18-447 Lecture 3: Single-Cycle Implementation

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - first try at implementing the RV32I ISA (on paper)
- Notices
  - Handout #4: HW1, due 2/8
  - Student survey on Blackboard, past due
  - Lab 1, Part A, due week of 1/30
  - Lab 1, Part B, due week of 2/6
- Readings
  - P&H Ch 4.1~4.4
  - finish P&H Ch 2 for next time
Lab and Office Hours

- James Hoe, MW 2:30~3:30, HH-A304
- TA office hours in lab (HH-1305)
  - Tuesday: [9~10:30 – Steven]
  - Tuesday: [10:30~12 – Brandon]
  - Tuesday: [12~1:30 – Steven]
  - Tuesday: [1:30~3 – Neil]
  - Thursday: [12:30~1:30 – Steven]
  - Thursday: [1:30~3 – Brandon + Steven]
  - Thursday: [3~4:30 – Amanda + Brandon]
  - Sunday: [1~3:30 – Amanda]

Instruction Processing FSM

- An ISA describes an abstract FSM
  - state = program visible state
  - next-state logic = instruction execution
- Nice ISAs have atomic instruction semantics
  - one state transition per instruction in abstract FSM
- The implementation FSM can look wildly different
Program Visible State
(aka Architectural State)

“Magic” Memory and Register File

- **Combinational Read**
  - output of the read data port is a combinational function of the register file contents and the corresponding read select port

- **Synchronous write**
  - the selected register (or memory location) is updated on the posedge clock transition when write enable is asserted
  Cannot affect read output in between clock edges
Characteristics of “RISC” (including RISC-V)

- Simple operations
  - 2-input, 1-output arithmetic and logical operations
  - few alternatives for accomplishing the same thing
- Simple data movements
  - ALU ops are register-to-register
  - “load-store” architecture, 1 addressing mode
- Simple branches
  - limited varieties of branch conditions and targets
  - PC-offset
- Simple instruction encoding
  - all instructions encoded in the same number of bits
  - simple, fixed formats

(RISC=Reduced Instruction Set Computer)

RISC Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access (not required by non-mem instructions)
  - write-back

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Single-Cycle Datapath for RV32I ALU Instructions

- Assembly (e.g., register-register addition)
  ADD rd, rs1, rs2
- Machine encoding
  
<table>
<thead>
<tr>
<th>0000000</th>
<th>rs2</th>
<th>rs1</th>
<th>000</th>
<th>rd</th>
<th>0110011</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-bit</td>
<td>5-bit</td>
<td>5-bit</td>
<td>3-bit</td>
<td>5-bit</td>
<td>7-bit</td>
</tr>
</tbody>
</table>

- Semantics
  - GPR[rd] ← GPR[rs1] + GPR[rs2]
  - PC ← PC + 4
- Exceptions: none (ignore carry and overflow)
- Variations
  - Arithmetic: {ADD, SUB}
  - Compare: {signed, unsigned} x {Set if Less Than}
  - Logical: {AND, OR, XOR}
  - Shift: {Left, Right-Logical, Right-Arithmetic}

Register-Register ALU Instructions
ADD rd rs1 rs2

if MEM[PC] == ADD rd rs1 rs2
GPR[rd] ← GPR[rs1] + GPR[rs2]
PC ← PC + 4

R-Type ALU Datapath
Reg-Immediate ALU Instructions

- Assembly (e.g., reg-immediate additions)
  ADDI rd, rs1, imm_{12}
- Machine encoding
  \[
  \begin{array}{cccccc}
  \text{imm[11:0]} & \text{rs1} & 000 & \text{rd} & 0010011 \\
  \text{12-bit} & \text{5-bit} & \text{3-bit} & \text{5-bit} & \text{7-bit} \\
  \end{array}
  \]
- Semantics
  - GPR[rd] ← GPR[rs1] + sign-extend (imm)
  - PC ← PC + 4
- Exceptions: none (ignore carry and overflow)
- Variations
  - Arithmetic: \{ADDI, SUBI\}
  - Compare: \{signed, unsigned\} x \{Set if Less Than Immediate\}
  - Logical: \{ANDI, ORI, XORI\}
  - **Shifts by unsigned imm[4:0]: \{SLLI, SRLI, SRAI\}

ADDI rd rs1 immediate_{12}

Combinational state update logic
Datapath for R and I-type ALU Inst’s
Load Instructions

- Assembly (e.g., load 4-byte word)
  \( \text{LW } rd, \text{ offset}_{12}(\text{base}) \)
- Machine encoding
  \[
  \begin{array}{cccc|c}
  \text{offset}[11:0] & \text{base} & 010 & \text{rd} & \text{0000011} \\
  \multicolumn{5}{c}{\text{12-bit}} \\
  \multicolumn{5}{c}{\text{5-bit}} \\
  \multicolumn{5}{c}{\text{3-bit}} \\
  \multicolumn{5}{c}{\text{5-bit}} \\
  \multicolumn{5}{c}{\text{7-bit}} \\
  \end{array}
  \]
- Semantics
  - \( \text{byte\_address}_{32} = \text{sign-extend}(\text{offset}_{12}) + \text{GPR}[\text{base}] \)
  - \( \text{GPR}[\text{rd}] \leftarrow \text{MEM}_{32}[\text{byte\_address}] \)
  - \( \text{PC} \leftarrow \text{PC} + 4 \)
- Exceptions: none for now
- Variations: LW, LH, LHU, LB, LBU
  e.g., LB :: \( \text{GPR}[\text{rd}] \leftarrow \text{sign-extend}(\text{MEM}_8[\text{byte\_address}]) \)
  LBU :: \( \text{GPR}[\text{rd}] \leftarrow \text{zero-extend}(\text{MEM}_8[\text{byte\_address}]) \)

Note: RV32I memory is byte-addressable, little-endian

LW Datapath

if \( \text{MEM}[\text{PC}] = \text{LW } \text{rd, offset}_{12}(\text{base}) \)
\( \text{EA} = \text{sign-extend}(\text{offset}) + \text{GPR}[\text{base}] \)
\( \text{GPR}[\text{rd}] \leftarrow \text{MEM}[\text{EA}] \)
\( \text{PC} \leftarrow \text{PC} + 4 \)

Combinational state update logic
Store Instructions

- Assembly (e.g., store 4-byte word)
  \[ \text{SW } rs2, \text{ offset}_{12}(\text{base}) \]
- Machine encoding
  \[
  \begin{array}{|c|c|c|c|c|c|}
  \hline
  \text{offset}_{11:5} & \text{rs2} & \text{base} & \text{010} & \text{0fst}_{4:0} & \text{0100011} \\
  \hline
  \end{array}
  \]
- Semantics
  - \[ \text{byte_address}_{32} = \text{sign-extend}(\text{offset}_{12}) + \text{GPR}[\text{base}] \]
  - \[ \text{MEM}_{32}[\text{byte_address}] \leftarrow \text{GPR}[\text{rs2}] \]
  - \[ \text{PC} \leftarrow \text{PC} + 4 \]
- Exceptions: none for now
- Variations: SW, SH, SB
  e.g., SB:: \[ \text{MEM}_8[\text{byte_address}] \leftarrow (\text{GPR}[\text{rs2}])[7:0] \]

SW Datapath

if \( \text{MEM}[\text{PC}] = \text{SW } rs2, \text{ offset}_{12}(\text{base}) \)
\[ \text{EA} = \text{sign-extend}(\text{offset}) + \text{GPR}[\text{base}] \]
\[ \text{MEM}[\text{EA}] \leftarrow \text{GPR}[\text{rs2}] \]
\[ \text{PC} \leftarrow \text{PC} + 4 \]
Load-Store Datapath

Datapath for Non-Control Flow Inst’s
Single-Cycle Datapath for Control Flow Instructions

Jump Instruction

- Assembly
  JAL rd imm$_{21}$
  Note: implied imm[0] (always 0)

- Machine encoding
  \[
  \begin{array}{c|c|c|c|c}
  \text{imm}[20:10|11:19:12] & \text{rd} & 1101111 & \text{UJ-type} \\
  \hline
  \text{20-bit} & \text{5-bit} & \text{7-bit} \\
  \end{array}
  \]

- Semantics
  - target = PC + sign-extend(imm$_{21}$)
  - GPR[rd] $\leftarrow$ PC + 4
  - PC $\leftarrow$ target

  How far can you jump?

- Exceptions: misaligned target (4-byte)

  *Note*: use “JAL x0 label” instead of “BEQ x0 x0 label”
Unconditional Jump Datapath

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

if MEM[PC] == JAL rd, immediate
GPR[rd] = PC + 4
PC = PC + sign-extend(immediate)

Conditional Branch Instructions

- Assembly (e.g., branch if equal)
  BEQ rs1, rs2, imm
  Note: implied imm[0] (always 0)

- Machine encoding
  | imm[12:10] | rs2 | rs1 | 000 | imm[4:1] | 1100011 |
  | 7-bit | 5-bit | 5-bit | 3-bit | 5-bit | 7-bit |

- Semantics
  - target = PC + sign-extend(imm)
  - if GPR[rs1] == GPR[rs2] then PC ← target
    else PC ← PC + 4

  How far can you jump?

- Exceptions: misaligned target (4-byte) if taken
- Variations
  - BEQ, BNE, BLT, BGE, BLTU, BGEU
Conditional Branch Datapath

JAL and taken Branch

Datapath Control Generation
Single-Bit Control Signals

<table>
<thead>
<tr>
<th>ALUSrc</th>
<th>When De-asserted</th>
<th>When asserted</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2nd ALU input from 2nd GPR read port</td>
<td>2nd ALU input from immediate</td>
<td>(opcode==IsRtype)</td>
</tr>
</tbody>
</table>
| RegWrite     | GPR write disabled | GPR write enabled | (opcode!=SW) &&
|              |                   |               | (opcode!=Bxx)          |
| MemtoReg     | Steer ALU result to GPR write port | steer memory load to GPR write port | opcode==LW/H/B       |
| PctoReg      | Steer above result to GPR write port | Steer PC+4 to GPR write port | (opcode==JAL) || (opcode==JALR) |
| MemRead      | Memory read disabled | Memory read port return load value | opcode==LW/H/B       |
| MemWrite     | Memory write disabled | Memory write enabled | opcode==SW/H/B        |

Multi-Bit Control Signals

<table>
<thead>
<tr>
<th>Options</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU Op</td>
<td>case opcode</td>
</tr>
<tr>
<td></td>
<td>RTypeALU: according to funct3, funct7[5]</td>
</tr>
<tr>
<td></td>
<td>ITypeALU : according to funct3 only (except shift)</td>
</tr>
<tr>
<td></td>
<td>LW/SW/JALR : ADD</td>
</tr>
<tr>
<td></td>
<td>Bxx : SUB and select bcond function</td>
</tr>
<tr>
<td></td>
<td>__ : pass through 2nd</td>
</tr>
<tr>
<td>ImmExtend</td>
<td>• select based on instruction format type</td>
</tr>
<tr>
<td></td>
<td>• (may want to have separate extension units for primary ALU and PC-offset adder)</td>
</tr>
<tr>
<td>PCSrc</td>
<td>case opcode</td>
</tr>
<tr>
<td></td>
<td>JAL : PC + immediate</td>
</tr>
<tr>
<td></td>
<td>JALR : GPR + immediate</td>
</tr>
<tr>
<td></td>
<td>Bxx : taken?(PC + immediate):(PC + 4)</td>
</tr>
<tr>
<td></td>
<td>__ : PC+4</td>
</tr>
<tr>
<td>LoadExtend</td>
<td>case func3</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
</tbody>
</table>
Architecture vs Microarchitecture

- **Functional Level**
  - A clock has a hour hand and a minute hand, ..... 
  - A computer does ....????....
  - You can read a clock without knowing how one works

- **Implementation Level**
  - A particular clock design has a certain set of gears
  - arranged in a certain configuration
  - A particular computer design has a certain datapath
  - and a certain control logic

- **Realization Level**
  - Machined alloy gears versus stamped sheet metal gears
  - CMOS versus ECL versus vacuum tube

[Computer Architecture, Blaauw and Brooks, 1997]