Great Ideas in Computer Architecture

RISC-V CPU Control, Pipelining

Instructor: Nick Riasanovsky
Agenda

- Datapath Review
- Control Implementation
- Administrivia
- Performance Analysis
- Pipelined Execution
- Pipelined Datapath
“Upper Immediate” instructions

- 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 (add to zero)
  - AUIPC – Add Upper Immediate to PC
Implementing **lui**

**Diagram Description:**
- **PC:** Program Counter
- **MEM:** Memory
- **alu:** ALU
- **dmem:** Direct Memory
- **wb:** Write Back
- ** Reg[]:** Register File
- **AddrD, AddrA, AddrB:** Address Inputs
- **DataD, DataA, DataB:** Data Inputs
- **inst[11:7], inst[19:15], inst[24:20], inst[31:7]:** Instruction Inputs
- **imm[31:0]:** Immediate Value
- **Branch Comp.:** Branch Comparator
- **Reg[rs1], Reg[rs2]:** Register Inputs
- **alu** Selects操作
- **MemRW = Read, MemRW = Write:** Memory Read/Write
- **WBSel = 1:** Write Back Selection
- **RegWEn = 1:** Register Write Enable
- **BrUn = *, BrE = *, BrLT = *:** Branch Conditions
- **InstSel = U:** Instruction Selection

**Equations:**
- \( \text{PCSel} = \text{pc} + 4 \)
- \( \text{inst}[31:0] \)
- \( \text{ImmSel} = \text{U}, \text{RegWEn} = 1 \)
- \( \text{Bsel} = 1, \text{Asel} = *, \text{ALUSel} = \text{B}, \text{MemRW} = \text{Read}, \text{WBSel} = 1 \)
Implementing auipc
All Immediates

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>25</td>
<td>24</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>15</td>
<td>14</td>
<td>12</td>
<td>11</td>
</tr>
<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>
</tr>
</tbody>
</table>

R-type

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

I-type

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

S-type

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

SB-type

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

U-type

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

UJ-type

Figure 2.3: RISC-V base instruction formats showing immediate variants.
Single-Cycle RISC-V RV32I Datapath
Single-Cycle RISC-V RV32I Datapath
Question: Which statement is TRUE about our RV32I ISA?

(A) When not in use, parts of the datapath cease to carry a value.
(B) Adding the instruction `lbu` will not change the datapath.
(C) All control signals will be don’t care (‘X’) in at least one instruction.
(D) Adding the instruction `bge` will not change the datapath.
**Question:** Which statement is TRUE about the RV32I ISA?

(A) When not in use, parts of the datapath cease to carry a value.

(B) Adding the instruction `lbu` will not change the datapath.

(C) All control signals will be don’t care (‘X’) in at least one instruction.

(D) Adding the instruction `bge` will not change the datapath.
Agenda

• Quick Datapath Review
• Control Implementation
• Administrivia
• Performance Analysis
• Pipelined Execution
• Pipelined 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>RegWE</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>l</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>l</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>
RV32I, a nine-bit ISA!

Instruction type encoded using only 9 bits

| imm[31:12] | rd | 0110111 |
| imm[31:12] | rd | 0010111 |

| imm[11:0] | rs1 | 000 | rd | 1101011 |
| imm[11:0] | rs2 | 010 | rd | 0000011 |
| imm[11:0] | rs1 | 100 | rd | 0000011 |
| imm[11:0] | rs2 | 000 | rd | 0000011 |
| imm[11:0] | rs2 | 011 | rd | 0010011 |
| imm[11:0] | rs2 | 100 | rd | 0010011 |
| imm[11:0] | rs1 | 110 | rd | 0010011 |
| imm[11:0] | rs1 | 111 | rd | 0010011 |

LUI
AUIPC
JAL
JALR
BEQ
BNE
BLT
BGE
BLTUI
BGEU
LB
LH
LW
LB
LHU
SB
SH
SW
ADDI
SLTI
SLTIU
XORI
ORI
ANDI

Not in CS61C
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-based Control

11-bit address (inputs)

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

15 data bits (outputs)

- PCSel
- ImmSel[2:0]
- BrUn
- ASel
- BSel
- ALUSel[3:0]
- MemRW
- RegWEn
- WBSel[1:0]
Single-Cycle RISC-V RV32I Datapath

Control Logic

PCSel | inst[31:0] | ImmSel | RegWEn | BrUn | BrEq | BrLT | BSel | ASel | ALUSel | MemRW | WBSel
ROM Controller Implementation

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

Controller output (PCSel, ImmSel, ...)

Address Decoder

Inst[]
BrEQ
BrLT

11
Agenda

• Quick Datapath Review
• Control Implementation
• **Administrivia**
• Performance Analysis
• Pipelined Execution
• Pipelined Datapath
Administrivia

• Regrade requests are due tonight
• Homework 3/4 due 7/16! (NOT 7/13, oops)
• Project 2-2 due Friday
• Project 3 released on Thurs, will rely on lab 6, so make sure you’re caught up on labs!
• Guerilla session tomorrow night 7/11
• HW Grades
  – Make sure edx/instructional account emails match!
Agenda

• Quick Datapath Review
• Control Implementation
• Administrivia
• Performance Analysis
• Pipelined Execution
• Pipelined Datapath
<table>
<thead>
<tr>
<th>Phase</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></td>
<td></td>
</tr>
<tr>
<td></td>
<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>

7/10/2018
Instruction Timing

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

- Most blocks idle most of the time
  - E.g. \( f_{\text{max, ALU}} = \frac{1}{200\text{ps}} = 5 \text{ GHz}! \)
  - 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!

### Instruction Timing Table

<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></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></td>
<td>X</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>700ps</td>
</tr>
</tbody>
</table>

7/10/2018 CS61C Su18 - Lecture 12
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>Passenger Capacity</td>
<td>2</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>Sports 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>750 min</td>
<td>120 min</td>
</tr>
<tr>
<td>Gallons per passenger</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 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 battery charge or 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

\[
\text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \text{Cycle}
\]
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
- Processor implementation (or *microarchitecture*)
- E.g. for “our” single-cycle RISC-V design, CPI = 1
- Complex instructions (e.g. `strcpy`), 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. transistor size)
- 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

Energy per Task

Energy = Instructions * Energy
Program Program Instruction

Energy α Instructions * C V^2
Program Program

“Capacitance” depends on technology, processor features e.g. # of cores

Supply voltage, e.g. 1V

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

• “Next-generation” processor
  - C (Moore’s Law): -15 %
  - Supply voltage, $V_{sup}$: -15 %
  - Energy consumption: $1 - (1-0.85)^3 = -39 \%$

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

• We will cover Moore’s Law later in the course
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
Agenda

• Quick Datapath Review
• Control Implementation
• Administrivia
• Performance Analysis
  • Pipelined Execution
• Pipelined Datapath
Pipeline Analogy: Doing Laundry

• Damon, Emaan, Nick, and Steven each have one load of clothes to wash, dry, fold, and put away
  – Washer takes 30 minutes
  – Dryer takes 30 minutes
  – “Folder” takes 30 minutes
  – “Stasher” takes 30 minutes to put clothes into drawers
• Sequential laundry takes 8 hours for 4 loads
Pipelined Laundry

- Pipelined laundry takes 3.5 hours for 4 loads!
Pipelining Lessons (1/2)

- Pipelining doesn’t help *latency* of single task, just *throughput* of entire workload
- *Multiple* tasks operating simultaneously using different resources
- Potential speedup = number of pipeline stages
- Speedup reduced by time to *fill* and *drain* the pipeline: 8 hours/3.5 hours or 2.3X v. potential 4X in this example
• Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
  – Pipeline rate limited by *slowest* pipeline stage
  – Unbalanced lengths of pipeline stages reduces speedup
Agenda

• Quick Datapath Review
• Control Implementation
• Administrivia
• Performance Analysis
• Pipelined Execution
  • Pipelined Datapath
### 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>

- **Instruction sequence**
  - add $t_0$, $t_1$, $t_2$
  - or $t_3$, $t_4$, $t_5$
  - sll $t_6$, $t_0$, $t_3$

---

*7/10/2018*
Pipeline Performance

• Use $T_c$ ("time between completion of instructions") to measure speedup
  
  $T_{c,pipelined} \geq \frac{T_{c,single-cycle}}{\text{Number of stages}}$

  – Equality only achieved if stages are balanced
    (i.e. take the same amount of time)

• If not balanced, speedup is reduced

• Speedup due to increased throughput
  
  – Latency for each instruction does not decrease
Pipelining with RISC-V

- \( \text{add } t_0, t_1, t_2 \)
- \( \text{or } t_3, t_4, t_5 \)
- \( \text{sll } t_6, t_0, t_3 \)

### Single Cycle vs. Pipelining

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

\( t_{\text{cycle}} \) represents the cycle time, \( t_{\text{step}} \) represents the step time, and \( t_{\text{instruction}} \) represents the instruction time.
Sequential vs Simultaneous

What happens sequentially, what happens simultaneously?

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

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

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)

addi t2, t2, 1

Instruction sequence

7/10/2018
Instruction Level Parallelism (ILP)

• Pipelining allows us to execute parts of multiple instructions at the same time using the same hardware!
  – This is known as *instruction level parallelism*

• Later: Other types of parallelism
  – DLP: same operation on lots of data (SIMD)
  – TLP: executing multiple threads “simultaneously” (OpenMP)
Pipelined Control

• Control signals derived from instruction
  – As in single-cycle implementation
  – Information is stored in pipeline registers for use by later stages
**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a **4-stage** pipelined version.

<table>
<thead>
<tr>
<th>Instr Fetch</th>
<th>Reg Read</th>
<th>ALU Op</th>
<th>Mem Access</th>
<th>Reg Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>

1) The *latency* will be 1.25x slower.
2) The *throughput* will be 3x faster.

(A) F F 
(B) F T 
(C) T F 
(D) T T
**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a 4-stage pipelined version.

<table>
<thead>
<tr>
<th>Instr Fetch</th>
<th>Reg Read</th>
<th>ALU Op</th>
<th>Mem Access</th>
<th>Reg Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>

1) The *latency* will be 1.25x slower.
2) The *throughput* will be 3x faster.

**No mem access**

**throughput:**

\[
1/(IF+ID+EX+WB) = 1/600 \rightarrow 4/(4*\text{max\_stage}) = 1/200
\]

\[
1/200*600/1 = 3x \text{ faster}
\]
Question: Assume the stage times shown below. Suppose we remove loads and stores from our ISA. Consider going from a single-cycle implementation to a 4-stage pipelined version.

<table>
<thead>
<tr>
<th>Instr Fetch</th>
<th>Reg Read</th>
<th>ALU Op</th>
<th>Mem Access</th>
<th>Reg Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>

1) The *latency* will be 1.25x slower.
2) The *throughput* will be 3x faster.

No mem access

latency:
IF+ID+EX+WB = 600 → 4*max_stage = 800
800/600 = 1.33x slower!
Question: Assume the stage times shown below. Suppose we remove loads and stores from our ISA. Consider going from a single-cycle implementation to a 4-stage pipelined version.

<table>
<thead>
<tr>
<th>Instr Fetch</th>
<th>Reg Read</th>
<th>ALU Op</th>
<th>Mem Access</th>
<th>Reg Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>

1) The **latency** will be 1.25x slower.
2) The **throughput** will be 3x faster.

(A) F F F
(B) F T T
(C) T F F
(D) T T T
Summary

• Implementing controller for your datapath
  – Take decoded signals from instruction and generate control signals

• Pipelining improves performance by exploiting Instruction Level Parallelism
  – 5-stage pipeline for RV32I: IF, ID, EX, MEM, WB
  – Executes multiple instructions in parallel
  – Each instruction has the same latency
  – Be careful of signal passing (more on this next lecture)