Agenda

• Finish Single-Cycle RISC-V Datapath
• Controller
• Instruction Timing
• Performance Measures
• Introduction to Pipelining
• Pipelined RISC-V Datapath
• And in Conclusion, ...
Recap: Adding branches to datapath
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
    - *no* multiplication by 2 bytes
Adding \texttt{jalr} to datapath
Adding jalr to datapath
Implementing `jal` Instruction

- JAL saves PC+4 in Reg[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
Adding *jal* to datapath
Adding jal to datapath
Recap: Complete RV32I ISA

RV32I has 47 instructions total
37 instructions covered in CS61C

Other instructions (ex: lui, auipc)
can be implemented with no new additions to
the datapath and only by changing control

Not in CS61C
Single-Cycle RISC-V RV32I Datapath
What are proper control signals for \texttt{lui} instruction?

A: BSel=0, ASel=0, WBSel=0

B: BSel=0, ASel=0, WBSel=1

C: BSel=0, ASel=1, WBSel=1

D: BSel=1, ASel=1, WBSel=1
Implementing lui
Agenda

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

Control

Datapath

Registers

Arithmetic & Logic Unit (ALU)

PC

Memory

Enable?

Read/Write

Address

Write Data

Read Data

Program

Data

Bytes

Processor-Memory Interface
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>
Instruction type encoded using only 9 bits

\[ \text{inst}[30], \text{inst}[14:12], \text{inst}[6:2] \]
Control Block Design

11-bit address (inputs)

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

Combinational Logic Function(s)

Comb

15 data bits (outputs)

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

• ROM
  – “Read-Only Memory”
  – Regular structure
  – Can be easily reprogrammed
    ▪ fix errors
    ▪ add instructions
  – Popular when designing control logic manually

• Combinatorial Logic
  – Today, chip designers use logic synthesis tools to convert truth tables to networks of gates
ROM Controller Implementation

**Address Decoder**

- **Inst[]**
- **BrEQ**
- **BrLT**

**Control Word for**
- **add**
- **sub**
- **or**

**Controller output (PCSel, ImmSel, ...)**
Agenda

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

<table>
<thead>
<tr>
<th></th>
<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>200 ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>
### Instruction Timing

<table>
<thead>
<tr>
<th>Instr</th>
<th>IF = 200ps</th>
<th>ID = 100ps</th>
<th>ALU = 200ps</th>
<th>MEM=200ps</th>
<th>WB = 100ps</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
<tr>
<td>jal</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
<tr>
<td>lw</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>700ps</td>
</tr>
</tbody>
</table>

- **Maximum clock frequency**
  - \( f_{\text{max}} = 1/800\text{ps} = 1.25 \text{ GHz} \)

- **Most blocks idle most of the time**
  - E.g. \( f_{\text{max,ALU}} = 1/200\text{ps} = 5 \text{ GHz}! \)
  - How can we keep ALU busy all the time?
  - 5 billion adds/sec, rather than just 1.25 billion?
  - Idea: Factories use three employee shifts - 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:
    ▪ Quicker response time, so one job finishes faster?
    ▪ More jobs per unit time (e.g. web server returning pages)?
    ▪ Longer battery life?
## Transportation Analogy

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

### 50 Mile trip:

<table>
<thead>
<tr>
<th></th>
<th>Sports Car</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Travel Time</strong></td>
<td>15 min</td>
<td>60 min</td>
</tr>
<tr>
<td><strong>Time for 100 passengers</strong></td>
<td>750 min</td>
<td>120 min</td>
</tr>
<tr>
<td><strong>Gallons per passenger</strong></td>
<td>5 gallons</td>
<td>0.5 gallons</td>
</tr>
</tbody>
</table>
## Computer Analogy

<table>
<thead>
<tr>
<th>Transportation</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trip Time</td>
<td>Program execution time:</td>
</tr>
<tr>
<td></td>
<td>e.g. time to update display</td>
</tr>
<tr>
<td>Time for 100 passengers</td>
<td>Throughput:</td>
</tr>
<tr>
<td></td>
<td>e.g. number of server requests handled per</td>
</tr>
<tr>
<td></td>
<td>hour</td>
</tr>
<tr>
<td>Gallons per passenger</td>
<td>Energy per task*:</td>
</tr>
<tr>
<td></td>
<td>e.g. how many movies you can watch per</td>
</tr>
<tr>
<td></td>
<td>battery charge or</td>
</tr>
<tr>
<td></td>
<td>energy bill for datacenter</td>
</tr>
</tbody>
</table>

* **Note:** power is not a good measure, since low-power CPU might run for a long time to complete one task consuming more energy than faster computer running at higher power for a shorter time.
“Iron Law” of Processor Performance

\[
\text{Time}_{\text{Program}} = \frac{\text{Instructions}_{\text{Program}} \times \text{Cycles}_{\text{Instruction}}}{\text{Cycle}_{\text{Cycle}}}
\]
Instructions per Program

Determined by

• Task
• Algorithm, e.g. $O(N^2)$ vs $O(N)$
• Programming language
• Compiler
• Instruction Set Architecture (ISA)
(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
- Superscalar processors, CPI < 1 (next lecture)
Time per Cycle (1/Frequency)

Determined by

• Processor microarchitecture (determines critical path through logic gates)
• Technology (e.g. 14nm versus 28nm)
• Power budget (lower voltages reduce transistor speed)
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

<table>
<thead>
<tr>
<th>Energy</th>
<th>Instructions</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Program</td>
<td>* Instruction</td>
</tr>
</tbody>
</table>

\[
\text{Energy} \propto \alpha \text{Instructions} \times C \times V^2
\]

“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

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

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

Performance = Power * Energy Efficiency
(Tasks/Second) (Joules/Second) (Tasks/Joule)

• 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
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

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

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

[Graph showing trends in transistors, frequency, power, and cores over time from 1970 to 2011.]

[Olukotun, Hammond, Sutter, Smith, Batten] 39
Break!
Administrivia

• Project 3.1 released Tuesday night, due next Wednesday (3/7).
• Homework 2 due Friday
• Project party on Wednesday 7-10
• Guerrilla session Thursday 7-9pm
• Midterm 2, March 20, is moved to 8-10PM (was 7-9 on the website)
  • Alternative exam earlier, 6-8PM (so people don’t need to be in exams until midnight :)
  • submit exam conflict form if they haven’t
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](image1) ![Year 2](image2) ![Year 3](image3) ![Year 4](image4)

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

• **Option 2:** *pipelining*
  - 1 year
  - 4000 graduate
  - 4000 graduate
  - 4000 graduate
  - 4000 graduate
  - 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 does nothing to latency
    - sometimes worse (additional overhead e.g. for shift transition)
Simultaneous versus Sequential

- What happens *sequentially*?
- What happens *simultaneously*?
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

<table>
<thead>
<tr>
<th>Phase</th>
<th>Pictogram</th>
<th>$t_{\text{step\ Serial}}$</th>
<th>$t_{\text{cycle\ Pipelined}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Reg Read</td>
<td></td>
<td>100 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>ALU</td>
<td></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td>200 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>Register Write</td>
<td></td>
<td>100 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>$t_{\text{instruction}}$</td>
<td></td>
<td>800 ps</td>
<td>1000 ps</td>
</tr>
</tbody>
</table>

- $t_{\text{instruction}}$: Instruction sequence
- $t_{\text{cycle}}$: Cycle
- Instruction sequence:
  - add t0, t1, t2
  - or t3, t4, t5
  - sll t6, t0, t3
Pipelining with RISC-V

Instruction sequence:
- add t0, t1, t2
- or t3, t4, t5
- sll t6, t0, t3

Timing:
- Single Cycle: $t_{step} = 100 \ldots 200$ ps
- Pipelining: $t_{cycle} = 200$ ps

Register access only 100 ps
All cycles same length

Instruction time, $t_{instruction}$:
- Single Cycle: $= t_{cycle} = 800$ ps
- Pipelining: $1000$ ps

Clock rate, $f_s$:
- Single Cycle: $1/800$ ps $= 1.25$ GHz
- Pipelining: $1/200$ ps $= 5$ GHz

Relative speed:
- Single Cycle: $1 \times$
- Pipelining: $4 \times$
Sequential vs Simultaneous

What happens sequentially, what happens simultaneously?

Instruction sequence

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

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

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

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