CIS 429/529

Single-cycle

ISA Implementation

Prof. Michel A. Kinsky
Instruction Set Architecture (ISA)

- ISA is the hardware/software interface
  - Defines data types
  - Defines set of programmer visible state
  - Defines instruction semantics (operations, sequencing)
  - Defines instruction format (bit encoding)
  - Examples: MIPS, Alpha, x86, IBM 360, VAX, ARM, JVM
Instruction Set Architecture (ISA)

• Many possible implementations of one ISA
  ‣ 360 implementations: model 30 (c. 1964), z900 (c. 2001)
  ‣ x86 implementations: 8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC
  ‣ MIPS implementations: R2000, R4000, R10000, ...
  ‣ JVM: HotSpot, PicoJava, ARM Jazelle, ...
Processor Performance

• Instructions per program depends on source code, compiler technology and ISA

• Cycles per instructions (CPI) depends upon the ISA and the microarchitecture

• Time per cycle depends upon the microarchitecture and the base technology

\[
\text{Time} = \frac{\text{Instructions}}{\text{Cycles}} \times \frac{\text{Instruction}}{\text{Cycle}}
\]
Processor Performance

- Instructions per program depends on source code, compiler technology and ISA.
- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture.
- Time per cycle depends upon the microarchitecture and the base technology.

<table>
<thead>
<tr>
<th>Microarchitecture</th>
<th>CPI</th>
<th>cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcoded</td>
<td>&gt;1</td>
<td>short</td>
</tr>
<tr>
<td><strong>Single-cycle unpipelined</strong></td>
<td>1</td>
<td>long</td>
</tr>
<tr>
<td>Pipelined</td>
<td>1</td>
<td>short</td>
</tr>
</tbody>
</table>
The MIPS ISA

• Processor State
  ▸ 32 32-bit GPRs, R0 always contains a 0
  ▸ 32 single precision FPRs, may also be viewed as
  ▸ 16 double precision FPRs
  ▸ FP status register, used for FP compares & exceptions
  ▸ PC, the program counter
  ▸ some other special registers
The MIPS ISA

• Data types
  ‣ 8-bit byte, 16-bit half word
  ‣ 32-bit word for integers
  ‣ 32-bit word for single precision floating point
  ‣ 64-bit word for double precision floating point
The MIPS ISA

• Load/Store style instruction set
  ‣ data addressing modes- immediate & indexed
  ‣ branch addressing modes- PC relative & register indirect
  ‣ Byte addressable memory- big endian mode

• All instructions are 32 bits
Instruction Execution

- Execution of an instruction involves
  1. Instruction fetch
  2. Decode
  3. Register fetch
  4. ALU operation
  5. Memory operation (optional)
  6. Write back
  7. Computation of the address of the next instruction
Datapath: Reg-Reg ALU Instructions

```
<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>5</th>
<th>0</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

rd ← (rs) func (rt)
Datapath: Reg-Imm ALU Instructions

Opcode

0x4
Add

clk
addr
inst
Inst. Memory

inst<25:21>
inst<20:16>
inst<15:0>
inst<31:26>

RegWrite
clk

rt ← (rs) op immediate

alu
RegWrite

rd1
rd2
GPRs
we
rs1
rs2
ws
wd

Imm Ext

ExtSel

ALU Control

OpCode

6 5 5 16
31 26 25 21 20 16 15 0

Computer Architecture and Embedded Systems Laboratory (CAES Lab)
Conflicts in Merging Datapath

Computer Architecture and Embedded Systems Laboratory (CAES Lab)
Conflicts in Merging Datapath

```
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Add</th>
<th>clk</th>
<th>0x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs</td>
<td>rt</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>rt</td>
<td>rd</td>
<td>rd</td>
<td>6</td>
</tr>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>25</td>
</tr>
</tbody>
</table>
```

rd ← (rs) func (rt)
rt ← (rs) op immediate
Datapath for Memory Instructions

• Should program and data memory be separate?
  ‣ Harvard style: separate (Aiken and Mark I influence)
    • read-only program memory
    • read/write data memory
  ‣ Princeton style: the same (von Neumann’s influence)
    • single read/write memory for program and data
      • Executing a Load or Store instruction requires accessing the memory more than once
Harvard Architecture

![Harvard Architecture Diagram](image)

### Harvard Architecture Diagram

- **Add**: Operates with 0x4
- **ALU**: Performs logical operations
- **MemWrite**: Writes data to memory
- **WBSrc**: Selects the source for write-back
- **Inst. Memory**: Stores instructions
- **GPRs**: General Purpose Registers
- **RegWrite**: Writes to registers

### Addressing Mode

- **Opcode**: Determines the type of operation
- **Rs**: Source register
- **Rt**: Target register
- **Displacement**: Offset from base

### Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>6</td>
</tr>
<tr>
<td>Rs</td>
<td>5</td>
</tr>
<tr>
<td>Rt</td>
<td>5</td>
</tr>
<tr>
<td>Displacement</td>
<td>16</td>
</tr>
</tbody>
</table>

**Addressing Mode**: `(Rs) + Displacement`
MIPS Control Instructions

- **Conditional (on GPR) PC-relative branch**

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

  BEQZ, BNEZ

- **Unconditional register-indirect jumps**

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

  JR, JALR

- **Unconditional absolute jumps**

<table>
<thead>
<tr>
<th>opcode</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>

  J, JAL

- **PC-relative branches add offset*4 to PC+4 to calculate the target address (offset is in words): ±128 KB range**
MIPS Control Instructions

- Absolute jumps append target×4 to PC<31:28> to calculate the target address: 256 MB range
- jump-&-link stores PC+4 into the link register (R31)
- All Control Transfers are delayed by 1 instruction
- We will worry about the branch delay slot in later lectures
Conditional Branches (BEQZ, BNEZ)

Diagram showing the control flow and data paths for a conditional branch instruction in a computer architecture context. The diagram includes symbols for buses, registers, and ALU operations, indicating the flow of data and control signals for branch instructions.
Register-Indirect Jumps (JR)
Register-Indirect Jump-&-Link (JALR)

- PCSrc: br
- rind: pc+4
- Add: 0x4
- PC: inst
- Addr: Memory
- Inst. Memory: clk
- Alu: Imm Ext
- ALU: we rs1 rs2 rd1 ws wd rd2 GPRs
- ALU Control: z
- Add: clk
- RegWrite: RegDst ExtSel OpSel BSrc zero? we addr rdata wdata
- MemWrite: we addr
- WBSrc: clk
Absolute Jumps (J, JAL)
Harvard-Style Datapath for MIPS
Hardwired Control

- Hardwired Control is pure Combinational Logic

```
| ExtSel | BSrc | OpSel | MemWrite | WBSrc | RegDst | RegWrite | PCSrc |
```

Diagram:
- op code
- zero?
ALU Control & Immediate Extension

Inst<31:26> (Opcode)
Inst<5:0> (Func)

ALUop

OpSel (Func, Op, +, 0?)

ExtSel (sExt_{16}, uExt_{16}, High_{16})

Decode Map

OpSel

+ 0?
Single-Cycle Hardwired Control

• Harvard architecture: we will assume that

  ‣ clock period is sufficiently long for all of and the following steps to be “completed”:

    1. instruction fetch
    2. decode and register fetch
    3. ALU operation
    4. data fetch if required
    5. register write-back setup time

  ‣ \( t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB} \)
Two-State Controller

• In the Princeton Microarchitecture, a flipflop can be used to remember the phase

  fetch phase

  execute phase

- AddrSrc=PC
  - IRen=on
  - PCen=off
  - Wen=off

- AddrSrc=ALU
  - IRen=off
  - PCen=on
  - Wen=on
Hardwired Controller

**Old combinational logic (Harvard)**

- IR
- op code
- zero?
- ExtSel, BSrc, OpSel, WBSrc, RegDest, PCsrc1, PCsrc2
- MemWrite
- RegWrite
- Wen
  - Pcen
  - Iren
  - AddrSrc

**New combinational logic**

- 1-bit Toggle FF
- I-fetch / Execute
Clock Period

- Princeton architecture
  - \( t_{C-Princeton} > \max \{ t_M, t_{RF} + t_{ALU} + t_M + t_{WB} \} \)
  - \( t_{C-Princeton} > t_{RF} + t_{ALU} + t_M + t_{WB} \)

- while in the hardwired Harvard architecture
  - \( t_{C-Harvard} > t_M + t_{RF} + t_{ALU} + t_M + t_{WB} \)

- which will execute instructions faster?
Clock Rate vs CPI

- Suppose $t_M >> t_{RF} + t_{ALU} + t_{WB}$
  - $t_{C-Princeton} = 0.5 * t_{C-Harvard}$

- $\text{CPI}_{Princeton} = 2$
- $\text{CPI}_{Harvard} = 1$

- No difference in performance!
Can we overlap instruction fetch and execute?
• Only one of the phases is active in any cycle
  ‣ a lot of datapath is not in use at any given time

The same
(mux not shown)
Stalling the instruction fetch

- When stall condition is indicated
  - don’t fetch a new instruction and don’t change the PC
  - insert a nop in the IR
  - set the Memory Address mux to ALU (not shown)
Need to stall on branches

- When IR contains a jump or branch-taken
  - no “structural conflict” for the memory
  - but we do not have the correct PC value in the PC
Need to stall on branches

- Memory cannot be used – Address Mux setting is irrelevant
- Insert a nop in the IR
- Insert the nextPC (branch-target) address in the PC
Pipelined Princeton Architecture

- Clock: $t_{C-Princeton} > t_{RF} + t_{ALU} + t_{M}$

- CPI: $(1 - f) + 2f$ cycles per instruction
  where $f$ is the fraction of instructions that cause a stall
Next Class

• Instruction Pipelining and Hazards