You are Here!

- 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., all 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages
  Hardware
  Software

Logic Circuit Description
(Circuit Schematic Diagrams)

Design Hierarchy

System

Control

Datapath

Memory

Input/Output

Register Units

Functional Units

Cache Memory

Register

10/5/17

Fall 2017 -- Lecture #10

• Levels of Representation/Interpretation
  High Level Language
  Assembly Language
  Machine Language
  Hardware Architecture
  Logic Circuit Description

• 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, ...

10/5/17

Fall 2017 -- Lecture #10
**Boolean Algebra**

- Use plus for OR
  - "logical sum"
- Use product for AND (a\(\cdot\)b or implied via ab)
  - "logical product"
- "Hat" to mean complement (NOT)
- Thus
  \[
  \begin{align*}
  ab + a + \overline{a} &= a \land b + a \land \overline{a} \\
  &= (a \land b) \lor a \lor (\neg c)
  \end{align*}
  \]

**Laws of Boolean Algebra**

<table>
<thead>
<tr>
<th>Identity</th>
<th>Complementarity</th>
<th>Laws of 0's and 1's</th>
<th>Idempotent Laws</th>
<th>Commutativity</th>
<th>Associativity</th>
<th>Distribution</th>
<th>Uniting Theorem</th>
<th>Uniting Theorem x 2</th>
<th>DeMorgan's Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \lor \overline{x} = 1)</td>
<td>(x + \overline{x} = 1)</td>
<td>(x = 0 \implies \overline{x} = 1)</td>
<td>(x = 0)</td>
<td>(x = 1)</td>
<td>Idempotent Laws</td>
<td>Complementarity</td>
<td>Laws of 0's and 1's</td>
<td>Idempotent Laws</td>
<td>Complementarity</td>
</tr>
<tr>
<td>(x \land \overline{x} = 0)</td>
<td>(x \cdot \overline{x} = 0)</td>
<td>(x = 1 \implies \overline{x} = 0)</td>
<td>Idempotent Laws</td>
<td>Complementarity</td>
<td>Laws of 0's and 1's</td>
<td>Idempotent Laws</td>
<td>Complementarity</td>
<td>Laws of 0's and 1's</td>
<td>Idempotent Laws</td>
</tr>
<tr>
<td>(x \lor x = x)</td>
<td>(x \cdot x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
</tr>
<tr>
<td>(x \lor x = x)</td>
<td>(x \cdot x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
<td>(x = x)</td>
</tr>
</tbody>
</table>

**Boolean Algebraic Simplification Example**

\[
\begin{align*}
  y &= ab + a + c \\
  &= a(b + 1) + c \\
  &= a(1) + c \\
  &= a + c
\end{align*}
\]

**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 (ALU)
    - Last lecture was combinational logic
  - Sequential Logic (SL)
    - Circuits that “remember” or store information
    - E.g., memories and registers (Registers)
    - Rest of today’s lecture is sequential logic

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 ...

  - 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

- RS Latch keeps state despite 5-de asserted or memory
(Positive) Edge Triggered Flip-Flop

**RS latch operates on input levels**

**D FF operates on (clock) edges**

Clock “clk”

**Clock**

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

<table>
<thead>
<tr>
<th>Clock frequency $$f_s$$</th>
<th>Period $$t_{cycle}$$ = $$\frac{1}{f_s}$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 2.5 GHz</td>
<td>400 ps</td>
</tr>
<tr>
<td>Drum/heartbeat, 3Hz</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**

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$t_{setup}$$</td>
<td>time during which D must not change before positive clock edge</td>
</tr>
<tr>
<td>$$t_{hold}$$</td>
<td>time during which D must not change after positive clock edge</td>
</tr>
<tr>
<td>$$t_{delay}$$</td>
<td>delay after positive clock edge after which D appears at Q output</td>
</tr>
</tbody>
</table>
Maximum Clock Speed

- Minimum cycle time: $t_{\text{min}} = t_{\text{setup}} + t_{\text{hold}} + t_{\text{clk-to-Q}}$
- Maximum clock rate: $f_{\text{clk}} = \frac{1}{t_{\text{min}}}$

Example: $t_{\text{setup}} = 15\, \text{ps}$, $t_{\text{hold}} = 75\, \text{ps}$, $t_{\text{clk-to-Q}} = 10\, \text{ps}$
$t_{\text{min}} = 100\, \text{ps}$
$f_{\text{clk}} = 10\, \text{GHz}$

Design Hierarchy

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 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 Code</th>
<th>State Encoding</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>X</th>
<th>Input</th>
<th>Y</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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
Agenda

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

Design Hierarchy

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

N Instances of 1-bit-Wide Mux

\[
\begin{array}{|c|c|c|}
\hline
s & ab & c \\
\hline
0 & 00 & 0 \\
0 & 01 & 0 \\
0 & 10 & 1 \\
0 & 11 & 1 \\
1 & 00 & 0 \\
1 & 01 & 1 \\
1 & 10 & 0 \\
1 & 11 & 1 \\
\hline
\end{array}
\]

\[
c = \overline{s}a\overline{b} + s\overline{a}b + \overline{s}ab + sab \\
= \overline{s}(ab + ab) + s(\overline{a}b + ab) \\
= \overline{s}(a(b + b)) + s((\overline{a} + a)b) \\
= \overline{s}(a(1)) + s(1)b \\
= \overline{s}a + sb 
\]
How Do We Build a 1-bit-Wide Mux (in Logisim)?

```
How many rows in TT?
```

4-to-1 Multiplexer

```
e = s_1 s_0 a + s_1 s_0 b + s_1 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

```
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

```
a_3 a_2 a_1 a_0 + b_3 b_2 b_1 b_0
```

```
\begin{array}{cccc|cccc}

    a_3 & a_2 & a_1 & a_0 & b_3 & b_2 & b_1 & b_0 \\
\hline
0 0 & 0 & 0 & 0 & 0 1 & 1 & 0 & 0 \\
0 1 & 1 & 0 & 0 & 1 0 & 1 & 0 \\
1 0 & 0 & 0 & 1 & 1 0 & 1 & 0 \\
1 1 & 0 & 0 & 1 & 1 0 & 1 & 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) ...

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

N x 1-bit Adders \(\Rightarrow\) 1 N-bit Adder

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

- Boolean Algebra
- Timing and State Machines
- Datapath Elements: Mux + ALU
- RISC-V Lite Datapath
- 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

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 rd, rs1, rs2, imm12
  - bne rd, rs1, rs2, imm12

Register Transfer Language (RTL)

- RTL gives the meaning of the instructions

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, ==:
  \[ \text{ADD} R[rd] = R[rs1] + R[rs2]; \]  
  \[ \text{SUB} R[rd] = R[rs1] - R[rs2]; \]  
  \[ \text{OR} R[rt] = R[rs1] \mid \text{sign_ext}(\text{Imm12}); \]  
  \[ \text{BEQ} \text{if } ( R[rs] == R[rt] ); \]

- 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 \(\Rightarrow\) Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the instruction: mem(PC)
  - Update the program counter:
    - Sequential Code: \(\text{PC} = \text{PC} + 4\)
    - Branch and Jump: \(\text{PC} = \text{“something else”}\)
Step 3: Add & Subtract
- \( R[rd] = R[ra] \text{ op } R[rt] \) (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 CI for faster clock cycle—
    Split the critical path
- Use muxes to select among inputs
  - 1 input bits selects \( 2^n \) inputs
  - Each input can be \( n \)-bit 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 affects the register transfer
5. Formulate control logic
   - Formulate Logic Equations
   - Design Circuits