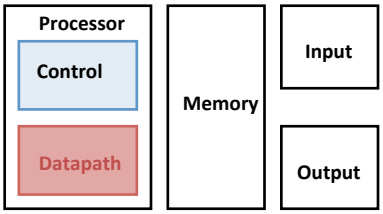
- Multiplexer
- ALU Design
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

Agenda

- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

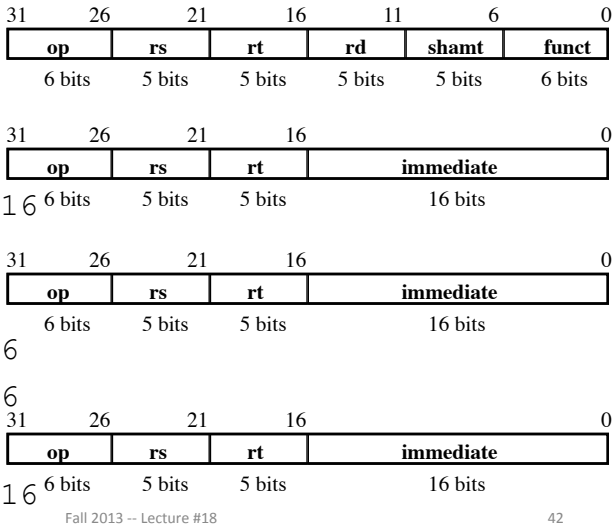
Processor Design Process

- Five 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
     - Formulate Logic Equations
     - Design Circuits



The MIPS-lite Subset

- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- OR Immediate:
  - `ori rt,rs,imm16`
- LOAD and STORE Word
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`
- BRANCH:
  - `beq rs,rt,imm16`



Register Transfer Language (RTL)

- RTL gives the meaning of the instructions

```
{op , rs , rt , rd , shamt , funct} ← MEM[ PC ]
{op , rs , rt ,   Imm16} ← MEM[ PC ]
```

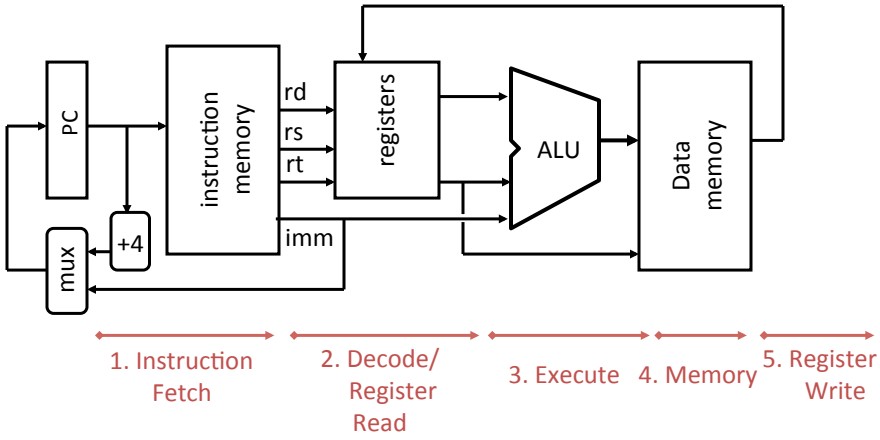
- All start by fetching the instruction

```
Inst    Register Transfers
ADDU    R[rd] ← R[rs] + R[rt]; PC ← PC + 4
SUBU    R[rd] ← R[rs] – R[rt]; PC ← PC + 4
ORI     R[rt] ← R[rs] | zero_ext(Imm16); PC ← PC + 4
LOAD    R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4
STORE   MEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]; PC ← PC + 4
BEQ     if ( R[rs] == R[rt] )
            then PC ← PC + 4 + (sign_ext(Imm16) || 00)
            else PC ← PC + 4
```


Step 1: Requirements of the Instruction Set

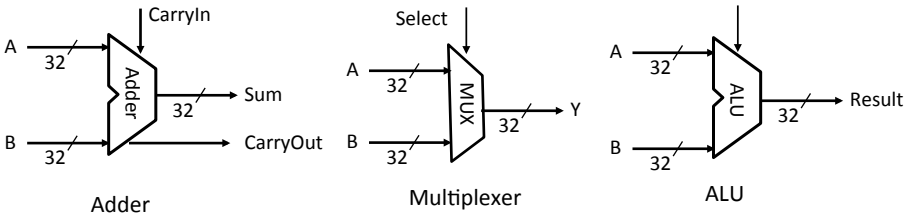
- Memory (MEM)
  - Instructions & data (will use one for each: really caches)
- Registers (R: 32 x 32)
  - Read *rs*
  - Read *rt*
  - Write *rt* or *rd*
- PC
- Extender (sign/zero extend)
- Add/Sub/OR unit for operation on register(s) or extended immediate
- Add 4 (+ maybe extended immediate) to PC
- Compare if registers equal?

Generic Steps of Datapath



Step 2: Components of the Datapath

- Combinational Elements
- State Elements + Clocking Methodology
- Building Blocks



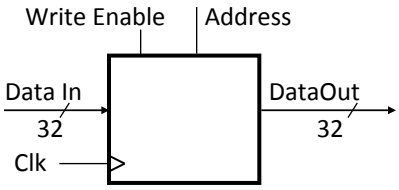
ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:

```
ADDU   R[rd] = R[rs] + R[rt]; ...
SUBU   R[rd] = R[rs] – R[rt]; ...
ORI    R[rt] = R[rs] | zero_ext(Imm16)...
BEQ    if ( R[rs] == R[rt] )...
```

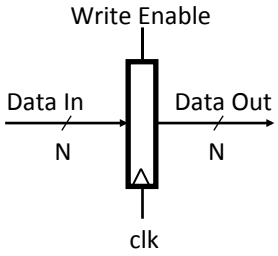
- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if A < B, 0 otherwise)
- ALU from Appendix C, section C.5

Storage Element: Idealized Memory



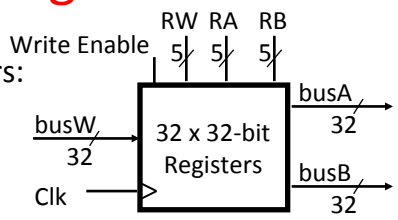
- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block: Address valid ⇒ Data Out valid after "access time"

Storage Element: Register (Building Block)



- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - Negated (or deasserted) (0): Data Out will not change
  - Asserted (1): Data Out will become Data In on rising edge of clock

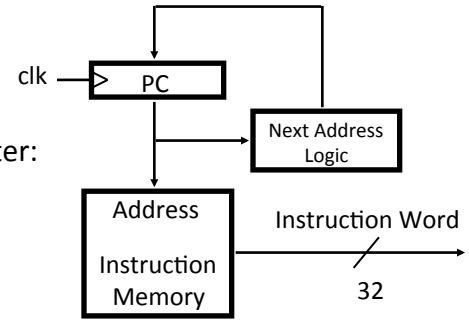
Storage Element: Register File



- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
  - Clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after "access time."

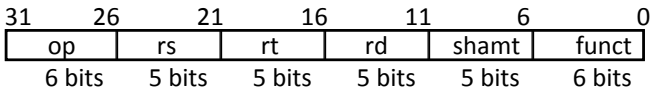
Step 3: Assemble DataPath Meeting Requirements

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

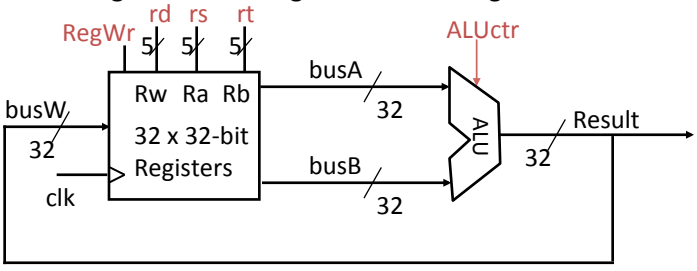


Step 3: Add & Subtract

- `R[rd] = R[rs] op R[rt] (addu rd,rs,rt)`
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields



  - ALUctr and RegWr: control logic after decoding the instruction

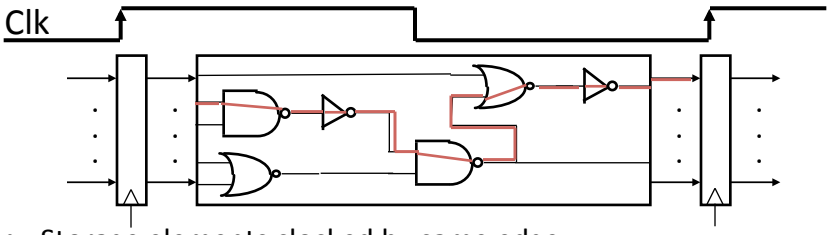


- … Already defined the register file & ALU

Agenda

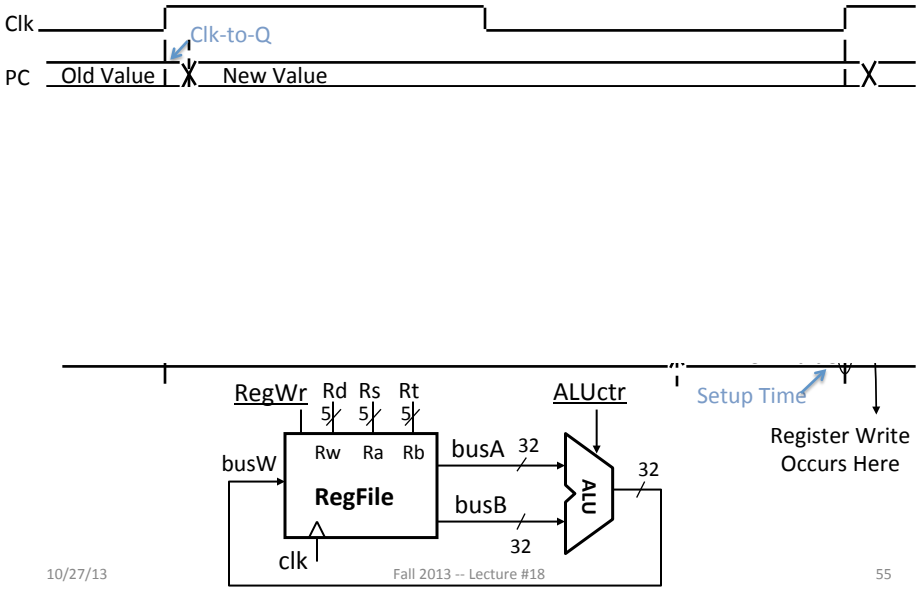
- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

Clocking Methodology



- Storage elements clocked by same edge
- "Critical path" (longest path through logic) determines length of clock period
- Have to allow for Clock-to-Q and Setup Times too
- This lecture (and P&H sections) 4.3-4.4 do whole instruction in 1 clock cycle for pedagogic reasons
  - Project 4 will do it in 2 clock cycles via simple pipelining
  - Soon explain pipelining and use 5 clock cycles per instruction

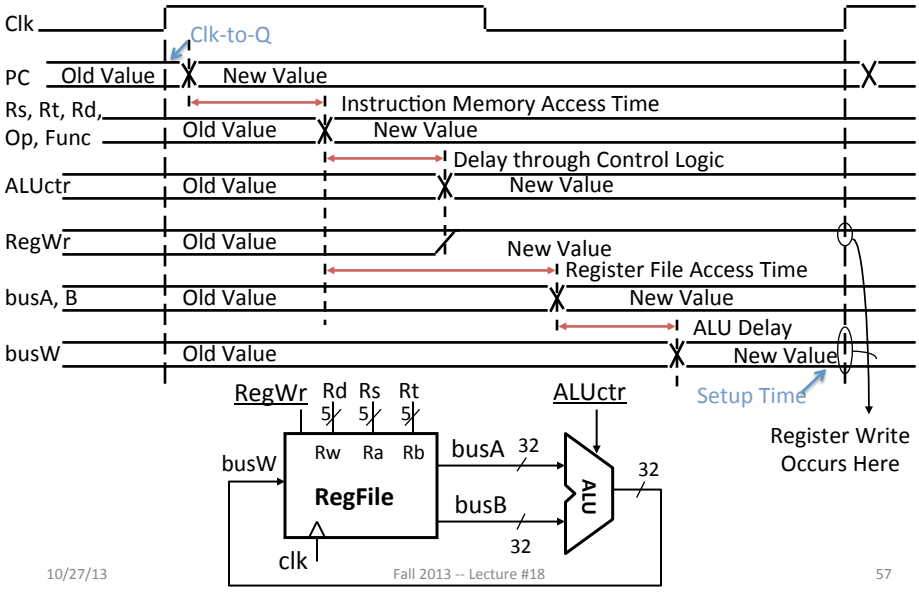
Register-Register Timing: One Complete Cycle



Agenda

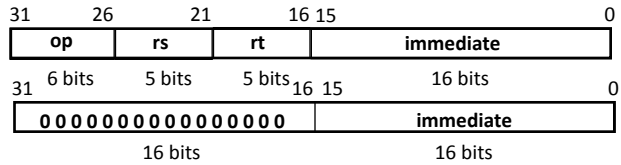
- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

Register-Register Timing: One Complete Cycle

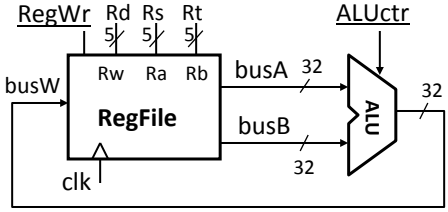


Logical Operations with Immediate

- R[rt] = R[rs] op ZeroExt[imm16]

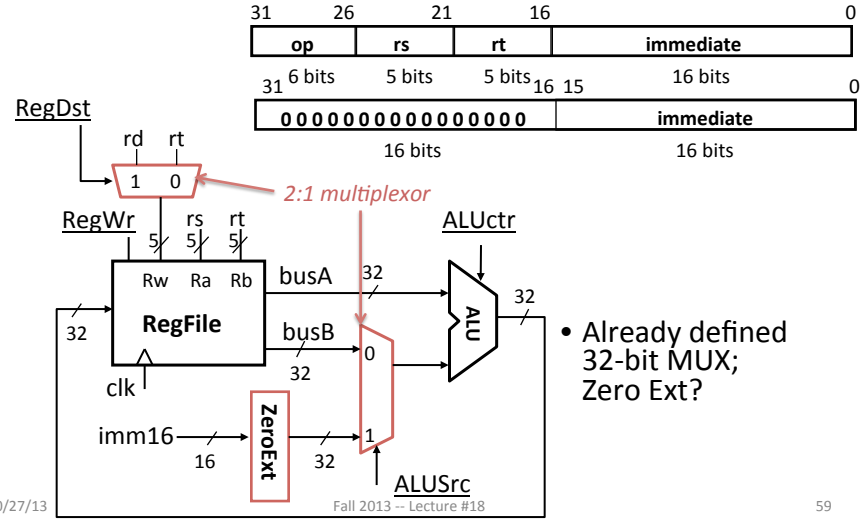


But we're writing to Rt register??
And immediate ALU input??



Logical Operations with Immediate

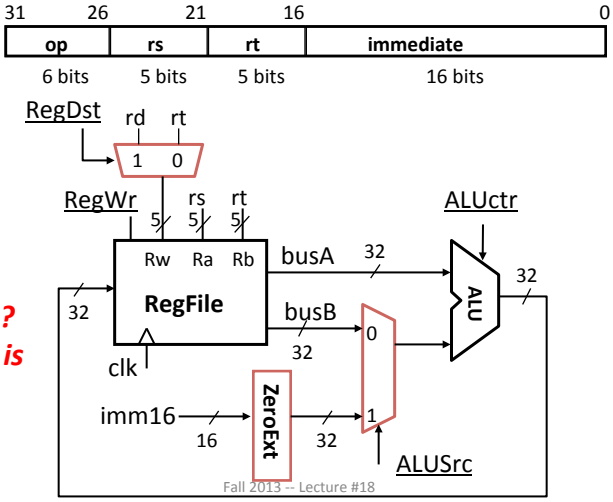
- R[rt] = R[rs] op ZeroExt[imm16]



- Already defined 32-bit MUX; Zero Ext?

Load Operations

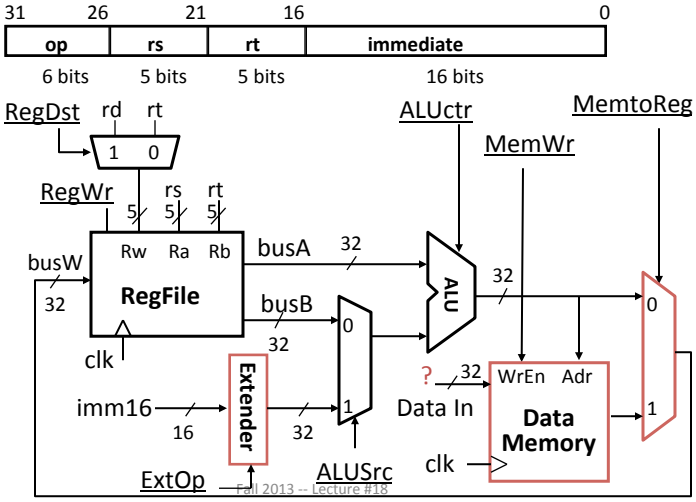
- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



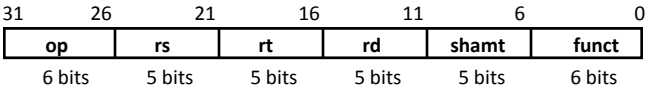
What sign extending?? And where is Mem??

Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



RTL: The `Add` Instruction

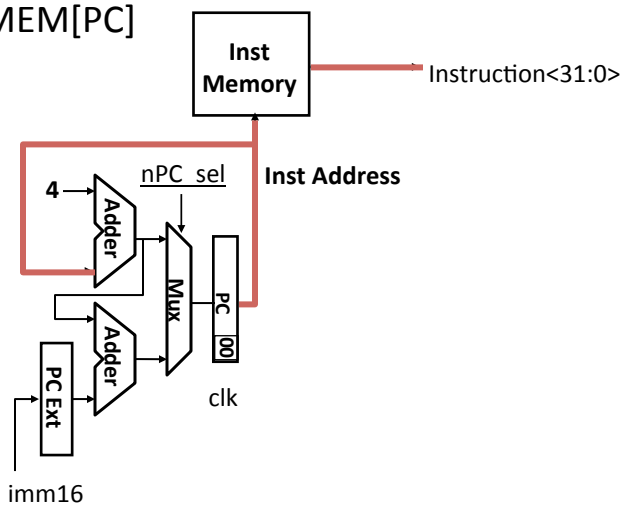


`add rd, rs, rt`

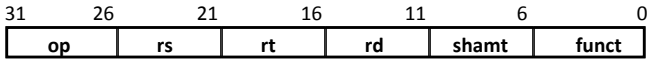
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at Beginning of Add

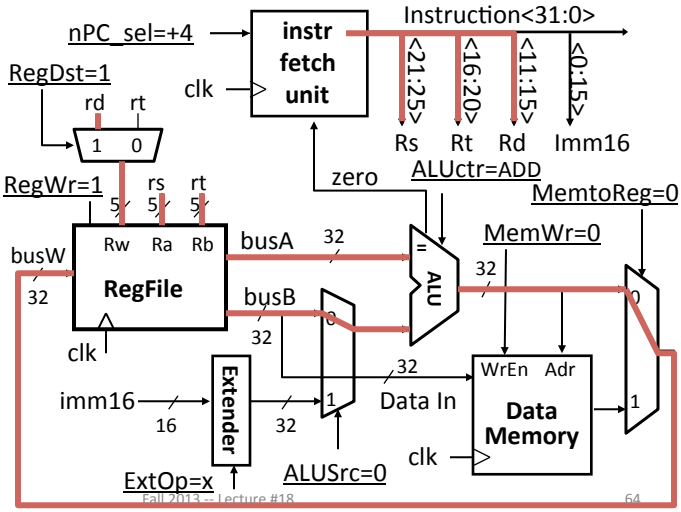
- Fetch the instruction from Instruction memory:
  Instruction = MEM[PC]
  - same for all instructions



Single Cycle Datapath during Add

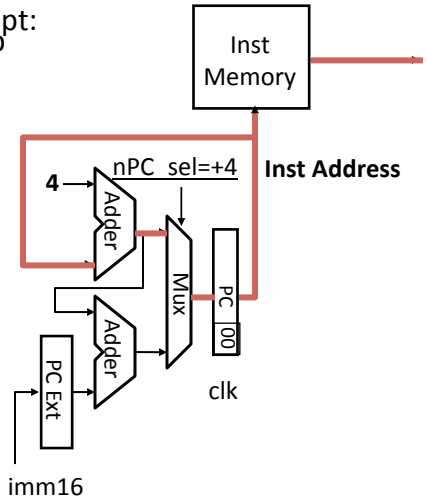


R[rd] = R[rs] + R[rt]

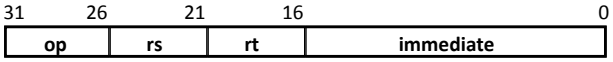


Instruction Fetch Unit at End of `Add`

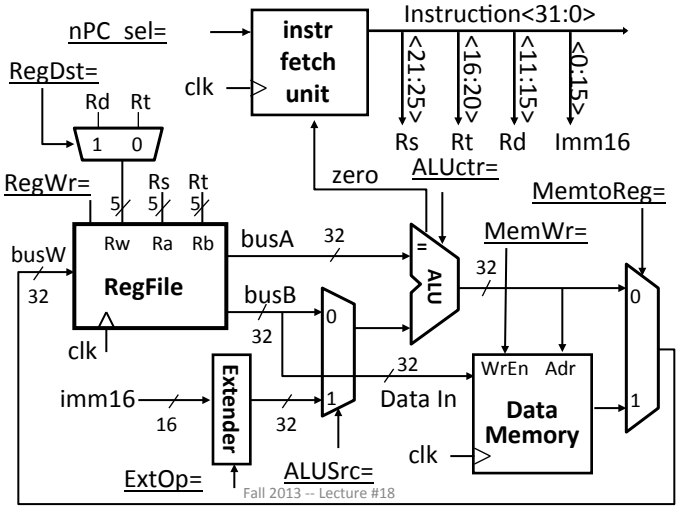
- PC = PC + 4
  - Same for all instructions except: Branch and Jump



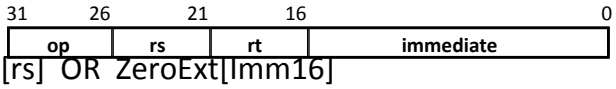
Single Cycle Datapath during `OR Immediate`



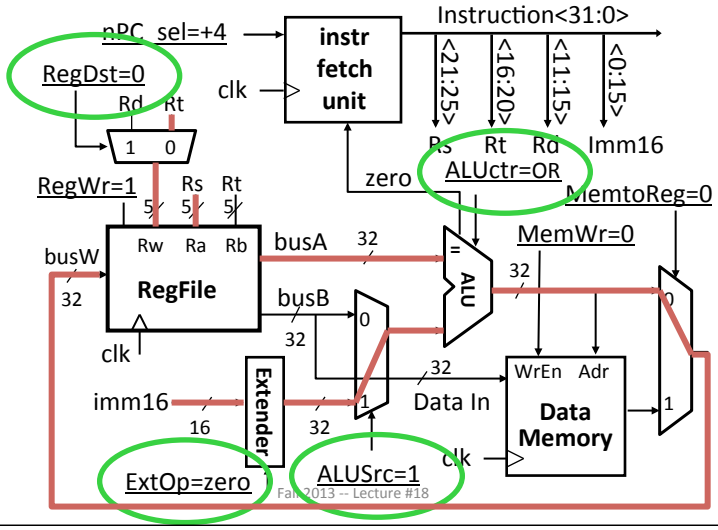
- R[rt] = R[rs] OR ZeroExt[Imm16]



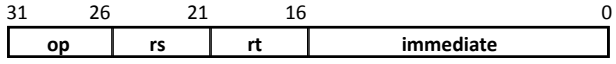
Single Cycle Datapath during OR Immediate



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

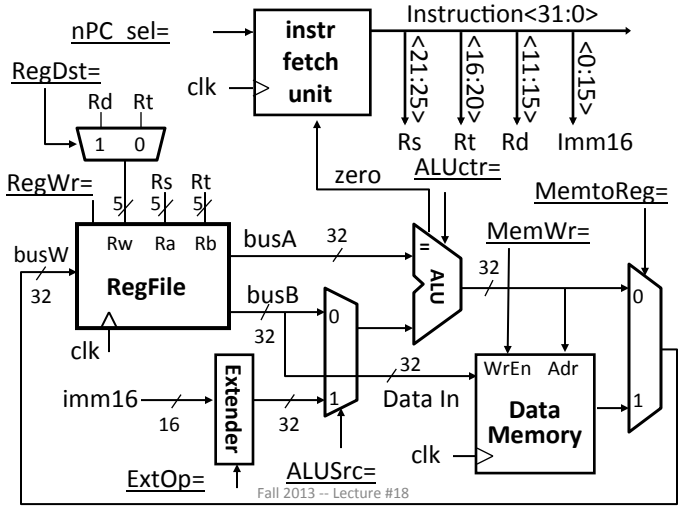


Single Cycle Datapath during Load

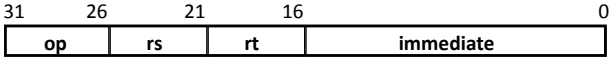


- R[rt] = Data Memory {R[rs] + SignExt[imm16]}

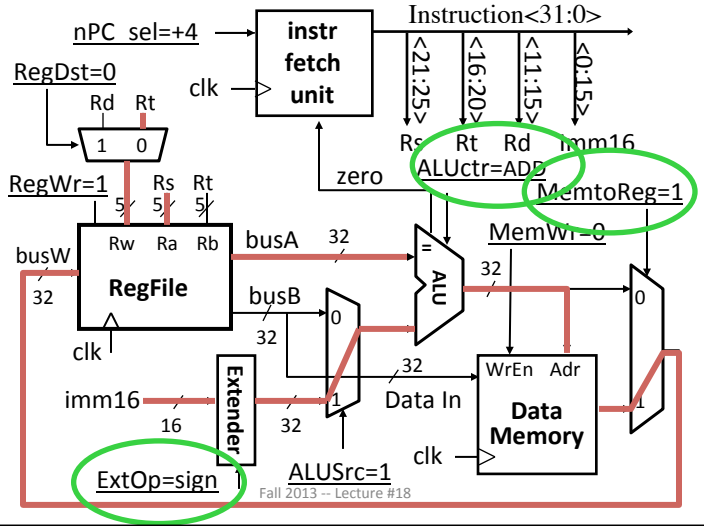
Student Roulette



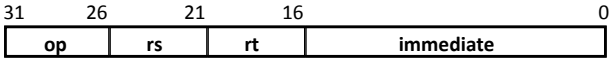
Single Cycle Datapath during Load



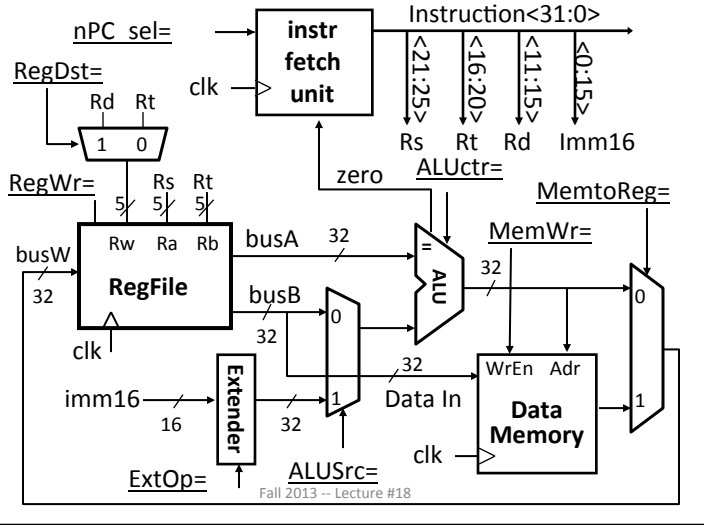
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



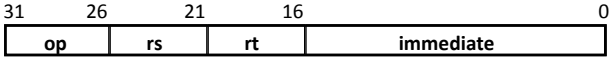
Single Cycle Datapath during Store



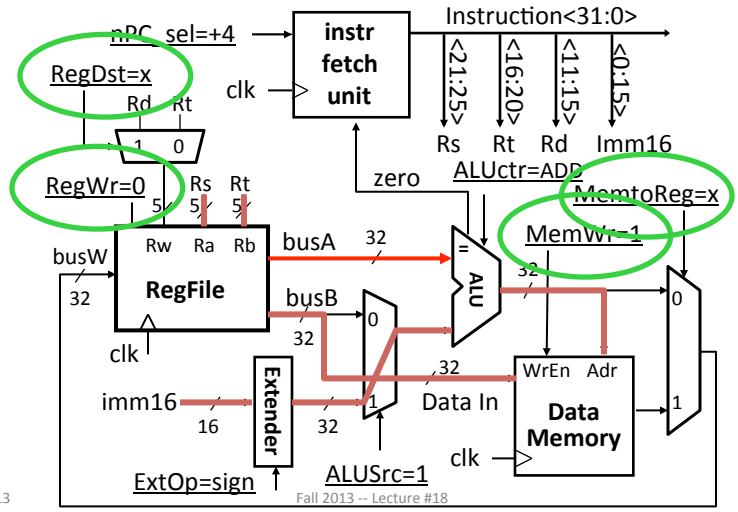
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



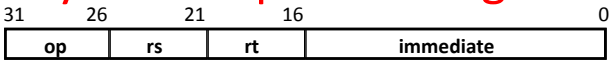
Single Cycle Datapath during Store



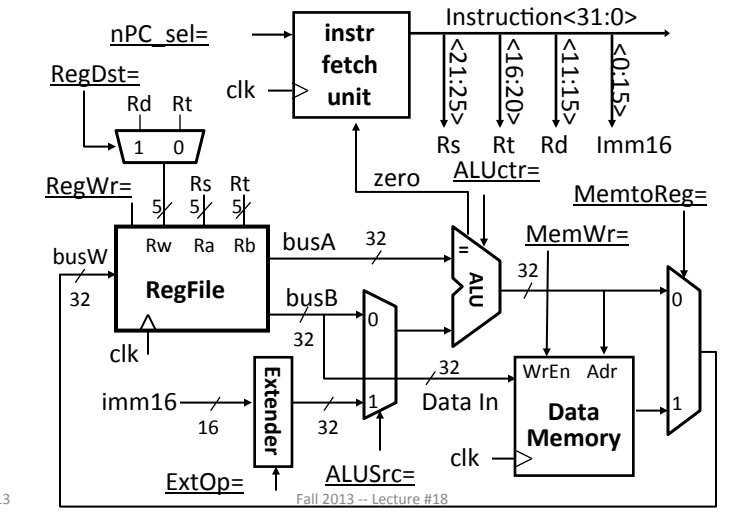
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



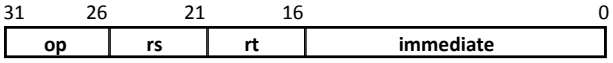
Single Cycle Datapath during Branch



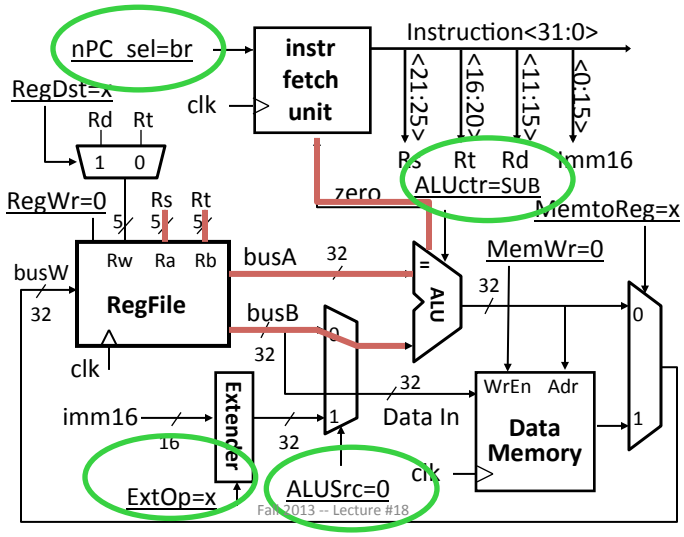
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



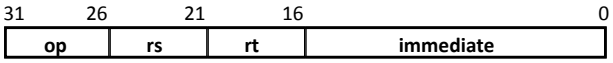
Single Cycle Datapath during Branch



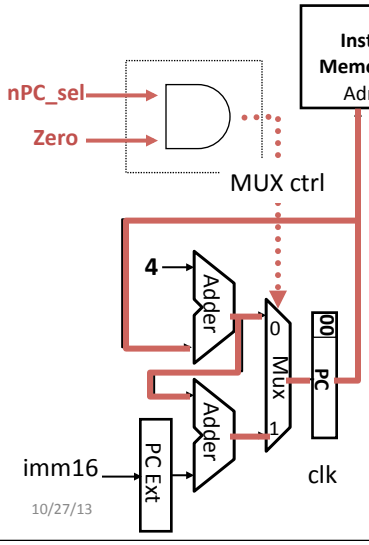
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Instruction Fetch Unit at the End of `Branch`



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



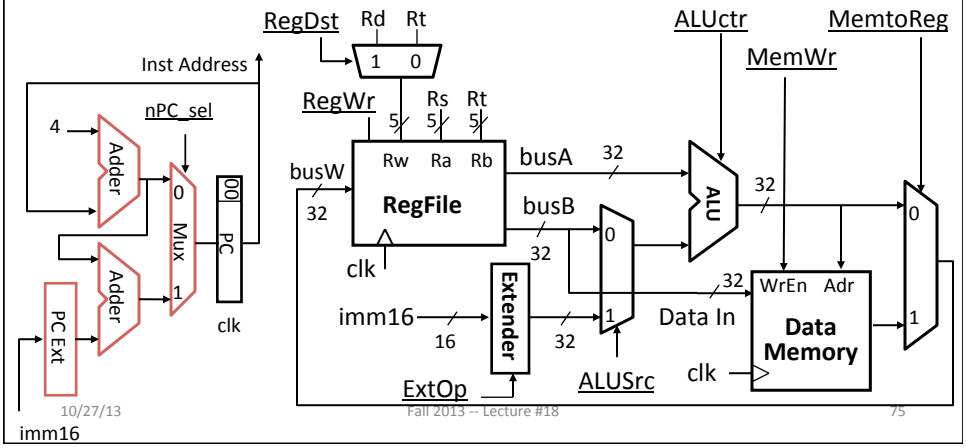
- What is encoding of nPC_sel?
  - Direct MUX select?
  - Branch inst. / not branch
- Let's pick 2nd option

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

Q: What logic gate?

Summary: Datapath's 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

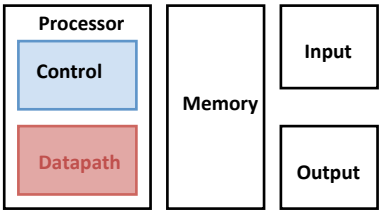


Agenda

- Timing and State Machines
- Datapath Elements: Mux + ALU
- MIPS-lite Datapath
- CPU Timing
- MIPS-lite Control
- And, in Conclusion, …

And. in Conclusion, … Single-Cycle Processor

- Use muxes to select among input
  - S input bits selects 2^S inputs
  - Each input can be n-bits wide, independent of S
- Can implement muxes hierarchically
- Arithmetic circuits are a kind of combinational logic



- Five steps to processor design:
  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
     - Formulate Logic Equations
     - Design Circuits