Recap: Complete RV32I ISA

State Required by RV32I ISA
Each instruction reads and updates this state during execution:
- Registers (x0..x31)
  - Register file (or regfile) Reg holds 32 registers x 32 bits/register: Reg[0]..Reg[31]
  - First register read specified by rs1 field in instruction
  - Second register read specified by rs2 field in instruction
  - Write register (destination) specified by rd field in instruction
- x0 is always 0 (writes to Reg[0] are ignored)
- Program Counter (PC)
  - Holds address of current instruction
- Memory (MEM)
  - Holds both instructions & data, in one 32-bit byte-addressed memory space
  - We’ll use separate memories for instructions (IMEM) and data (DMEM)
  - Load/store instructions access data memory

One-Instruction-Per-Cycle RISC-V Machine
- On every tick of the clock, the computer executes one instruction
- Current state outputs drive the inputs to the combinational logic, whose outputs settles at the values of the state before the next clock edge
- At the rising clock edge, all the state elements are updated with the combinational logic outputs, and execution moves to the next clock cycle

Implementing the add instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>rs2</th>
<th>rs1</th>
<th>080</th>
<th>rd</th>
<th>011011</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>add rd, rs1, rs2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Instruction makes two changes to machine’s state:
- Reg[rd] = Reg[rs1] + Reg[rs2]
- PC = PC + 4
Implementing the **sub** instruction

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>010000</td>
<td>m2</td>
<td>m1</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
</tr>
</tbody>
</table>

- Almost the same as add, except now have to subtract operands instead of adding them
- \textit{inst[30]} selects between add and subtract

Implementing other R-Format instructions

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>010000</td>
<td>m2</td>
<td>m1</td>
<td>001</td>
<td>rd</td>
<td>0110011</td>
<td>SLL</td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>010</td>
<td>rd</td>
<td>0110011</td>
<td>SRS</td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>011</td>
<td>rd</td>
<td>0110011</td>
<td>SLT</td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>100</td>
<td>rd</td>
<td>0110011</td>
<td>SNE</td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>101</td>
<td>rd</td>
<td>0110011</td>
<td>SRL</td>
<td></td>
</tr>
<tr>
<td>010000</td>
<td>m2</td>
<td>m1</td>
<td>110</td>
<td>rd</td>
<td>0110011</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>m2</td>
<td>m1</td>
<td>111</td>
<td>rd</td>
<td>0110011</td>
<td>AND</td>
<td></td>
</tr>
</tbody>
</table>

- All implemented by decoding \textit{funct3} and \textit{funct7} fields and selecting appropriate ALU function

Implementing the **addi** instruction

- RISC-V Assembly Instruction:

\begin{verbatim}
addi x15, x1, -50
\end{verbatim}

<table>
<thead>
<tr>
<th>imm(16)</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

\begin{verbatim}
111111001110 | 00001 | 000 | 00111 | 0010011 |
\end{verbatim}

imm=-50 rs1=1 ADD rd=15 OP-Imm
### I-Format immediates

<table>
<thead>
<tr>
<th>31</th>
<th>30:19</th>
<th>18:12</th>
<th>11:7</th>
<th>6:0</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>imm(11:0)</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-----</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inst[7:0]</td>
<td>imm(10:0)</td>
<td></td>
<td>immSel = I</td>
</tr>
</tbody>
</table>

- High 12 bits of instruction (inst[11:0]) copied to low 12 bits of immediate (imm[11:0]).
- Immediate is sign-extended by copying value of inst[11] to fill the upper 20 bits of the immediate value (imm[31:12]).

### Adding addi to datapath

- Also works for all other I-format arithmetic instruction (sll, sltu, andi, ori, xor, sll, srl, sra) just by changing ALUSel.

---

**TSMC Announces 3nm CMOS Fab**

Latest Apple iPhone 8, iPhone X use TSMC’s 10nm process technology. 3nm technology should allow 10x more stuff on the same sized chip (10/3)^2.

The new manufacturing plant will occupy nearly 200 acres and cost around $15B, open in around 5 years (~2022).

Currently, fabs use 193nm light to expose masks. For 3nm, some layers will use Extreme Ultra-Violet (13.5nm).
Implementing Load Word instruction

- RISC-V Assembly Instruction:
  \[ \text{lw} \ x14, \ 8(x2) \]

\[
\begin{array}{cccccc}
31 & 29 & 28 & 27 & 26 & 25 \\
imm[11:0] & rs1 & funct3 & rd & 7 & 6 \\
12 & 5 & 3 & 5 & 7 & 8 \\
\end{array}
\]

\[
\begin{array}{cccccc}
000000001000 & 00010 & 010 & 01110 & 0000011 \\
imm=+8 & rs1=2 & LW & rd=14 & LOAD \\
\end{array}
\]

Adding \text{addi} to datapath

- RISC-V Assembly Instruction:
  \[ \text{sw} \ x14, \ 8(x2) \]

\[
\begin{array}{cccccccccc}
31 & 29 & 28 & 27 & 26 & 25 & 24 & 23 & 22 & 21 \\
12 & 5 & 3 & 5 & 7 & 8 & 5 & 6 & 7 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0000000 & 01110 & 00010 & 010 & 01000 & 0100011 \\
\text{sw} = 0 & rs2=14 & rs1=2 & SW & \text{offset}[4:0] & STORE \\
& & & & & =8 \\
\end{array}
\]

All RV32 Load Instructions

- Supporting the narrower loads requires additional circuits to extract the correct byte/halfword from the value loaded from memory, and sign- or zero-extend the result to 32 bits before writing back to register file.
Adding \textbf{lw} to datapath

Adding \textbf{sw} to datapath

\textbf{I-Format immediates}

\textbf{I & S Immediate Generator}

\textbf{Implementing Branches}
Adding **sw** to datapath

![Adding sw to datapath diagram]

Adding branches to datapath

![Adding branches to datapath diagram]

Branch Comparator

- BrEq = 1, if A=B
- BrLT = 1, if A < B
- BrUn = 1 selects unsigned comparison for BrLT, 0=signed
  - BGE branch: A >= B, if !(A<B)

Administrivia (1/2)

- Midterm 1 has been graded!
- Regrade Requests will open tonight
  - Due next Tuesday (in one week)
  - Piazza will explain the instructions

Administrivia (2/2)

- Project 1 has been released
  - Part 1 is due next Monday
  - Project Party in Cory 293 on Wednesday 7-9pm (possibly later if needed)
- Homework 2 is due this Friday at 11:59pm
  - Will help to do this before the project!
- No Guerrilla Session this week—will start up again next Tuesday
Multiply Branch Immediates by Shift?

- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes.
- Standard approach: treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches.

```

\[
\begin{array}{c|c|c|c|c|c}
| sign-extension | \text{imm} | \text{imm} | \text{imm} | \text{imm} |
\end{array}
\]

- B-immediate (shift left by 1)

Each instruction immediate bit can appear in one of two places in output immediate value – so need one 2-way mux per bit.

RISC-V Branch Immediates

- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes.
- RISC-V approach: keep 11 immediate bits in fixed position in output value, and rotate LSB of S-format to be bit 12 of B-format.

```

\[
\begin{array}{c|c|c|c|c|c|c}
| sign | \text{imm} | \text{imm} | \text{imm} | \text{imm} |
\end{array}
\]

- S-immediate

```

\[
\begin{array}{c|c|c|c|c|c|c}
| sign | \text{imm} | \text{imm} | \text{imm} | \text{imm} |
\end{array}
\]

- B-immediate (shift left by 1)

Only one bit changes position between S and B, so only need a single-bit 2-way mux.

Implementing JALR Instruction (I-Format)

- JALR rd, rs, immediate
  - Writes PC+4 to Reg[rd] (return address)
  - Sets PC = Reg[rs1] + immediate
  - Uses same immediates as arithmetic and loads

```

\[
\begin{array}{c|c|c|c|c|c|c|c}
| \text{imm} | \text{imm} | \text{imm} | \text{imm} |
\end{array}
\]

- offset[12:0]

```

- pc+4

Adding branches to datapath

Only bit 7 of instruction changes role in immediate between S and B.

Upper bits sign-extended from inst[31] always.
Implementing jal Instruction

- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within ±2^{15} locations, 2 bytes apart
  - ±2^{15} 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost
And in Conclusion, ...

- Universal datapath
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions

- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases

- Controller specifies how to execute instructions
  - what new instructions can be added with just most control?