Supporting Loads
R+I Arithmetic/Logic Datapath

RISC-V (37)

Garcia, Nikolić
**Add lw**

- **RISC-V Assembly Instruction (I-type):** `lw x14, 8(x2)`

<table>
<thead>
<tr>
<th>31</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>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>offset[11:0]</td>
<td>base</td>
<td>width</td>
<td>dest</td>
<td>LOAD</td>
<td></td>
</tr>
</tbody>
</table>

| 000000001000 | 00010 | 010 | 01110 | 0000011 |

- imm = +8  
  - rs1 = 2  
  - lw  
  - rd = 14  
  - LOAD

- 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`
R+I Arithmetic/Logic Datapath

Control logic

Garcia, Nikolić
RISC-V (40)

Garcia, Nikolić

R+I Arithmetic/Logic Datapath

Control logic

RISC-V (40)
### 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>

- 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 + a few gates

`lb`, `lh`, `lw`, `lbu`, `lhu`
Datapath for Stores
Adding **sw** instruction

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

**sw x14, 8 (x2)**

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>01110</td>
<td>00010</td>
<td>010</td>
<td>01000</td>
<td>0100011</td>
</tr>
</tbody>
</table>
```

**offset[11:5] = 0**

**combined 12-bit offset = 8**

**STORE**
Datapath with lw

Control logic

RISC-V (44)
Adding SW to Datapath

Control logic

Inst[31:0] ImmSel =S RegWriteEnable =0 Bsel =1 ALUSel =Add MemRW =Write WB Sel =*

(don't care)

RISC-V (45)
Adding \textit{sw} to Datapath

Control logic

\hspace{1cm}

RISC-V (46)
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
All RV32 Store Instructions

- Store byte writes the low byte to memory
- Store halfword writes the lower two bytes to memory

<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>sb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

width
Implementing Branches
B-format is mostly same as S-Format, with two register sources (rs1/rs2) and a 12-bit immediate imm[12:1]

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)
Datapath So Far

Control logic

PC
pc+4

IMEM

Addr

Inst

DataD

AddrD

Reg

Inst[11:7]

Inst[19:15]

Inst[24:20]

Imm[31:0]

BSel

ImmSel

RegWriteEnable

BSel

ALUSel

MemRW

WBSel

alu

DataR

DataW

DMEM

alu

1

0

clk

Add

+4
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 \( rs2 \)
  - But have only one ALU – need more hardware
Adding Branches

RISC-V (53)

Garcia, Nikolić

Inst[31:0] | ImmSel = B | RegWEn = 0 | BrUn = B | BrEq = 1 | BrLT = 1 | Asel = 1 | ALUSel = add | MemRW = read | WB Sel = *
Branch Comparator

- \( \text{BrEq} = 1 \), if \( A=B \)
- \( \text{BrLT} = 1 \), if \( A<B \)
- \( \text{BrUn} = 1 \) selects unsigned comparison for \( \text{BrLT} \), \( 0=\text{signed} \)

**BGE branch:** \( A \geq B \), if \( A<B \)

\[
A<B = !(A<B)
\]
Branch Immediates (In Other ISAs)

- 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

Each instruction immediate bit can appear in one of two places in output immediate value – so need one 2-way mux per bit
### Branch Immediates (In Other ISAs)

- 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**

<table>
<thead>
<tr>
<th>S-Immediate</th>
<th>B-Immediate (shift left by 1)</th>
</tr>
</thead>
</table>

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>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]

- **I-type**
  - Inst[31]
  - Inst[30:25]
  - Inst[24:21]
  - Inst[20]

- **S-type**
  - Inst[31]
  - Inst[30:25]
  - Inst[11:8]
  - Inst[7]

- **B-type**
  - Inst[31]
  - Inst[7]
  - Inst[30:25]
  - Inst[11:8]
  - 0

Upper bits sign-extended from Inst[31] always

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

**RISC-V (57)**
Lighting Up Branch Path

- \( \text{ALUSel} = \text{add} \)
- \( \text{IMEM} \) and \( \text{DMEM} \) inputs:
  - \( \text{PC} = \text{pc} + 4 \)
  - \( \text{Inst[24:20]} \)
  - \( \text{AddrB}, \text{DataA} \)
  - \( \text{AddrD}, \text{DataD} \)
  - \( \text{Inst[31:20]} \)
  - \( \text{Inst[11:7]} \)
  - \( \text{AddrA} \)
- \( \text{alu} \) inputs:
  - \( \text{R[rs1]}, \text{R[rs2]} \)
  - \( \text{ImmSel} = B \)
  - \( \text{RegWEn} = 0 \)
  - \( \text{BrUn} = 0 \)
  - \( \text{BrLT} = 1 \)
  - \( \text{Bsel} = 1 \)
  - \( \text{ALUSel} = \text{add} \)
  - \( \text{MemRW} = \text{read} \)
  - \( \text{WBSel} = ^* \)

Control logic:
- \( \text{PCSel} = \text{taken/not taken} \)
- \( \text{Inst[31:0]} \)
- \( \text{ImmSel} = B \)
- \( \text{RegWEn} = 0 \)
- \( \text{BrUn} = 0 \)
- \( \text{BrLT} = 1 \)
- \( \text{Bsel} = 1 \)
- \( \text{ALUSel} = \text{add} \)
- \( \text{MemRW} = \text{read} \)
- \( \text{WBSel} = ^* \)

Garcia, Nikolić, RISC-V (58)
Adding JALR to Datapath
Let’s Add JALR (I-Format)

- JALR rd, rs, immediate
- Two changes to the state
  - Writes PC+4 to rd (return address)
  - Sets PC = rs1 + immediate
- Uses same immediates as arithmetic and loads
  - *no* multiplication by 2 bytes
  - LSB is ignored
Datapath So Far, With Branches

[Diagram of RISC-V datapath with branch logic]
Datapath With JALR

Control logic

- PCSel = taken
- Inst[31:0] = I
- RegWEn = 1
- BrUn = *
- BrLT = *
- Bsel = 1
- BrEq = *
- ASEL = Add
- MemRW = Read
- WBSEL = 2

PCSEL

IMEM

Addr

Inst

AddrA

DataA

AddrB

DataB

AddrD

DataD

Reg

Inst[19:15]

Inst[11:7]

Inst[24:20]

Inst[20:16]

Imm[31:0]

ImmSel = I

Imm. Gen

alu

Branch Comp

ALU

DataB

DMEM

DataD

DMEM

R[rs1]

R[rs2]

Label

Inst[31:20]

Inst[20:16]

Circuit diagram with implementing JALR instruction for RISC-V.

Garcia, Nikolić
Datapath With JALR

Control logic

Branch Comp

ALU

DataB

DMEM

addr

inst

IMEM

Imm. Gen

Inst[31:20]

Imm[31:0]

PCSel = taken

Inst[31:0]

ImmSel = I

RegWE

=1

BrUn = *

BrLT = *

Asel = 0

ALUSel = Add

MemRW

=Read

WBSel = 2
Adding JAL
Two changes to the state
- `jal` saves PC+4 in register `rd` (the return address)
- Set PC = PC + offset (PC-relative jump)

Target somewhere within ±2^{19} locations, 2 bytes apart
- ±2^{18} 32-bit instructions

Immediate encoding optimized similarly to branch instruction to reduce hardware cost
Datapath with JAL

Control logic:
- PCSel = taken
- Inst[31:0] = J
- ImmSel = Imm[31:0]
- RegWEn = 1
- BrUn = *
- BrLT = *
- Asel = 0
- ALUSel = Add
- MemRW = Read
- WBSel = 2

ALUSel = Add
ImmSel = Imm[31:0]
RegWEn = 1
BrUn = *
BrLT = *
Asel = 0
ALUSel = Add
MemRW = Read
WBSel = 2

PCSel = taken
Inst[31:0] = J
RegWEn = 1
BrUn = *
BrLT = *
Asel = 0
ALUSel = Add
MemRW = Read
WBSel = 2
Light Up JAL Path

Control logic

- PCSel = taken
- Inst[31:0] = J
- ImmSel = J
- RegWEn = 1
- BrUn = *
- BrLT = *
- Asel = 0
- ALUSel = Add
- MemRW = Read
- WBSel = 2

RISC-V (67)
Adding U-Types
U-Format for “Upper Immediate” Instructions

- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, \( rd \)
- Used for two instructions
  - \( \text{lui} \) – Load Upper Immediate
  - \( \text{auipc} \) – Add Upper Immediate to PC
Datapath With LUI, AUIPC

RISC-V (70)

Garcia, Nikolić
Lighting Up LUI

Control logic

PCSel = pc+4  Inst[31:0]  ImmSel = U  RegWE = 1  BrUn =*  BrLT =*  Asel =*  ALUSel =B  MemRW = Read  WBSel = 1

Inst[19:15]  AddrA  DataA  R[rs2]

Imm[31:0]  ImmGen

Branch Comp

alu

DataB  DMEM

DataA  IMEM

AddrA  AddrB  AddrD  DataA  DataB  AddrD  AddrB  AddrD  DataA  DataB

Add

+4

pc+4

clk

pc

inst

addr

Inst[31:20]  Imm[31:0]  Bsel = U  MemRW = Read  WBSel = 1

RISC-V (71)
Lighting Up AUIPC

RISC-V (72)
“And In Conclusion...”
Complete RV32I Datapath!

Control logic

IMEM

Add

PC

alu

DMEM

Reg [ ]

DataD

Inst[24:20]

Inst[19:15]

Inst[11:7]

addr

inst

+4

Inst[31:20]

Imm[31:10]

Imm. Gen

RegWE

Inst[31:0]

brEq

brLT

brUn

mem

MemRW

WBSel

alu

IMEM

AddrD

AddrA

AddrB

DataA

DataB

R[rs1]

R[rs2]

WBSel

MemRW

IMEM

addr

inst

Add

pc+4

Clk

ghart

ghart

ghart
Complete RV32I ISA!

### Open RISC-V Reference Card

#### Base Integer Instructions: RV32I

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Fmt</th>
<th>RV32I Base</th>
<th>Category</th>
<th>Name</th>
<th>Fmt</th>
<th>RV32I Base</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shifts</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>Shifts</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Left Logical</td>
<td>R SLL rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Loads</td>
<td>LB rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Left Log. Imm.</td>
<td>I SLLI rd,rs1,shamt</td>
<td></td>
<td></td>
<td>Load Halfword</td>
<td>LH rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Right Logical</td>
<td>R SRL rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Load Byte Unsigned</td>
<td>LBU rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Right Log. Imm.</td>
<td>I SRLI rd,rs1,shamt</td>
<td></td>
<td></td>
<td>Load Half Unsigned</td>
<td>LHU rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Right Arithmetic</td>
<td>R SRA rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Load Word</td>
<td>LW rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift Right Arith. Imm.</td>
<td>I SRAI rd,rs1,shamt</td>
<td></td>
<td></td>
<td>Stores</td>
<td>SB rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arithmetic</strong></td>
<td>ADD</td>
<td>R</td>
<td>ADD rd,rs1,rs2</td>
<td>Store Halfword</td>
<td>SH rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD Immediate</td>
<td>I ADDI rd,rs1,imm</td>
<td></td>
<td></td>
<td>Store Word</td>
<td>SW rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBtract</td>
<td>R SUB rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Branch =</td>
<td>BEQ rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Upper Imm</td>
<td>U LUI rd,imm</td>
<td></td>
<td></td>
<td>Branch =</td>
<td>BNE rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add Upper Imm to PC</td>
<td>U AUIPC rd,imm</td>
<td></td>
<td></td>
<td>Branch &lt;</td>
<td>BLT rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical XOR</td>
<td>R XOR rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Branch ≥</td>
<td>BGE rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR Immediate</td>
<td>I XORI rd,rs1,imm</td>
<td></td>
<td></td>
<td>Branch &lt; Unsigned</td>
<td>BLTU rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>R OR rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Branch ≥ Unsigned</td>
<td>BGEU rs1,rs2,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR Immediate</td>
<td>I ORI rd,rs1,imm</td>
<td></td>
<td></td>
<td>Jump &amp; Link</td>
<td>JAL rd,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>R AND rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Jump &amp; Link Register</td>
<td>JALR rd,rs1,imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND Immediate</td>
<td>I ANDI rd,rs1,imm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Set &lt;</td>
<td>R SLT rd,rs1,rs2</td>
<td></td>
<td></td>
<td>Synch</td>
<td>FENCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set &lt; Immediate</td>
<td>I SLTI rd,rs1,imm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set &lt; Unsigned</td>
<td>R SLTU rd,rs1,rs2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set &lt; Imm Unsigned</td>
<td>I SLTIU rd,rs1,imm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not in 61C
Review

- We have designed a complete 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
  - We still need to design it