Digital Logic Design
234262
Alex Bronstein

Lecture 5: MIPS Instruction Set Architecture
Abstraction Hierarchy

Application Software

Operating System *(threads, files, exceptions)*

Computer Architecture *(instruction set)*

Micro-Architecture *(execution pipeline)*

Logic *(adders, multipliers, FSMs)*

Digital Circuits *(gates)*

Analog Circuits *(amplifiers)*

Process and Devices *(MOS FET transistors)*

Physics *(electrons, holes, tunneling, band gap)*
Controllers discussed thus far were **special purpose**

**Processor** (aka **CPU**) is programmable hardware that can implement various functions (Turing complete)
RISC vs. CISC

- **Reduced Instruction Set Computer (RISC)**
  - Small structured and highly optimized instruction set
  - Simple encoding, few addressing modes, load/store architecture
  - More complicated compiler
  - More efficient hardware
  - Examples: ARM, MIPS, PowerPC, DEC Alpha, SPARC, Blackfin

- **Complex Instruction Set Computer (CISC)**
  - Complicated and heterogenous instruction set
  - More cumbersome hardware
  - Examples: Intel x86, DEC PDP11/VAX
The following interface is exposed to software:

- 32-bit bit instructions of 3 types: I, R and J
- 32 general purpose 32-bit integer registers
- 32 general purpose single-precision floating point registers
  (paired into 16 double-precision registers)
- 2 special registers (Lo/Hi) to store results of multiplication/division
  (not directly addressable)
- 32-bit addressable data memory
- 32-bit addressable instruction memory addressed by the dedicated
  32-bit program counter register (PC)
Toolchain

High-level program (C)

void swap(int v[], int k) {
    int temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}

Assembly language (MIPS)

swap:
sll $t0, $a1, 2
add $t0, $a0, $t0
lw $t1, 0($t0)
lw $t2, 4($t0)
sw $t1, 4($t0)
sw $t2, 0($t0)
jr $ra

Machine code (MIPS)

00054080 00884020
8d090000 8d0a0004
ad090004 ad0a0000
00000008
MIPS Memory

- Registers
  - ×32 32-bit (4-byte) integer registers
  - 32-bit program counter (PC, not directly addressable)
  - (More stuff: FP registers, special Lo/Hi registers)

- Data memory
  - $2^{30}$ addressable 32-bit word space
  - Each word address has two LSBs = 0

- Instruction memory
  - Each instruction is encoded in exactly 32-bit (no variable length)
  - $2^{30}$ word instruction memory space
  - Memory is often practically implemented on the same physical RAM as modified Harvard architecture
MIPS Registers

Except for \$0, all registers have the same status. The following convention is common for cross-compatibility of assemblers and compilers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Alias</th>
<th>Conventional use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>zero</td>
<td>value 0</td>
</tr>
<tr>
<td>$1</td>
<td>$a\text{t}$</td>
<td>assembler temporary reserved by the assembler</td>
</tr>
<tr>
<td>$2–$3</td>
<td>$v0–v1$</td>
<td>values from expression evaluation &amp; function results</td>
</tr>
<tr>
<td>$4–$7</td>
<td>$a0–a3$</td>
<td>arguments – first four arguments of a function</td>
</tr>
<tr>
<td>$8–$15</td>
<td>$t0–t7$</td>
<td>temporaries</td>
</tr>
<tr>
<td>$16–$23</td>
<td>$s0–s7$</td>
<td>saved values</td>
</tr>
<tr>
<td>$24–$25</td>
<td>$t8–t9$</td>
<td>temporaries</td>
</tr>
<tr>
<td>$26–$27</td>
<td>$k0–k1$</td>
<td>reserved by interrupt handler</td>
</tr>
<tr>
<td>$28</td>
<td>$gp$</td>
<td>global pointer</td>
</tr>
<tr>
<td>$29</td>
<td>$sp$</td>
<td>stack pointer</td>
</tr>
<tr>
<td>$30</td>
<td>$s8/fp$</td>
<td>saved value/frame pointer</td>
</tr>
<tr>
<td>$31</td>
<td>$ra$</td>
<td>return address</td>
</tr>
</tbody>
</table>
MIPS Instruction Set Architecture (ISA)

- **Arithmetic/Logic** (works on registers only)
  - Integer
  - Floating point

- **Memory access** (moves data between registers and RAM)
  - Load/store

- **Control** (affects program counter)
  - Jump
  - Conditional branch
  - Function call/return
MIPS Instructions

- Instructions are read in order from address pointed by the PC and executed.
- Every instruction is encoded in exactly 32-bit (no exceptions).
- Every instruction in the ISA is characterized by:
  - **Syntax** – how the instruction is written in assembly language
  - **Semantics** – what the instruction does
  - **Binary coding** – the way the processor sees it
  - **Implementation-specific details** – cycle count, delay slots, other limitations
### MIPS Instruction Encoding

- **Register** (R) type instructions accept three registers
  
  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  | OP | s  | t  | d  | shamnt | func |
  | [5 bit] | [5 bit] | [5 bit] | [5 bit] | [5 bit] | [5 bit] |

- **Immediate** (I) type instructions accept two registers and one fixed value (immediate)
  
  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  | OP | s  | t  | IM |
  | [5 bit] | [5 bit] | [5 bit] | [16 bit] |

- **Jump** (J) type instructions receive a fixed (immediate) address
  
  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  | OP | XI |
  | [5 bit] | [26 bit] |
MIPS R-type Instruction Encoding

Integer addition

- **Mnemonics:** add rd, rs, rt
- **Semantics:** \( rd \leftarrow rs + rt \)
- **Encoding:**

\[
\begin{array}{cccccccccccccccccccccc}
0 & s & t & d & 0 & & & & & & & & & & & & & & & & & & & & & & 32_{10} \\
\end{array}
\]

\( s, t, d = 5\text{-bit register address (0..31) } \)
MIPS I-type Instruction Encoding

Integer add immediate

- Mnemonics: addi rt, rs, IM
- Semantics: rt ← rs+IM
- Encoding:

```
  31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  8_{10}  s  t    IM
```

s,t,d = 5-bit register address (0..31)
IM = 16-bit immediate value coded into instruction
MIPS I-type Instruction Encoding

Load word

- **Mnemonics:** lw rt, IM(rs)
- **Semantics:** rt ← MEM[rs+IM]
- **Encoding:**

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 35 | s | t | IM |

Store word

- **Mnemonics:** sw rt, IM(rs)
- **Semantics:** MEM[rs+IM] ← rt
- **Encoding:**

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 43 | s | t | IM |
MIPS I-type Instruction Encoding

Branch on equal

- Mnemonics: \( \text{beq } rs, rt, \text{ADDRESS} \)

- Semantics:
  \[
  \text{if } (rs==rt):
  \]
  \[
  \text{PC } \leftarrow \text{PC+}(IM\ll2)
  \]

- Encoding:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4_{10}</td>
<td>s</td>
<td>t</td>
<td>IM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MIPS J-type Instruction Encoding

Jump

- Mnemonics: \( j \) ADDR
- Semantics: \( PC \leftarrow (PC \& F0000000_{16}) | (XI \ll 2) \)
- Encoding:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8_{10} |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

\( XIM = 26 \)-bit extended immediate value containing new absolute address to jump to
MIPS Integer Arithmetic/Logic Instructions

- **Addition:**  
  - add (R) • addu (R) • addi (I) • addiu (I)

- **Subtraction:**  
  - sub (R) • subu (R) • subi (I) • subiu (I)

- **Logic:**  
  - or (R) • xor (R) • nor (R) • and (R)
  - ori (I) • xori (I) • nori (I) • andi (I)

- **Shift:**  
  - sll (R) • srl (R) • sra (R) • sllv (R) • srlv (R) • srav (R)

- **Other:**  
  - slt (R) • sltu (R) • slti (I) • sltui (I) • lui (I)

- **Mult/Div:**  
  - mult (R) • multu (R) • div (R) • divu (R)
MIPS Memory Access Instructions

- Load: \texttt{lw (R)} • \texttt{lb (R)}
- Store: \texttt{sw (R)} • \texttt{sb (R)}
MIPS Jump Instructions

- **Unconditional:**  
  - \( j \) (J) • \( jr \) (R) • \( jal \) (J) • \( jalr \) (R)

- **Conditional:**  
  - \( bltz \) (I) • \( bgez \) (I) • \( bltzal \) (I) • \( bgezal \) (I)
  - \( beq \) (I) • \( bne \) (I) • \( blez \) (I) • \( bgez \) (I)
MIPS Instructions

- Instruction reads at most two registers ($rs$, $rt$)
- Instruction modifies at most one register
  - R-type might change $rd$
  - I-type might change $rt$
- RAM-related instructions ($lw$/ $sw$) do not perform calculations
MIPS Datapath Building Blocks

- Register file
- Arithmetic-Logic Unit (ALU)
- Memory (MEM)
- Program Counter (PC)

These building blocks are present in all implementations.

Other implementations may have additional building blocks.
Abstract MIPS Datapath

Instruction Memory

Next PC

ALU

Next PC

ALU out

Data Memory
Abstract MIPS Datapath

add rd, rs, rt
Abstract MIPS Datapath

```
addi rd, rs, IM
```

Diagram showing the Abstract MIPS Datapath, including the instruction memory, register file, ALU, and data memory. The diagram illustrates the flow of data and address calculations for the `addi` instruction.
$lw \, rt, \, IM(rs)$
Abstract MIPS Datapath

\[
sw \ rt, \ IM(rs)
\]