EECS151/251A : Introduction to Digital Design and ICs

Lecture 8 – Combinational Logic, RISC-V

Bora Nikolić


IEEE Working Group P3109 releases interim report on 8-bit binary floating-point formats.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>binary8pp</th>
<th>binary8p5</th>
<th>binary8p4</th>
<th>binary8p3</th>
<th>binary16</th>
<th>binary32</th>
<th>binary64</th>
</tr>
</thead>
<tbody>
<tr>
<td>k, storage width in bits</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>p, precision in bits</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
<td>11</td>
<td>24</td>
<td>53</td>
</tr>
<tr>
<td>emax, max exponent</td>
<td>28−p+1−1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>15</td>
<td>127</td>
<td>1023</td>
</tr>
<tr>
<td>w, exponent field width</td>
<td>8−p</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>bias, E−e</td>
<td>emax+1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>15</td>
<td>127</td>
<td>1023</td>
</tr>
<tr>
<td>sign bit</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t, significand field width</td>
<td>p−1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>23</td>
<td>52</td>
</tr>
</tbody>
</table>

https://github.com/P3109/Public/tree/main/Shared Reports
Review

• **Boolean algebra**
  - Deal with variables that are either True or False
  - Map naturally to hardware logic gates
  - Use theorems of Boolean algebra and Karnaugh maps to simplify equations

• **RISC-V ISA**
  - Open, with increasing adoption

• **RISC-V processor**
  - A large state machine
  - Datapath + control
R-Format Instructions: Datapath
R-Format Example

- RISC-V Assembly Instruction:
  \[ \text{add} \ x_{18}, x_{19}, x_{10} \]

<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

| 00000000 | 01010 | 10011 | 000 | 10010 | 0110011 |

\[ \text{add} \quad \text{rs2}=10 \quad \text{rs1}=19 \quad \text{add} \quad \text{rd}=18 \quad \text{Reg-Reg \ OP} \]
Implementing the **add** instruction

<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
<td>00</td>
<td>000</td>
<td>01</td>
<td>01011</td>
</tr>
</tbody>
</table>

**add**  

rs2  

rs1  

add  

rd  

Reg-Reg  

OP

```
add rd, rs1, rs2
```

- Instruction makes two changes to machine’s state:
  - \( \text{Reg}[\text{rd}] = \text{Reg}[\text{rs1}] + \text{Reg}[\text{rs2}] \)
  - \( \text{PC} = \text{PC} + 4 \)
Datapath for **add**

**PC = PC + 4**  
**Reg[rd] = Reg[rs1] + Reg[rs2]**

Control logic

RegWriteEnable (RegWEn) = 1

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>20</th>
<th>19</th>
<th>15</th>
<th>14</th>
<th>12</th>
<th>11</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>rs2</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**add**  

5  

5  

add  

5  

Reg-Reg  

OP
Timing Diagram for add

Clock
PC
PC+4
inst[31:0]
Reg[rs1]
Reg[rs2]
alu
Reg[1]

1000
1004
1008

add x1, x2, x3
add x6, x7, x9

Reg[2]
Reg[3]
Reg[7]
Reg[9]
Implementing the `sub` instruction

- Almost the same as `add`, except now have to subtract operands instead of adding them
- `inst[30]` selects between `add` and `subtract`
Datapath for add/sub

Control logic

RegWEn
(1=Write, 0=NoWrite)

ALUSel
(add=0/sub=1)
Implementing other R-Format instructions

<table>
<thead>
<tr>
<th>Format</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0100000</td>
<td>rs2</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>001</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>010</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>011</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>100</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>101</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0100000</td>
<td>rs2</td>
<td>rs1</td>
<td>101</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>110</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>0000000</td>
<td>rs2</td>
<td>rs1</td>
<td>111</td>
<td>rd</td>
<td>0110011</td>
</tr>
</tbody>
</table>

- All implemented by decoding funct3 and funct7 fields and selecting appropriate ALU function
Administrivia

• **Midterm 1**: Thursday 10/5, 7-8:30pm
• Additional office hours (Thomas and William) on Wednesday and Thursday for homework help
• **Lab 3 finished**
  • If you couldn’t attend the lab, please get checked off in office hours this week
• **Lab 4 is this week**
  • Cycle wraps up on Monday
  • Class ‘ambassadors’
  • Lab 5 starts next week
• **Homework 3 due on Friday**
I-Format Instructions: Datapath
Instruction Encoding

- Instructions are encoded to simplify logic
  - sub and sra differ in Inst[30] from add and srl
- RV64I widens registers (XLEN=64)
- Additional instructions manipulate 32-bit values, identified by a suffix W
  - ADDW, SUBW
  - RV64I opcode field for ‘W’ instructions is 0111011 (0110011 for RV32I)

<table>
<thead>
<tr>
<th>00000000</th>
<th>rs2</th>
<th>rs1</th>
<th>000</th>
<th>rd</th>
<th>0110011</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>rs2</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0111011</td>
</tr>
</tbody>
</table>

addw

add

64b

32b
**I-Format Instruction Layout**

<table>
<thead>
<tr>
<th>31</th>
<th>25 24</th>
<th>20 19</th>
<th>15 14</th>
<th>12 11</th>
<th>7 6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>funct7</td>
<td>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>7 12 5</td>
<td>5 3</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Only one field is different from R-format, rs2 and funct7 replaced by 12-bit signed immediate, \( \text{imm}[11:0] \)
- Remaining fields (rs1, funct3, rd, opcode) same as before
- \( \text{imm}[11:0] \) can hold values in range \([-2048_{\text{ten}}, +2047_{\text{ten}}]\)
- Immediate is always sign-extended to 32-bits before use in an arithmetic operation
- Other instructions handle immediates > 12 bits
### All RV32 I-format Arithmetic Instructions

<table>
<thead>
<tr>
<th>imm[11:0]</th>
<th>rs1</th>
<th>000</th>
<th>rd</th>
<th>0010011</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>010</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>011</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>100</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>110</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>111</td>
<td>rd</td>
<td>0010011</td>
</tr>
</tbody>
</table>

The same Inst[30] immediate bit is used to distinguish “shift right logical” (SRLI) from “shift right arithmetic” (SRAI).

### Shift-by-immediate Instructions

“Shift-by-immediate” instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions).
Implementing I-Format - **addi** instruction

- **RISC-V Assembly Instruction** – add immediate:

  \[
  \text{addi } x15, x1, -50
  \]

<table>
<thead>
<tr>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>imm([11:0])</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

| 111111001110 | 000001 | 000 | 01111 | 0010011 |

- \(\text{imm} = -50\)  
- \(\text{rs1} = 1\)  
- \(\text{add}\)  
- \(\text{rd} = 15\)  
- **OP-Imm**
Datapath for **add/sub**

---

**IMEM**

- **pc + 4**
- **clk**
- **+4**
- **Add**
- **Inst[24:20]**
- **Inst[19:15]**
- **Inst[11:7]**
- **Inst[24:20]**

**Addr**

- **AddrA**
- **AddrB**

**Data**

- **DataA**
- **DataB**

**Reg**

- **Reg[rs1]**
- **Reg[rs2]**

**ALU**

- **alu**

**Control logic**

- **Inst[31:0]**
- **RegWEn** (1=Write, 0=NoWrite)
- **ALUSel** (add=0/sub=1)

**Immediate should be here**
Adding **addi** to Datapath

### Control logic

- **Inst[31:0]**
- **RegWEn**
  - (1=Write, 0=NoWrite)
- **BSel**
  - (rs2=0/Imm=1)
- **ALUSel**
  - (add=0/sub=1)

### IMEM

- **pc+4**
- **addr**
- **inst**

### ALU

- **Reg[rs1]**
- **Reg[rs2]**
- **DataD**
- **AddrD**
- **DataA**
- **AddrA**
- **DataB**
- **AddrB**
- **DataD**
- **AddrD**

### Control signals

- **RegWEn**
- **BSel**
- **ALUSel**

### Adder

- **Add**

### Bus connections

- **Inst[24:20]**
- **AddrA**
- **DataA**
- **AddrB**
- **DataB**
- **AddrD**
- **DataD**
- **Imm[31:0]**
Adding `addi` to Datapath

```
Adding addi to Datapath

PC + 4
```

```
+4
Add

IMEM

addr
inst

Inst[11:7]
Inst[19:15]
Inst[24:20]

AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]

Inst[31:20]

Imm. Gen

Imm[31:0]

alu

Reg[rs1]
Reg[rs2]

ImmSel = 1
RegWEn = 1
BSel (rs2=0/Imm=1)
ALUSel = add

Control logic

Inst[31:0]

ImmSel
RegWEn
BSel
ALUSel

EECS151 L09 RISC-V
I-Format immediates

- **inst[31] -**
  - High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])
  - Immediate is sign-extended by copying value of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])
  - Sign extension often in critical path

- **ImmSel=I**

---

- **imm[11:0]**
- **rs1**
- **funct3**
- **rd**
- **opcode**

---

- **inst[31:0]**

---

- **inst[30:20]**

---

- **imm[31:0]**
R+I Datapath

Works for all other I-format arithmetic instructions (slti, sltiu, andi, ori, xori, slli, srli, srai) just by changing ALUSel
Add `lw` to Datapath

- **RISC-V Assembly Instruction (I-type):**

  \[
  lw \ x14, \ 8(x2)
  \]

  \[
  \begin{array}{cccccccc}
    31 & 20 & 19 & 15 & 14 & 12 & 11 & 7 & 6 & 0 \\
    \hline
    imm[11:0] & rs1 & funct3 & rd & opcode \\
    \hline
    12 & 5 & 3 & 5 & 7 \\
    offset[11:0] & base & width & dest & LOAD \\
    \hline
    31 & 20 & 19 & 15 & 14 & 12 & 11 & 7 & 6 & 0 \\
    \hline
    000000001000 & 00010 & 010 & 01110 & 0000011 \\
    \hline
    imm= +8 & rs1=2 & LW & rd=14 & LOAD \\
  \end{array}
  \]

- The 12-bit signed immediate is added to the base address in register `rs1` to form the memory address
  - This is very similar to the add-immediate operation but used to create address not to create final result

- The value loaded from memory is stored in register `rd`
Adding lw to Datapath

- **Add**
- **Inst**
- **IMEM**
- **PC**
- **DMEM**
- **alu**
- **Reg [ ]**
- **Imm[31:0]**
- **Inst[31:20]**
- **AddrB**
- **AddrD**
- **DataA**
- **DataB**
- **DataD**
- **Reg[rs1]**
- **Reg[rs2]**
- **MemRW**
- **WBSel**
- **ImmSel**
- **Wb**
- **Imm Gen**
- **Inst[31:0]**
- **RegWEn**
- **Bsel**
- **ALUSel**

Control logic:
- **Inst[31:0] = I**
- **RegWEn = 1**
- **Bsel = 1**
- **ALUSel = Add**
- **MemRW = Read**
- **WBSel = 0**

Schematic diagram showing the datapath with added lw support.
All RV32 Load Instructions

<table>
<thead>
<tr>
<th>imm[11:0]</th>
<th>rs1</th>
<th>000</th>
<th>rd</th>
<th>0000011</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>001</td>
<td>rd</td>
<td>0000011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>010</td>
<td>rd</td>
<td>0000011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>100</td>
<td>rd</td>
<td>0000011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>101</td>
<td>rd</td>
<td>0000011</td>
</tr>
</tbody>
</table>

**funct3 field encodes size and ‘signedness’ of load data**

- Supporting the narrower loads requires additional logic 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.
  - It is just a mux for load extend, similar to sign extension for immediates
S-Format Instructions: Datapath
S-Format Used for Stores

- Store needs to read two registers, rs1 for base memory address, and rs2 for data to be stored, as well immediate offset!
- Can’t have both rs2 and immediate in same place as other instructions!
- Note that stores don’t write a value to the register file, no rd!
- RISC-V design decision is move low 5 bits of immediate to where rd field was in other instructions – keep rs1/rs2 fields in same place
  - register names more critical than immediate bits in hardware design
Adding **sw** Instruction

- **sw**: Reads two registers, `rs1` for base memory address, and `rs2` for data to be stored, as well immediate offset!

```assembly
sw x14, 8(x2)
```

<table>
<thead>
<tr>
<th>Imm[11:5]</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>imm[4:0]</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

00000000 01110 00010 010 01000 0100011

**offset[11:5] rs2=14 rs1=2 SW offset[4:0] STORE =0**

combined 12-bit offset = 8
Datapath with lw

**IMEM**
- pc
- addr
- clk

**Add**
- +4

**PC**
- pc
- pc
- clk

**DataD**
- Inst[11:7]
- AddrD
- AddrA
- AddrB
- DataA
- DataB
- Reg[rs1]
- Reg[rs2]

**Imm Gen**
- Inst[31:0]
- Imm[31:0]

**ALU**
- +
- alu
- Reg [ ]
- clk
- Imm[31:0]
- +
- Imm[31:0]

**DMEM**
- addr
- DataR
- mem
- wb
- wb
- clk

**Control logic**
- Inst[31:0]
- ImmSel
- RegWEn
- BSel
- ALUSel
- MemRW
- WBSel
Adding **sw** to Datapath

Adding **sw** to Datapath
All RV32 Store Instructions

- Store byte, halfword, word

<table>
<thead>
<tr>
<th>Imm[11:5]</th>
<th>rs2</th>
<th>rs1</th>
<th>000</th>
<th>imm[4:0]</th>
<th>0100011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imm[11:5]</td>
<td>rs2</td>
<td>rs1</td>
<td>001</td>
<td>imm[4:0]</td>
<td>0100011</td>
</tr>
<tr>
<td>Imm[11:5]</td>
<td>rs2</td>
<td>rs1</td>
<td>010</td>
<td>imm[4:0]</td>
<td>0100011</td>
</tr>
</tbody>
</table>

width
I+S Immediate Generation

- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction.
- Other bits in immediate are wired to fixed positions in instruction.
RISC-V
B-Format Instructions
Datapath So Far (R-, I-, S Instruction Types)
B-Format - RISC-V Conditional Branches

• E.g., BEQ x1, x2, Label

• Branches read two registers but don’t write a register (similar to stores)

• How to encode label, i.e., where to branch to?
RISC-V Feature, n×16-bit instructions

- Extensions to RISC-V base ISA support 16-bit compressed instructions and also variable-length instructions that are multiples of 16-bits in length.

- To enable this, RISC-V scales the branch offset by 2 bytes even when there are no 16-bit instructions.

- Reduces branch reach by half and means that ½ of possible targets will be errors on RISC-V processors that only support 32-bit instructions (as used in this class).

- RISC-V conditional branches can only reach \( \pm 2^{10} \times 32\)-bit instructions on either side of PC.
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.

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

## Instruction encodings, inst[31:0]

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>25</th>
<th>24</th>
<th>20</th>
<th>19</th>
<th>15</th>
<th>14</th>
<th>12</th>
<th>11</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>funct7</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 32-bit immediates produced, imm[31:0]

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>5</th>
<th>4</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

### Upper bits sign-extended from inst[31] always

Only bit 7 of instruction changes role in immediate between S and B
beq x19, x10, offset = 16 bytes

13-bit immediate, imm[12:0], with value 16

imm[12] discarded, always zero


Implementing Branches

• B-format is mostly same as S-format, with two register sources (rs1/rs2) and a 12-bit immediate

• But now immediate represents values -4096 to +4094 in 2-byte increments

• The 12 immediate bits encode even 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)
To Add Branches

• Different change to the state:
  • \[ PC = \begin{cases} 
  PC + 4, & \text{branch not taken} \\
  PC + \text{immediate}, & \text{branch taken} 
\end{cases} \]

• Six branch instructions: BEQ, BNE, BLT, BGE, BLTU, BGEU

• Need to compute \( PC + \text{immediate} \) and to compare values of \( rs1 \) and \( rs2 \)
  • Need another add/sub unit
Adding Branches

Control logic

PCSel = taken/not taken

Inst[31:0]

ImmSel = B

RegWEn = 0

BrUn = 1

BrLT = 0

BrEq = 1

Bsel = 1

ALUSel = Add = 1

MemRW = Read

WBSel = *

(* = Don’t care)
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**
## All RISC-V Branch Instructions

|-------------|-----|-----|-----|-------------|---------|

- **BEQ**: Branch Equal
- **BNE**: Branch Not Equal
- **BLT**: Branch Less Than
- **BGE**: Branch Greater Than or Equal
- **BLTU**: Branch Less Than or Unpredictable
- **BGEU**: Branch Greater Than or Equal or Unpredictable
Summary

• RISC-V ISA
  • Reviewed R, I, S and B formats
  • Datapath + control