CS 61C:
Great Ideas in Computer Architecture

Lecture 13: RISC-V Control & Operating Speed
Agenda

• Completion of Single-Cycle RISC-V Datapath
• Controller
• Instruction Timing
• Performance Measures
• Introduction to Pipelining
• Pipelined RISC-V Datapath
• And in Conclusion, ...
Implementing jal Instruction

- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
  - Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
    - $\pm 2^{18}$ 32-bit instructions
  - Immediate encoding optimized similarly to branch instruction to reduce hardware cost
Adding jal to datapath
Adding jal to datapath
Single-Cycle RISC-V RV32I Datapath
Recap: Complete RV32I ISA

RV32I has 47 instructions total
37 instructions covered in CS61C

Not in CS61C

Remaining instructions (ex: lui, auipc) can be implemented with no significant additions to the datapath: adding a “pass B” option to the ALU and another immediate decoding option. Rest is all control logic.
And in Conclusion, ...

• Universal datapath
  • Capable of executing all RISC-V instructions in one cycle each
  • datapath is the “union” of all the units used by all the instructions. Muxes provide the options.
  • 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
Agenda

- Finish Single-Cycle RISC-V Datapath
- **Controller**
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...
Processor

Processor-Memory Interface

Enable?  Read/Write

Address

Write  Data

Read  Data

Program  Bytes  Data

Memory

Processor

Control

Datapath

PC

Registers

Arithmetic & Logic Unit (ALU)
Single-Cycle RISC-V RV32I Datapath
### Control Logic “Truth Table” (incomplete)

<table>
<thead>
<tr>
<th>Inst[31:0]</th>
<th>BrEq</th>
<th>BrLT</th>
<th>PCSel</th>
<th>ImmSel</th>
<th>BrUn</th>
<th>ASel</th>
<th>BSel</th>
<th>ALUSel</th>
<th>MemRW</th>
<th>RegWEn</th>
<th>WBSel</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>-</td>
<td>-</td>
<td>Reg</td>
<td>Reg</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>ALU</td>
</tr>
<tr>
<td>sub</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>-</td>
<td>-</td>
<td>Reg</td>
<td>Reg</td>
<td>Sub</td>
<td>Read</td>
<td>1</td>
<td>ALU</td>
</tr>
<tr>
<td>(R-R Op)</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>-</td>
<td>-</td>
<td>Reg</td>
<td>Reg</td>
<td>(Op)</td>
<td>Read</td>
<td>1</td>
<td>ALU</td>
</tr>
<tr>
<td>addi</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>I</td>
<td>-</td>
<td>Reg</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>ALU</td>
</tr>
<tr>
<td>lw</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>I</td>
<td>-</td>
<td>Reg</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>Mem</td>
</tr>
<tr>
<td>sw</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>S</td>
<td>-</td>
<td>Reg</td>
<td>Imm</td>
<td>Add</td>
<td>Write</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>beq</td>
<td>0</td>
<td>*</td>
<td>+4</td>
<td>B</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>beq</td>
<td>1</td>
<td>*</td>
<td>ALU</td>
<td>B</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>bne</td>
<td>0</td>
<td>*</td>
<td>ALU</td>
<td>B</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>bne</td>
<td>1</td>
<td>*</td>
<td>+4</td>
<td>B</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>blt</td>
<td>*</td>
<td>1</td>
<td>ALU</td>
<td>B</td>
<td>0</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>bltu</td>
<td>*</td>
<td>1</td>
<td>ALU</td>
<td>B</td>
<td>1</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>jalr</td>
<td>*</td>
<td>*</td>
<td>ALU</td>
<td>I</td>
<td>-</td>
<td>Reg</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>PC+4</td>
</tr>
<tr>
<td>jal</td>
<td>*</td>
<td>*</td>
<td>ALU</td>
<td>J</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>PC+4</td>
</tr>
<tr>
<td>auipc</td>
<td>*</td>
<td>*</td>
<td>+4</td>
<td>U</td>
<td>-</td>
<td>PC</td>
<td>Imm</td>
<td>Add</td>
<td>Read</td>
<td>1</td>
<td>ALU</td>
</tr>
</tbody>
</table>

* means “for all values”  
- means “don’t care, use any value”
Note: Instruction type encoded using only 9 bits
inst[30], inst[14:12], inst[6:2]

<table>
<thead>
<tr>
<th>imm[31:12]</th>
<th>rd</th>
<th>0110111</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[31:12]</td>
<td>rd</td>
<td>0010111</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>000</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>000</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>001</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>010</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>100</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>101</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>000</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>010</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>101</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>110</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rsl</td>
<td>111</td>
</tr>
</tbody>
</table>

**inst[30]**

- LUI
- AUIPC

**inst[14:12]**

- JAL
- JALR
- BEQ
- BNE
- BLT
- BGE
- BLTU
- BGEU
- LB
- LHI
- LW
- LBU
- LHU

**inst[6:2]**

Not in CS61C

<table>
<thead>
<tr>
<th>imm[31:12]</th>
<th>shamt</th>
<th>rsl</th>
<th>000</th>
<th>rd</th>
<th>0010011</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>101</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>110</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>111</td>
<td>rd</td>
<td>0010011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>000</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>010</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>011</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>101</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>110</td>
<td>rd</td>
<td>0110011</td>
</tr>
<tr>
<td>imm[31:12]</td>
<td>shamt</td>
<td>rsl</td>
<td>111</td>
<td>rd</td>
<td>0110011</td>
</tr>
</tbody>
</table>

SLLI
SRLI
SRAI
ADD
SUB
SLT
SLTU
SLT
XOR
SRL
SRA
OR
AND
FENCE
FENCE.I
ECALL
EBREAK
CSRW
CSRR
CSRRC
CSRWR
CSRSRI
CSRRCI
Control Block Design

11-bit input

Inst[30,14:12,6:2] -> BrEq -> BrLT

15-bit output

PCSel
ImmSel[2:0]
BrUn
ASel
BSel
ALUSel[3:0]
MemRW
RegWEn
WBSel[1:0]
Controller Realization Options

- **ROM (Read-Only Memory)**
  - Regular structure made from transistors
  - Can be easily reprogrammed during the design process to
    - fix errors
    - add instructions
  - Popular when designing control logic manually

- **Combinatorial Logic**
  - Today, chip designers often use logic synthesis tools to convert truth tables to networks of gates
  - Logic equation for each control signal (common sub-expressions shared among control signal equations)
  - Can exploit output “don’t cares” and input “for all values” to simplify circuit.
ROM (read only memory) Controller Implementation

11-bit address

Inst[]
BrEQ
BrLT

Address Decoder

add
sub
or
jal

Control Word for add
Control Word for sub
Control Word for or

Controller output (PCSel, ImmSel, …)

15 data bits
Agenda

• Finish Single-Cycle RISC-V Datapath
• Controller
• **Instruction Timing**
• Performance Measures
• Introduction to Pipelining
• Pipelined RISC-V Datapath
• And in Conclusion, ...
Typical Approximate Worst-Case Instruction Timing

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-MEM</td>
<td>Reg Read</td>
<td>ALU</td>
<td>D-MEM</td>
<td>Reg W</td>
<td></td>
</tr>
<tr>
<td>200 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td>200 ps</td>
<td>100 ps</td>
<td>800 ps</td>
</tr>
</tbody>
</table>
Instruction Timing

- Maximum clock frequency
  \[ f_{max} = \frac{1}{800\text{ps}} = 1.25 \text{ GHz} \]
- Most blocks idle most of the time
  - E.g. \[ f_{max,\text{ALU}} = \frac{1}{200\text{ps}} = 5 \text{ GHz}! \]
- How can we keep data-path resources (such as ALU) busy all the time?
- For ALU could have 5 billion adds/sec, rather than just 1.25 billion?
- Idea: Factories “assembly line” - all equipment is always busy!
Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- Instruction Timing
- **Performance Measures**
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...
Performance Measures

• “Our” RISC-V executes instructions at 1.25 GHz
  • 1 instruction every 800 ps
• Can we improve its performance?
  • What do we mean with this statement?
  • Not so obvious:
    • Less time for each instruction?
    • More instructions per unit time?
    • Aren’t these the same? Yes, for our simple single-cycle processor, but not so when we employ parallelism.
    • Is energy efficiency a measure of performance?
# Transportation Analogy

<table>
<thead>
<tr>
<th></th>
<th>Race Car</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passenger Capacity</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Travel Speed</td>
<td>200 mph</td>
<td>50 mph</td>
</tr>
<tr>
<td>Gas Mileage</td>
<td>5 mpg</td>
<td>2 mpg</td>
</tr>
</tbody>
</table>

## 50 Mile trip:

<table>
<thead>
<tr>
<th></th>
<th>Race Car</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travel Time</td>
<td>15 min</td>
<td>60 min</td>
</tr>
<tr>
<td>Time for 100 passengers</td>
<td>1500 min</td>
<td>120 min</td>
</tr>
<tr>
<td>Gallons per passenger</td>
<td>10 gallons</td>
<td>0.5 gallons</td>
</tr>
</tbody>
</table>
**Processor Analogy**

<table>
<thead>
<tr>
<th>Transportation</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trip Time</td>
<td>Instruction execution time (<em>latency</em>)</td>
</tr>
<tr>
<td>Time for 100 passengers</td>
<td>Total number of instructions executed per unit time (<em>throughput</em>)</td>
</tr>
<tr>
<td>Gallons per passenger</td>
<td>Energy per instruction (<em>energy efficiency</em>): e.g. how many total instructions executed per battery charge or per unit on energy bill for datacenter</td>
</tr>
</tbody>
</table>
## Computer Task-level Analogy

<table>
<thead>
<tr>
<th>Transportation</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trip Time</td>
<td>Program execution time (<strong>latency</strong>):</td>
</tr>
<tr>
<td></td>
<td>e.g. time to update display</td>
</tr>
<tr>
<td>Time for 100 passengers</td>
<td>Total number of tasks per unit time (<strong>throughput</strong>):</td>
</tr>
<tr>
<td></td>
<td>e.g. number of server requests handled per hour</td>
</tr>
<tr>
<td>Gallons per passenger</td>
<td>Energy per task (<strong>energy efficiency</strong>):</td>
</tr>
<tr>
<td></td>
<td>e.g. how many movies you can watch per battery charge or energy bill for datacenter</td>
</tr>
</tbody>
</table>
"Iron Law" of Processor Performance

\[
\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}
\]
Instructions per Program

- Determined by
  - Task specification
  - Algorithm, e.g. $O(N^2)$ vs $O(N)$
  - Programming language
  - Compiler
  - Instruction Set Architecture (ISA)

\[
\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}
\]
(Average) Clock cycles per Instruction

- Determined by
  - ISA (CISC versus RISC)
  - Processor implementation (or **microarchitecture**)
    - E.g. for “our” single-cycle RISC-V design, CPI = 1
  - Pipelined processors, CPI > 1 (next lecture)
  - Superscalar processors, CPI < 1 (next lecture)
Time per Cycle (1/Frequency)

\[
\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}
\]

- Determined by
  - Processor microarchitecture (determines critical path through logic gates)
  - Technology (e.g. 5nm versus 14nm)
  - Supply voltage (lower voltage reduces transistor speed, but improves energy efficiency)
Speed Tradeoff Example

• For some task (e.g. image compression) …

<table>
<thead>
<tr>
<th></th>
<th>Processor A</th>
<th>Processor B</th>
</tr>
</thead>
<tbody>
<tr>
<td># Instructions</td>
<td>1 Million</td>
<td>1.5 Million</td>
</tr>
<tr>
<td>Average CPI</td>
<td>2.5</td>
<td>1</td>
</tr>
<tr>
<td>Clock rate $f$</td>
<td>2.5 GHz</td>
<td>2 GHz</td>
</tr>
<tr>
<td>Execution time</td>
<td>1 ms</td>
<td>0.75 ms</td>
</tr>
</tbody>
</table>

Processor B is faster for this task, despite executing more instructions and having a lower clock rate!
Energy per Task

\[
\frac{\text{energy}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{energy}}{\text{instruction}} \\
\frac{\text{energy}}{\text{program}} \propto \frac{\text{instructions}}{\text{program}} \cdot C V_{dd}^2
\]

“Capacitance” depends on technology, microarchitecture, circuit details

Supply voltage, e.g. 1V

Want to reduce capacitance and voltage to reduce energy/task
Energy Tradeoff Example

• For instance, “Next-generation” processor (Moore’s law):
  • Capacitance, C: reduced by 15 %
  • Supply voltage, $V_{sup}$: reduced by 15 %
  • Energy consumption: $(.85C)(.85V)^2 = .63E \Rightarrow -39 \%$ reduction

• Significantly improved energy efficiency thanks to
  • Moore’s Law AND
  • Reduced supply voltage
Energy “Iron Law”

• Energy efficiency (e.g., instructions/Joule) is key metric in all computing devices

• For power-constrained systems (e.g., 20MW datacenter), need better energy efficiency to get more performance at same power

• For energy-constrained systems (e.g., 1W phone), need better energy efficiency to prolong battery life

\[
\text{performance} = \text{power} \cdot \text{energy efficiency} \\
\text{(tasks/second)} \quad \text{(Joules/sec)} \quad \text{(tasks/Joule)}
\]
End of Scaling

• In recent years, industry has not been able to reduce supply voltage much, as reducing it further would mean increasing "leakage power" where transistor switches don’t fully turn off (more like dimmer switch than on-off switch)

• Also, size of transistors and hence capacitance, not shrinking as much as before between transistor generations
  - Rather than horizontal modern CMOS uses vertically-aligned transistors to pack them closer together... But that doesn't reduce capacitance just allows for higher density

• Power becomes a growing concern – the “power wall”

• Cost-effective air-cooled chip limit around ~150W
Processor Trends

[Olukotun, Hammond, Sutter, Smith, Batten]
Agenda

• Finish Single-Cycle RISC-V Datapath
• Controller
• Instruction Timing
• Performance Measures
• Introduction to Pipelining
• Pipelined RISC-V Datapath
• And in Conclusion, ...
Pipelining

• A familiar example:
  • Getting a university degree

  Year 1
  Year 2
  Year 3
  Year 4

• Shortage of Computer scientists (your startup is growing):
  • How long does it take to educate 16,000 students?
Computer Scientist Education

• **Option 1:** *serial*
  4000 enter
  4000 graduate
  4000 graduate
  4000 graduate
  4000 graduate
  4000

  16,000 in 16 years, average throughput is 1000/year

• **Option 2:** *pipelining*
  1 year
  4000 graduate
  4000 graduate
  4000 graduate
  4000 graduate

  7 years

  • 16,000 in 7 years
  • Steady state throughput is 4000/year
  • Resources used efficiently
  • *4-fold improvement over serial education*
Latency versus Throughput

**Latency**
- Time from entering college to graduation
  - Serial: 4 years
  - Pipelining: 4 years

**Throughput**
- Average number of students graduating each year
  - Serial: 1000
  - Pipelining: 4000

**Pipelining**
- Increases throughput (4x in this example)
- But can *never improve* latency
  - sometimes worse (additional overhead)
Simultaneous versus Sequential

- What happens *sequentially*?
- What happens *simultaneously*? A form of parallel processing!
Agenda

• Finish Single-Cycle RISC-V Datapath
• Controller
• Instruction Timing
• Performance Measures
• Introduction to Pipelining
• **Pipelined RISC-V Datapath**
• And in Conclusion, ...
# Pipelining with RISC-V

The following table and diagram illustrate the process of pipeline stages for executing instructions in a RISC-V architecture. Each stage, labeled as `Instruction Fetch`, `Reg Read`, `ALU`, `Memory`, and `Register Write`, is represented with a pictogram and its associated time delay. The cycle time (`t_cycle`) for a pipeline stage is compared between pipelined and serial execution.

## Table: Pipeline Stages and Timing

<table>
<thead>
<tr>
<th>Phase</th>
<th>Pictogram</th>
<th>$t_{step}$ Serial</th>
<th>$t_{cycle}$ Pipelined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td><img src="image" alt="IM" /></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Reg Read</td>
<td><img src="image" alt="Reg" /></td>
<td>100 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>ALU</td>
<td><img src="image" alt="ALU" /></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Memory</td>
<td><img src="image" alt="DM" /></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Register Write</td>
<td><img src="image" alt="Reg" /></td>
<td>100 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>$t_{instruction}$</td>
<td><img src="image" alt="Pipeline" /></td>
<td>800 ps</td>
<td>1000 ps</td>
</tr>
</tbody>
</table>

## Diagram: Instruction Sequence

An instruction sequence is executed as follows:
- `add t0, t1, t2`
- `or t3, t4, t5`
- `sll t6, t0, t3`

The total time for instruction execution is $t_{instruction}$. The cycle time ($t_{cycle}$) is the sum of all pipeline stage delays, which is $1000$ ps in pipelined execution.
Pipelining with RISC-V

add t0, t1, t2
or t3, t4, t5
sll t6, t0, t3

**Single Cycle**

<table>
<thead>
<tr>
<th>Timing</th>
<th>$t_{step} = 100 \ldots 200$ ps</th>
<th>$t_{cycle} = 200$ ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Register access only 100 ps)</td>
<td></td>
<td>All cycles same length</td>
</tr>
<tr>
<td>Instruction time, $t_{instruction}$</td>
<td>$= t_{cycle} = 800$ ps</td>
<td>1000 ps</td>
</tr>
<tr>
<td>Clock rate, $f_s$</td>
<td>$1/800$ ps $= 1.25$ GHz</td>
<td>$1/200$ ps $= 5$ GHz</td>
</tr>
<tr>
<td>Relative speed</td>
<td>1 x</td>
<td>4 x</td>
</tr>
</tbody>
</table>
Sequential vs Simultaneous

What happens sequentially, what happens simultaneously?

$\text{t}_{\text{cycle}} = 200 \text{ ps}$

$\text{t}_{\text{instruction}} = 1000 \text{ ps}$

- add $t0, t1, t2$
- or $t3, t4, t5$
- sll $t6, t0, t3$
- sw $t0, 4(t3)$
- lw $t0, 8(t3)$
- addi $t2, t2, 1$
RISC-V Pipeline

- add $t0$, $t1$, $t2$
- or $t3$, $t4$, $t5$
- slt $t6$, $t0$, $t3$
- sw $t0$, 4($t3$)
- lw $t0$, 8($t3$)
- addi $t2$, $t2$, 1

$t_{\text{cycle}} = 200$ ps

$t_{\text{instruction}} = 1000$ ps

Resource use of instruction over time

Resource use in a particular time slot
Single-Cycle RISC-V RV32I Datapath

- **IMEM**
  - pc+4
  - pc
- **+4**
- **ALU**
- **Imm. Gen**
- **Reg[]**
  - DataD
  - AddrD
  - AddrA
  - DataA
  - AddrB
  - DataB
  - +4
  - wb
- **Branch Comp.**
  - Reg[rs1]
  - Reg[rs2]
- **DMEM**
  - Addr
  - DataR
  - DataW
  - pc+4
  - mem
  - wb
- **alu**
- **mem**
- **wb**
- **alu**

- **Inst**
  - inst[31:0]
  - ImmSel
  - RegWEn
  - BrUn
  - BrEq
  - BrLT
  - BSel
  - ASel
  - ALUSel
  - MemRW
  - WBSel

- **Reg[rs1]**
- **Reg[rs2]**
Pipelining RISC-V RV32I Datapath

Instruction Fetch (F)

Instruction Decode/Register Read (D)

ALU Execute (X)

Memory Access (M)

Write Back (W)
Recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline

Must pipeline instruction along with data, so control operates correctly in each stage
Each stage operates on different instruction

Pipeline registers separate stages, hold data for each instruction in flight
Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
  - Information is stored in pipeline registers for use by later stages
And in Conclusion, ...

- **Controller**
  - Tells universal datapath how to execute each instruction

- **Instruction timing**
  - Set by instruction complexity, architecture, technology
  - Pipelining increases clock frequency, “instructions per second”
    - But does not reduce time to complete instruction

- **Performance measures**
  - Different measures depending on objective
    - Response time
    - Jobs / second
    - Energy per task
Agenda

- RISC-V Pipeline
- Pipeline Control
- Next time:
  - Hazards
    - Structural
    - Data
      - R-type instructions
      - Load
    - Control
  - Superscalar processors