• **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**

---

Today

You are Here!
Levels of Representation/Interpretation

High Level Language Program (e.g., C)  
Compiler
Assembly Language Program (e.g., RISC-V)  
Assembler
Machine Language Program (RISC-V)

Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)
Architecture Implementation
Logic Circuit Description (Circuit Schematic Diagrams)

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)

Anything can be represented as a number, i.e., data or instructions

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
Design Hierarchy

- System
  - Datapath
    - Code registers
    - Multiplexer
    - Comparator
  - Register
    - Logic
    - Switching networks
  - Combinational logic
  - State registers

- Control
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
Boolean Algebra

• Use plus for OR
  — “logical sum”
• Use product for AND (a•b or implied via ab)
  — “logical product”
• “Hat” to mean complement (NOT)
• Thus
  \[ab + a + \overline{c}\]
  \[= a\cdot b + a + \overline{c}\]
  \[= (a \text{ AND } b) \text{ OR } a \text{ OR (NOT } c\)\]
Boolean Algebra: Circuit & Algebraic Simplification

Original circuit:
\[ y = ((ab) + a) + c \]
\[ y = ab + a + c \]
\[ y = a(b + 1) + c \]
\[ y = a + c \]

Simplified circuit:

Equation derived from original circuit:

Algebraic simplification:
Laws of Boolean Algebra

\[ X \overline{X} = 0 \]
\[ X \cdot 0 = 0 \]
\[ X \cdot 1 = X \]
\[ X \cdot X = X \]
\[ X \cdot Y = Y \cdot X \]
\[ (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) \]
\[ X \cdot (Y + Z) = X \cdot Y + X \cdot Z \]
\[ X \cdot Y + X = X \]
\[ \overline{X} \cdot Y + X = X + Y \]
\[ \overline{X} \cdot Y = \overline{X} + \overline{Y} \]

\[ X + \overline{X} = 1 \]
\[ X + 1 = 1 \]
\[ X + 0 = X \]
\[ X + X = X \]
\[ X + Y = Y + X \]
\[ (X + Y) + Z = X + (Y + Z) \]
\[ X + Y + Z = (X + Y) \cdot (X + Z) \]
\[ (X + Y) \cdot X = X \]
\[ (X + Y) \cdot X = X \cdot Y \]
\[ (\overline{X} + Y) \cdot X = X \cdot Y \]
\[ \overline{X} + \overline{Y} = \overline{X} \cdot \overline{Y} \]

Complementarity
Laws of 0’s and 1’s
Identities
Idempotent Laws
Commutativity
Associativity
Distribution
Uniting Theorem
Uniting Theorem v. 2
DeMorgan’s Law
Boolean Algebraic Simplification Example

\[
y = ab + a + c
\]
Boolean Algebraic Simplification Example

\[
y = ab + a + c = a(b + 1) + c \quad \text{distribution, identity}
\]

\[
y = a(1) + c \quad \text{law of 1’s}
\]

\[
y = a + c \quad \text{identity}
\]
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
Types of Circuits

- *Synchronous Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL) circuits
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
    - Last lecture was combinational logic
  - Sequential Logic (SL)
    - Circuits that “remember” or store information
    - aka “State Elements”
    - E.g., memories and registers (Registers)
    - Rest of today’s lecture is sequential logic
Design Hierarchy
Uses for State Elements

• Place to store values for later re-use:
  – Register files (like $1-$31 on the RISC-V)
  – Memory (caches and main memory)

• *Help control flow of information between combinational logic blocks*
  – State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage
Program Counter: First Design

- **Program Counter “PC”**
  - Instruction address
  - Next PC: add 4 to current value
  - Let’s try ...

![Diagram of PC and PC + 4]

- Something is not quite right:
  - PC and PC+4 simultaneously on same wire???
Fix

- Memory element breaks the feedback loop
- How does it work?
RS Latch

Q and Qbar keep state, despite S deasserted → memory

10/5/17
(Positive) Edge Triggered Flip-Flop

RS latch operates on input levels
D FF operates on (clock) edges
Program Counter: Improved Design

![Diagram showing the program counter and its improved design. The diagram includes a clock signal (clk), current program counter (PC), next program counter (Next PC), and the addition process.]
Clock

$f_s = \frac{1}{t_{\text{cycle}}}$

<table>
<thead>
<tr>
<th>Clock frequency $f_s$</th>
<th>Period $t_{\text{cycle}} = \frac{1}{f_s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 2.5 GHz</td>
<td>400 ps</td>
</tr>
<tr>
<td>Drum/heartbeat, 1Hz</td>
<td>1 s</td>
</tr>
</tbody>
</table>
Maximum Clock Speed

• How fast can the PC circuit operate?
  – I.e. what is the maximum clock frequency?

• Let’s look at the timing requirements of each part
  – Let’s start with the flip-flop
Flip-Flop Timing

![Diagram of a flip-flop with timing labels: setup, hold, and clk-to-Q.]
Flip-Flop Timing

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{\text{setup}}$</td>
<td>time during which $D$ must not change before positive clock edge</td>
</tr>
<tr>
<td>$t_{\text{hold}}$</td>
<td>time during which $D$ must not change after positive clock edge</td>
</tr>
<tr>
<td>$t_{\text{clk-to-Q}}$</td>
<td>delay after positive clock edge after which $D$ appears at Q output</td>
</tr>
</tbody>
</table>
Camera Analogy

- clk
- You
- Shutter
- Display

- hold-still
- open
- portrait
Maximum Clock Speed

Combinatorial Logic (CL)

Clock

Register (flip-flops)

Input

Optional Feedback

Output

Input

CL

Output

CLK

Input[-1]

Input

Input[+1]

Input[+2]

CL[-1]

CL

CL[+1]

Output[-1]

Output

Output[+1]

CLK

t_{CLK}

t_{setup}

10/5/17

Lecture 10: Sequential Logic
Maximum PC Clock Speed

- Minimum cycle time: \[ t_{\text{min}} = t_{\text{setup}} + t_{\text{add}} + t_{\text{clk-to-Q}} \]
- Maximum clock rate: \[ f_{\text{clk-max}} = \frac{1}{t_{\text{min}}} \]

Example: \[ t_{\text{setup}} \geq 15\text{ps}, \ t_{\text{add}} = 75\text{ps}, \ t_{\text{clk-to-Q}} \geq 10\text{ps} \]
\[ t_{\text{min}} \geq 100\text{ps} \]
\[ f_{\text{clk-max}} \leq 10\text{GHz} \]
Design Hierarchy

- System
  - Datapath
    - Code registers
    - Multiplexer
    - Comparator
  - Control
    - State registers
    - Combinational logic
  - Register
    - Logic
  - Switching networks
Example: Serial Communication

- Wifi sends data “1 bit at a time”
- How do we know where a byte “starts”?
- Send “preamble ...”
  - E.g. 3 one’s in a row
FSM* to Detect 3 One’s

* FSM = Finite State Machine

FSM to detect the occurrence of 3 consecutive 1’s in the input.

State transition diagram:
FSM* to Detect 3 One’s

* FSM = Finite State Machine

FSM to detect the occurrence of 3 consecutive 1’s in the input.

State transition diagram:

State transitions are controlled by the clock: on each clock cycle the FSM
• checks the inputs,
• transitions to a new state, and
• produces a new output ...
FSM Combinatorial Logic

State Encoding

<table>
<thead>
<tr>
<th>State</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Truth Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FSM Combinatorial Logic

State Encoding

<table>
<thead>
<tr>
<th>State</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>unused</td>
<td>11</td>
</tr>
</tbody>
</table>

Truth Table

<table>
<thead>
<tr>
<th>PS</th>
<th>Input</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td>00</td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Implementation of FSM
Finite State Machines (FSM) - Summary

• Describe computation over time

• Represent FSM with “state transition diagram”

• Start at given state and input, follow some edge to next (or same) state

• With combinational logic and registers, any FSM can be implemented in hardware
Administrivia

• Homework 2 set to be released Friday
  – It will be multiple-choice and free response style (similar to HW0)
  – Due in next Friday (in 1 week)

• Project 1 Part 1 will also be released Friday
  – Due Monday, Oct 9

• Homework 1 regrades:
  – Google form will be released tonight
  – Instructions and guidelines will be on the piazza post
LIFE IN HELL

IMPORTANT QUESTIONS ABOUT MONSTERS
BY WILL AND ABE

IS MOTHER A GOOD MoTH OR A BAD MoTH?

HOW DO YOU KILL A MUMMY?

WHY DO CYCLOPS HAVE ONLY ONE EYE?

WHY DO PEOPLE GO TO DRACULA'S HOUSE??

BECAUSE THEY DIDN'T KNOW HE WAS A VAMPIRE.

BUT HIS NAME IS DRACULA!!

HAVE YOU EVER HEARD OF A MOVIE CALLED
"KING KONG VERSUS DRACULA"??

WHY DON'T VAMPIRES GET KIDS?? BECAUSE KIDS AREN'T TASTY??

WHY DOES GODZILLA ONLY ATTACK JAPAN JAPAN JAPAN??

HAVE YOU EVER HEARD OF A MOVIE CALLED
"BRIDE OF THE CREATURE OR THE BLACK LAGOON"?

WHY CAN'T ZOMBIES RUN??

WHY DID THE BRIDE OF FRANKENSTEIN MARRY FRANKENSTEIN?

IS THERE A WOLF-WOMAN??

WHY DO SKELETONS DANCE??

©1994 BY MATT GROENING

10/5/17
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
Design Hierarchy

system

datapath

code registers

multiplexer

comparator

state registers

combinational logic

logic

register

switching networks
Conceptual RISC-V Datapath
Data Multiplexer
(e.g., 2-to-1 x n-bit-wide)
N Instances of 1-bit-Wide Mux

\[
c = \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\
= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\
= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\
= \overline{s}(a(1) + s((1)b) \\
= \overline{s}a + sb
\]

<table>
<thead>
<tr>
<th>s</th>
<th>ab</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>
How Do We Build a 1-bit-Wide Mux (in Logisim)?
4-to-1 Multiplexer

How many rows in TT?

\[ e = \overline{s_1 s_0}a + \overline{s_1 s_0}b + s_1 \overline{s_0}c + s_1 s_0 d \]
Alternative Hierarchical Approach (in Logisim)
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

\[
\text{ALU} \quad \begin{align*}
\text{when } S=00, \ R &= A + B \\
\text{when } S=01, \ R &= A - B \\
\text{when } S=10, \ R &= A \text{ AND } B \\
\text{when } S=11, \ R &= A \text{ OR } B
\end{align*}
\]
Simple ALU
Adder/Subtractor: One-bit adder Least Significant Bit

\[
\begin{array}{ccc}
 a_3 & a_2 & a_1 & a_0 \\
 b_3 & b_2 & b_1 & b_0 \\
\hline
 s_3 & s_2 & s_1 & s_0
\end{array}
\]

\[
\begin{array}{cccc}
 a_0 & b_0 & s_0 & c_1 \\
 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 1
\end{array}
\]

\[s_0 = a_0 \text{ XOR } b_0\]
\[c_1 = a_0 \text{ AND } b_0\]
Adder/Subtractor: One-bit adder (1/2) ...

\[
\begin{array}{cccc|cc}
 a_i & b_i & c_i & s_i & c_{i+1} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[s_i = \text{XOR}(a_i, b_i, c_i)\]
\[c_{i+1} = \text{XOR}(s_i, c_i)\]
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 $\Rightarrow$ 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?
School is Hell

Lesson 5: Does School Last Forever?

1st Grade:
We're all going to put our heads on our desks until the north little boy who decorated in the usual comes forward.

2nd Grade:
You say your grandmother is from Russia? I'm very sorry to hear that, you know, they don't believe in God in Russia, and Christmas is against the law there.

3rd Grade:
The class has been divided into three reading groups. The gold group and the silver group will stay here. The brown group will go to a special room in the basement.

4th Grade:
And you'll stay in the garbage can until you can be a good citizen.

5th Grade:
Draw a small circle on the blackboard.

6th Grade:
Now stick your nose on it while the rest of the class goes out for recess.

7th Grade:
How do you like writing? I must remember to be cheerful and obedient. 500 times. Now write twice. I slowly rip it up before your eyes.

8th Grade:
OK, sex education. Um, is there anyone who doesn't know? Good. Next: dental hygiene.

9th Grade:
Perhaps this planning grade will steer you in the right direction.

10th Grade:
So you think marching in the hippie-trippy peace demonstration is more important than school. Eh, then I guess this "F" won't matter much.

11th Grade:
I'm afraid that insolent remark about our president will go on your permanent record, young man.

12th Grade:
If you think you can get through life drawing silly cartoons, you've got another think coming.

1st Year College:
Mr. Grozaje, I'm getting bad vibes from you. The best of the class grades (what is going on)?... why can't you?

2nd Year:
The sooner you all face up to the fact that you are lazy, untalented losers, unfit to kiss the feet of a genius like Friedrich Nietzsche, the better off you'll be.

3rd Year:
(Listen), I'll give you full credit as long as you don't come around and bother me anymore.

Final Year:
You do what you do tolerably well, Mr. Grozing. Now you must rise yourself. Is it worth doing?
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
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 RISC-V Lite Subset

- **ADD and SUB**
  - `add rd, rs1, rs2`
  - `sub rd, rs1, rs2`

- **OR Immediate:**
  - `ori rd, rs1, imm12`

- **LOAD and STORE Word**
  - `lw rd, rs1, imm12`
  - `sw rs2, rs1, imm12`

- **BRANCH:**
  - `beq rs1, rs2, imm12`
Register Transfer Language (RTL)

- RTL gives the **meaning** of the instructions
  
  \[
  \{ \text{op, rs1, rs2, rd, funct3} \} \leftarrow \text{MEM[ PC ]}
  \]
  
  \[
  \{ \text{op, rs1, rs2, Imm12} \} \leftarrow \text{MEM[ PC ]}
  \]

- All start by fetching the instruction

<table>
<thead>
<tr>
<th>Inst</th>
<th>Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[rd] ← R[rs1] + R[rs2]; PC ← PC + 4</td>
</tr>
<tr>
<td>SUB</td>
<td>R[rd] ← R[rs1] − R[rs2]; PC ← PC + 4</td>
</tr>
<tr>
<td>ORI</td>
<td>R[rd] ← R[rs1]</td>
</tr>
<tr>
<td>LW</td>
<td>R[rd] ← MEM[R[rs1] + sign_ext(Imm12)]; PC ← PC + 4</td>
</tr>
<tr>
<td>SW</td>
<td>MEM[R[rs1] + sign_ext(Imm12)] ← R[rs2]; PC ← PC + 4</td>
</tr>
</tbody>
</table>
| BEQ  | if ( R[rs] == R[rt] )  
  then PC ← PC + (sign_ext(Imm12) || 0)
  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 rs1
  - Read rs2
  - Write rd
- PC
- Sign Extender
- Add/Sub/OR unit for operation on register(s) or sign extended immediate
- Add 4 (or maybe sign extended immediate) to PC
- Compare if registers equal?
Generic Steps of Datapath

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Register Write
Step 2: Components of the Datapath

- Combinational Elements
- State Elements + Clocking Methodology
- Building Blocks
ALU Needs for RISC-V Lite + Rest of RISC-V

• Addition, subtraction, logical OR, ==:
  
  ADD  \( R[rd] = R[rs1] + R[rs2]; \) ...
  
  SUB  \( R[rd] = R[rs1] - R[rs2]; \) ...
  
  ORI  \( R[rt] = R[rs1] \mid \text{sign\_ext(Imm12)} \) ...
  
  BEQ  \( \text{if (} R[rs] == R[rt] \text{)} \) ...

• Test to see if output == 0 for any ALU operation gives == test. How?

• P&H Ch. 4 also adds AND, 64 bit LD/SD instructions

• ALU from Appendix A, Section A.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 $\Rightarrow$ 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 $\Rightarrow$ 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

• $R[rd] = R[rs] \text{ op } R[rt]$ (addu rd, rs, rt)
  – Ra, Rb, and Rw come from instruction’s Rs1, Rs2, and Rd fields

– ALUctr and RegWr: control logic after decoding the instruction

• ... Already defined the register file & ALU
Agenda

• Boolean Algebra
• Timing and State Machines
• Datapath Elements: Mux + ALU
• RISC-V Lite Datapath
• And, in Conclusion, ...
And in Conclusion, ...

• State Machines
  – Finite State Machines: made from *Stateless* combinational logic and *Stateful* “Memory” Logic (aka Registers)
  – Clocks synchronize D-FF change (Setup and Hold times important!)
  – Pipeline long-delay CL for faster clock cycle—Split the *critical path*
• Use muxes to select among inputs
  – 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