Architectural models for computational operations

- Stack architecture
- Accumulator architecture
- General Purpose Register (GPR) architectures
  - Register-memory
  - Memory-memory
  - Register-register (load/store)

CMPE04 Winter 2004
MIPS Instruction Set Architectures (ISA)

- Definition and types of ISAs
- RISC principles
- MIPS ISA
- MIPS binary instruction format

Textbook: 3.15, 3.4

Instruction Set Architecture

Operands specified per computational instruction: 0
Operands are kept on a stack. Computational operations remove the operands from the stack and put the result onto the stack.
Example: \( A = B \times (C + D \times B) \)

1:  
2:  
3:  
4:  
5:  
6:  
7:  
8:  
9:  

Assembly language

Instruction Set Architecture

Hardware

Permits many different hw implementations to run identical sw.
**Instruction Set Architectures**

**GPR architecture: Memory-memory**

Operands per computational instruction: 3, all memory locations.
Example: \( A = B \times (C + D \times B) \)

1:
2:
3:
4:
5:
6:
7:

---

**Accumulator architecture**

Operands per computational instruction 1 (the other one is the accumulator by default). The result is put into the accumulator.
Example: \( A = B \times (C + D \times B) \)

1:
2:
3:
4:
5:

---

**GPR architecture: Register-register (load/store)**

Operands per computational instruction: 2 or 3. Operands are loaded from memory into registers. The Arithmetic/Logic Unit (ALU) only operates on registers.
Example: \( A = B \times (C + D \times B) \)

1:
2:
3:
4:
5:
6:
7:

---

**GPR architecture: Register-memory**

Operands per computational instruction 2, a register and a memory location. The result is put back into the source register.
Example: \( A = B \times (C + D \times B) \)

1:
2:
3:
4:
5:
RISC design principles

- If hardware can not implement a feature cheaply, it's often better to implement the feature in software.
- Simplicity favors regularity (and makes debugging easier).
- Smaller is faster.
- Make the common case fast.

RISC architecture features

- Fixed-size instructions (for easy decoding).
- Memory transactions allowed only in separate load/store instructions (for easier pipelining).
- Few and simple addressing modes.
- Large orthogonal register sets (no/few special registers).
- No/few complicated instructions.

Common misconception: RISC = small number of instruction opcodes.

Instruction Set Architectures

Comparison of the three GPR ISAs

- Memory memory (VAX, PA)
  - 2-3 memory accesses per typical ALU instruction.
- Register memory (VAX, PA, 68000, IBM 360, 8086)
  - 1-2 memory accesses per typical ALU instruction.
  - Smaller size code.
  - Variable instruction length.
  - Easier hand-coding.
  - Memory bottleneck.
- Register register (load/store)
  - 0 memory accesses per typical ALU instruction.
  - Simple instructions and code generation.
  - Higher instruction count.
  - RISC (MIPS) best for current compiler technology.

Motivation for Reduced Instruction Set Computer (RISC)

Research in the early 1980s found that:

- Most applications used a small set of simple instruction types, as many of the complex instructions of the earlier Complex Instruction Set Computer (CISC) architectures could not be effectively used by compilers, (example of CISC: Intel's x86, Motorola's 68K family and DEC's VAX).
- In some cases, sequences of simpler instructions gave better performance and allowed better compiler optimizations than complex instructions.
Bits, bytes, and such

- **BIT:** one binary digit (1 or 0)
- **NYBBLE:** a 4-bit binary number
- **BYTE:** an 8-bit binary number
- **WORD:** an ordered set of \( n \) bits, where \( n \) is **architecture-dependent**
- **DOUBLEWORD:** an ordered set of 2 words
- **QUADWORD:** an ordered set of 4 words

MIPS

**“Microprocessor without Interlocked Pipeline Stages”**

- founded in 1984 by researchers from Stanford University (Hennessy), IBM and Motorola, **MIPS Computer Systems**, released in 1986 the the R2000, one of the earliest commercially available RISC microprocessors

(from MIPS home page: [www.MIPS.com](http://www.MIPS.com))

- used by NEC, Sony, Nintendo, SGI, Philips, Siemens, Toshiba, TI, ...

How many different values...

...can be expressed with \( a \):

- bit?
- nybble?
- byte?
- 16-bit word?
- 24-bit word?
- 32-bit word?
- 64-bit word?
- 79-bit word?
- 143-bit word?

**Our MIPS ISA**

- 32-bit architecture what does it mean?
- 32 registers
- instruction set:
  - computational: **and, andi, or, ori, add, addi, sub, slt,**
  - load/store: **lw, sw**
  - flow control: **j, beq**
MIPS binary instruction formats

I-type (Immediate) instructions

<table>
<thead>
<tr>
<th>op</th>
<th>base/rs</th>
<th>rt</th>
<th>offset/immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26 25</td>
<td>21 20</td>
<td>16 15</td>
</tr>
</tbody>
</table>

- **op (opcode)**: the basic operation of instruction (e.g., lw opcode = 35, addi opcode = 8, beq opcode = 4)
- **rs/base**: register containing a source operand or base address
- **rt**: destination register in addi or lw, source register in sw, or second operand in beq
- **offset/immediate**: immediate field in computational instructions, byte address offset (wrt rs) in load/store instructions, word address offset (wrt PC) in branch instructions always sign-extended to a 32-bit value

Example assembly instruction:
```assembly
add $t0, $s1, $s2
```

Binary instruction:

<table>
<thead>
<tr>
<th>MEANING</th>
<th>comp</th>
<th>$s1</th>
<th>$s2</th>
<th>$t0</th>
<th>-</th>
<th>add</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

``add`` is an **R-type (Register)** instruction

MIPS binary instruction formats

R-type (Register) instructions

Format:
```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26 25</td>
<td>21 20</td>
<td>16 15</td>
<td>11 10</td>
<td>6 5</td>
</tr>
</tbody>
</table>
```

- **op (opcode)**: the basic operation of instruction, also lets the CPU figure out the instruction format. op = 0 in all R-type instructions
- **rs**: first register source operand
- **rt**: second register source operand
- **rd**: register destination
- **shamt**: shift amount
- **funct**: function variant (e.g., add and sub both have op = 0 but add has funct = 32 and sub has funct = 34)
MIPS binary instruction formats

R-, I-, and J-type format comparison

R-type:

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>

J-type:

<table>
<thead>
<tr>
<th>op</th>
<th>base/pc</th>
<th>rt</th>
<th>offset/immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>26</td>
</tr>
</tbody>
</table>

J-type:

<table>
<thead>
<tr>
<th>op</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
</tr>
</tbody>
</table>

Homework

Recommended exercises

- Ex. 3.1 to 3.11