Supporting Loads
R+I Arithmetic/Logic Datapath

**Control logic**
- ImmSel
- RegWriteEnable
- BSel
- ALUSel

**ALU**
- DataD
- AddrD
- AddrA
- DataA
- AddrB
- DataB

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

**Imm[31:0]**
- Imm[31:20]
- Inst[31:0]
- Inst[24:20]
- Inst[19:15]
- Inst[11:7]
- Inst[20:20]
- Inst[2:2]

**Imm. Gen**
- ImmSel

**Inst[31:0]**
- Inst[20:0]
- Inst[15:0]
- Inst[0:0]

**Add**
- +4

**Addr**
- Addr

**IMEM**
- IMEM
- PC
- clk

**Add**
- Add

**alu**
- clk

**RISC-V (37)**

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

- 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

1. **PC** (Program Counter) +4
2. **IMEM** (Instruction Memory)
3. **IMInst** (Instruction Register)
4. **alu** (Arithmetic Logic Unit)
5. **Reg[rs1]**
6. **Reg[rs2]**
7. **DataA**
8. **DataB**
9. **DataD**
10. **AddrA**
11. **AddrB**
12. **AddrD**
13. **Imm[31:0]**
14. **ImmSel**
15. **ALUSel**
16. **WBSel**
17. **MemRW**
18. **Clk**
19. **RegWriteEnable**
20. **BSeal**

**Control logic**

Components:
- **Add**
- **Imm. Gen**
- **Reg[ ]**
- **alu**
- **DMEM**
- **RISC-V (39)**
RISC-V (40)

**R+I Arithmetic/Logic Datapath**

- **pc+4**
- **+4**
- **Add**
- **IMEM**
- **Addr**
- **Inst**
- **Imm. Gen**
- **DataD**
- **AddrD**
- **DataA**
- **AddrA**
- **DataB**
- **AddrB**
- **DataD**
- **AddrD**
- **Reg[rs1]**
- **RegWEn=1**
- **BSel=1**
- **ALUSel=Add**
- **MemRW=Read**
- **WB Sel=0**

Control logic:
- **ImmSel = I**
- **RegWEn=1**
- **BSel=1**
- **ALUSel=Add**
- **MemRW=Read**
- **WB Sel=0**

Garcia, Nikolić
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
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>01110</td>
<td>00010</td>
<td>010</td>
<td>01000</td>
<td>0100011</td>
<td>STORE</td>
</tr>
</tbody>
</table>

- **offset[11:5]**
  - `=0`
  - `rs2=14`
  - `rs1=2`

- **offset[4:0]**
  - `SW = 8`

Combined 12-bit offset = 8
Datapath with lw

Control logic

IMEM

Add

Add

pc+4

clk

addr

inst

Inst[31:20]

Inst[19:15]

Inst[11:7]

Inst[24:20]

Reg [ ]

DataD

AddrD

AddrA

DataA

AddrB

Reg[rs1]

Reg[rs2]

alu

DataR

Addr

DMEM

DataD

AddrD

DataA

AddrA

AddrB

Reg[rs1]

Reg[rs2]

Imm[31:0]

ImmSel

RegWriteEnable

BSel

ALUSel

MemRW

WBSel

Inst[31:0]

immSel

clk

clk
Adding sw to Datapath

- ALUSel = Add
- IMEM addr = pc+4
- DMEM addr = Inst[24:20]
- Inst[24:20]
- Inst[19:15]
- Inst[11:7]
- AddrB = Inst[31:20]
- AddrD = Inst[19:15]
- DataA = AddrA DataB
- DataD = AddrD DataB
- Reg[rs1] = RegWriteEnable = 0
- Bsel = ImmSel = S
- ALUSel = Add
- MemRW = Write (don't care)
- WBSel = *
- clk
- add
- Inst[31:0]
- Control logic
- Imm[31:0]
- DataR
- DataW
- mem
- alu
Adding sw to Datapath

Control logic:
- ImmSel = S
- RegWEn = 0
- BSel = 1
- ALUSel = Add
- MemRW = Write
- WBSel = *

Network:
- pc+4
- clk
- AddrD
- AddrA
- AddrB
- DataA
- DataB
- DataD
- Inst[31:20]
- Inst[31:7]
- Imm[31:0]
- alu
- mem
- IMEM
- DMEM
- Inst[19:15]
- Inst[24:20]
- Inst[11:7]
- Inst[20]
- Reg[rs1]
- Reg[rs2]
- Reg[
- DataW
- addr
- dmem

Garcia, Nikolić
• 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>Imm[11:5]</td>
<td>rs2</td>
<td>rs1</td>
<td>001</td>
<td>imm[4:0]</td>
<td>0100011</td>
</tr>
<tr>
<td>sh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

width
Implementing Branches
RISC-V B-Format for 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

- IMEM
- PC
- Add
- ALU
- Reg [ ]
- ALUSel
- ImmSel
- BSel
- MemRW
- WBSel
- DataA
- DataB
- DataD
- AddrA
- AddrB
- AddrD
- Inst[0:31]
- Imm[0:31]
- Inst[11:7]
- Inst[19:15]
- Inst[24:20]
- Inst[20:20]
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

Garcia, Nikolić

Adding Branches

Control logic

PCSel = taken/not taken

Inst[31:0]

ImmSel = B

RegWEn = 0

BrUn = 1

BrLT = 1

Asel = 1

ALUSel = add

MemRW = read

WBSel = *

Inst[11:7]

Inst[19:15]

Inst[24:20]

AddrD

DataA

DataB

R[rs1]

R[rs2]

Imm[31:0]

PC

addr

inst

IMEM

AddrB

AddrA

IMEM

addr

DMEM

pc + 4

Add

+4

Add

clk

addr

inst

IMEM

Inst[24:20]

Inst[19:15]

Inst[11:7]

AddrB

DataA

DataB

alu

R[rs1]

R[rs2]

Imm[31:0]

MemRW

WBSel

BrUn

BrLT

Bsel

Asel

ALUSel

MemRW

WBSel

Berkeley

University of California

RISC-V (53)
Branch Comparator

\[ \text{BrEq} = 1, \text{if } A=B \]

\[ \text{BrLT} = 1, \text{if } A<B \]

\[ \text{BrUn} = 1 \text{ selects unsigned comparison for } \text{BrLT}, \]  
\[ 0=\text{signed} \]

\[ \text{BGE branch: } A \geq B, \text{if } A<B \]

\[ A<B \implies \neg \text{(A}<B) \]
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S-Immediate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Immediate</td>
<td>(shift left by 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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>(\text{imm}[11:0])</td>
<td>(\text{rs1})</td>
<td>(\text{funct3})</td>
<td>(\text{rd})</td>
<td>(\text{opcode})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{imm}[11:5])</td>
<td>(\text{rs2})</td>
<td>(\text{rs1})</td>
<td>(\text{funct3})</td>
<td>(\text{imm}[4:0])</td>
<td>(\text{opcode})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{imm}[12\mid10:5])</td>
<td>(\text{rs2})</td>
<td>(\text{rs1})</td>
<td>(\text{funct3})</td>
<td>(\text{imm}[4:1\mid11])</td>
<td>(\text{opcode})</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**
  - \(-\text{inst}[31]-\)
  - \(\text{inst}[30:25] \quad \text{inst}[24:21] \quad \text{inst}[20]\)

- **S-type**
  - \(-\text{inst}[31]-\)
  - \(\text{inst}[30:25] \quad \text{inst}[11:8]\)
  - \(\text{inst}[7]\)

- **B-type**
  - \(-\text{inst}[31]-\)
  - \(\text{inst}[7] \quad \text{inst}[30:25] \quad \text{inst}[11:8] \quad 0\)

Upper bits sign-extended from \(\text{inst}[31]\) always

Only bit 7 of instruction changes role in immediate between S and B
Lighting Up Branch Path

Control logic

PCSel = taken/not taken

Inst[31:0]

ImmSel = B

RegWEn = 0

BrUn = 1

BrEq = 1

Asel = 1

ALUSel = add

MemRW = read

WBSel = *

Reg [ ]

Inst[19:15]

Inst[11:7]

Inst[24:20]

AddrD

DataD

R[rs1]

R[rs2]

AddrB

AddrA

DataA

alu

_addr inst

IMEM

+4

PC

clk

Imm. Gen

Inst [31:20]

MemRW

WBSel

IMEM

DMEM

PC

pc+4

Add

cl

branch

Comp

DataB

DataD

Mem

clk

0

1
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
Datapath With JALR

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

RISC-V (62)
Datapath With JALR

- **PCSel** = taken
- **Inst[31:0]**
- **ImmSel** = I
- **Reg WE** = 1
- **Branch Comp**
- **Mem RW** = Read
- **WB Sel** = 2

Control logic:
- **Br Un** = *
- **Br LT** = *
- **B sel** = 1
- **Asel** = 0
- **ALU Sel** = Add

- **IMEM**
- **DMEM**

**Datapath Components:**
- **R[rs1]**
- **R[rs2]**
- **Imm[31:0]**
- **Imm Gen**
- **Addr A Data A**
- **Addr B Data B**
- **Addr D Data D**
- **alu**
- **PC**
- **Inst[11:7]**
- **Inst[19:15]**
- **Inst[24:20]**

**Other Symbols:**
- **clk**
- **addr**
- **inst**
- **0**
- **1**
- **2**

Garcia, Nikolić
Adding JAL
### J-Format for Jump Instructions

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>12</th>
<th>11</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
</table>

- Two changes to the state
  - \text{jal} saves PC+4 in register \text{rd} (the return address)
  - Set PC = PC + offset (PC-relative jump)

- Target somewhere within $\pm2^{19}$ locations, 2 bytes apart
  - $\pm2^{18}$ 32-bit instructions

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

![Datapath Diagram]

- **IMEM**
  - pc+4
  - addr
  - clk
- **DMEM**
  - addr
  - DMEM
- **PC**
  - pc+4
  - clk
- **AddrB**
  - Inst[24:20]
- **AddrA**
  - Inst[19:15]
- **Inst[11:7]**
- **DataA**
- **Reg[ ]**
- **DataB**
- **ALU**
  - R[rs1]
  - R[rs2]
- **Imm[31:0]**
- **ImmSel**
  - Imm[31:0]
- **WBSel**
  - MemRW = Read
- **MemRW**
- **RegWEn**
  - 1
- **Branch Comp**
  - BrUn =*
  - BrLT =*
  - BrEq =*
  - Bsel = 1
- **Asel**
  - ALUSel = Add
- **PCSel**
  - 1
- **Imm. Gen**
  - ImmSel = J
- **Control logic**
  - Branch
  - Comp
  - MemRW
  - BrUn
  - BrLT
  - BrEq
  - Bsel
  - Asel
  - ALUSel
  - MemRW
  - WBSel

Garcia, Nikolić

RISC-V (66)
Light Up JAL Path

![Diagram of RISC-V JAL Path]

- **IMEM**: Instruction Memory
- **PC**: Program Counter
- **Inst**: Instruction
- **Addr**: Address
- **Data**: Data
- **Reg**: Register
- **Imm**: Immediate
- **alu**: ALU
- **ALU Sel**: ALU Selection
- **Branch Comp**: Branch Comparator
- **Mem RW**: Memory Read-Write
- **Branch**: Branch Condition
- **PC Sel**: Program Counter Selection
- **Mem**: Memory
- **Addr**: Address
- **Data**: Data
- **Reg**: Register
- **Imm**: Immediate
- **Inst**: Instruction
- **Control logic**: Control Logic

- **ALU Sel = Add**: ALU selection for addition
- **Mem RW = Read**: Memory read mode
- ** Branch = J**: Branch condition
- **PC Sel = taken**: Program counter selection for branch
- **Reg WE = 1**: Register write enable
- **Imm Sel = J**: Immediate selection
- **Inst[31:0]**: Instruction bits
- **Inst[24:20]**: Instruction bits
- **Inst[19:15]**: Instruction bits
- **Inst[11:7]**: Instruction bits
- **Inst[24:20]**: Instruction bits
- **Inst[11:0]**: Instruction bits

**RISC-V (67)**
Adding U-Types
### U-Format for "Upper Immediate" Instructions

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-immediate[31:12]</td>
<td>dest LUI</td>
</tr>
<tr>
<td>U-immediate[31:12]</td>
<td>dest AUIPC</td>
</tr>
</tbody>
</table>

- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, **rd**
- Used for two instructions
  - **lui** – Load Upper Immediate
  - **auipc** – Add Upper Immediate to PC
Datapath With LUI, AUIPC
Lighting Up LUI

Control logic:
- PCSel = pc + 4
- Inst[31:0]
- ImmSel = U
- RegWE = 1
- BrUn = *
- BrLT = *
- MemRW = Read
- MemSel = Read
- WBSel = 1

Logic:
- IMEM
- DMEM
- ALU
- AddrA
- DataA
- AddrB
- DataB
- AddrD
- DataD
- AddrB
- DataB

Signals:
- clk
- addr
- inst
- IMEM
- DMEM
- PC
- pc + 4
- BrUn
- BrLT
- BrEq
- Asel
- ALUSel
- MemRW
- WBSel

Values:
- R[rs1]
- R[rs2]
- Imm[31:0]
- ImmSel
- Imm. Gen
- Inst[31:20]
- Inst[11:7]
- Inst[19:15]
- Inst[24:20]
- Reg[]
- DataD
- Address
- Branch Comp
- pc
- pc + 4
- MemSel
- BrSel
- ALUSel
- MemRW
- WBSel

Boolean Expressions:
- ALUSel = B
- MemSel = Read
- WBSel = 1
- BrUn = *
- BrLT = *
- BrEq = *
- Asel = *
- ALUSel = B
- MemRW = Read
- WBSel = 1

Other:
- Garcia, Nikolić
- RISC-V (71)
Lighting Up AUIPC

![AUIPC Circuit Diagram]

- **Control logic**
  - PCSel = pc+4
  - Inst[31:0] = U
  - ImmSel = U
  - RegWE = 1
  - BrUn = *
  - BrEq = *
  - MemSel = add
  - MemRW = Read
  - WBSel = 1

- **IMEM**
  - Inst[31:20]
  - Inst[19:15]
  - Inst[11:7]
  - Inst[24:20]

- **IMU**
  - AddrD
  - AddrA
  - AddrB

- **DataA**
  - DataB

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

- **Imm[31:0]**
  - ImmSel = U

- **Branch Comp**
  - PCSel = pc+4
  - BrUn = *
  - BrEq = *
  - BrLT = *

- **ALU**
  - Asel = *
  - ALUSel = add

- **DMEM**
  - Addr

- **alu**
“And In Conclusion...”
Complete RV32I Datapath!

![Diagram of RV32I Datapath](image_url)
# Complete RV32I ISA!

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

Not in 61C
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