A Simple Microprocessor

RISC Processors

- Architectures and complex instruction sets:
  - Long design time (control circuit, ...)
  - Fast evolution of technology not fully taken advantage of:
    - Operands in memory → long access time
    - More complex compilers
- RISC (Reduced Instruction Set Computer):
  - Few instructions, few addressing modes, few data formats, fixed instruction size.
  - Operands in registers only for fast access.
  - Instructions decomposed into pipeline stages
    - More simple design, low cycle time.

Instruction Sets

- The instruction set describes an abstract version of a processor (ISA: Instruction Set Architecture).
- An instruction set can correspond to many different implementations.
  - Example: x86 ISA and Intel processors.
- The instruction set must be able to:
  - Access memory
  - Perform arithmetic and logic operations
  - Control the program flow (branching)
**Designing a Simple Processor**

**Specifications** of the LC-2

- Use a 16-bit word:
  - Circuit size and speed constraints
  - Memory size
- A RISC processor (load/store, register operands, fixed-size instructions)
- Minimum ISA:
  - ALU: addition and minimum number of logic operations; operands: either registers, or immediate;
  - Memory: absolute addressing, index addressing
  - Control: unconditional branches, procedure calls, conditional branches (possible tests: zero, positive, negative); system calls; direct and indexed addressing
- Questions:
  - What ISA format?
  - What implementation for the processor?

---

**Instruction Set**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Rs1</th>
<th>Rs2</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Instructions size = balance between
  - Number of instructions/operands
  - Number of registers
  - Memory size
  - Control circuit complexity
- Example: 16 bits for data word/instructions:
  - 16 instructions → 4 opcode bits.
  - 12 bits for operands (3 registers for ALU instructions; example: ADD Rd=Rs1,Rs2)
  - Maximum 4 bits per register number (16 registers)
  - Base size (LOAD Rd=Rs1,base)
  - 16 bits for data word/instructions
- Predict the most important characteristics of upcoming architectures:
  - Adding new instructions
  - Increasing the number of registers
  - Facilitate memory access (large base size)
  - ...

---

**Instruction Set Design**

- Arithmetic and logic instructions:
  - Be able to write any logic expression.
  - Be able to make any arithmetic computation.
- Two addressing modes:
  - Immediate
  - Direct
  - ADD Rd=Rs1,Rs2
  - ADD Rd=Rs1,Rs2
  - ADD Rd=ET(Rs1,Rs2)
  - ADD Rd=Rs, valeur
  - ADD Rd=ET(Rs, valeur)
  - NOT Rd=Rs
  - NOT Rd=Rs

---
Instruction Set Design

- Memory access:
  - Load
  - Store
  - 3 addressing modes:
    - Direct
    - Indexed
    - Immediate

- LD Rd→M(address)
- ST Rs→M(address)
- Direct addressing
  - Address = PC[15:9],offset[8:0]
- LDR Rd→Rs, index
  - Direct addressing
  - Address = Rs + index
- STR Rs2→Rs1,index
  - Direct addressing
  - Address = Rs1 + index
- LEA Rd→address
  - Direct addressing
  - Rd = PC[15:9],offset[8:0]

Instruction Set Format

- Opcode: 4 bits
- Operands: 12 bits

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>RR</td>
<td>0010</td>
<td>LD</td>
</tr>
<tr>
<td>0100</td>
<td>JMP/JSR</td>
<td>0110</td>
<td>LDR</td>
</tr>
<tr>
<td>1100</td>
<td>JMPH/JSRR</td>
<td>1110</td>
<td>LEA</td>
</tr>
<tr>
<td>1111</td>
<td>TRAP</td>
<td>0111</td>
<td>ST</td>
</tr>
<tr>
<td>1111</td>
<td>RET</td>
<td>0111</td>
<td>STR</td>
</tr>
</tbody>
</table>

- ≤ 16 instructions → 4 opcode bits.
- Opcode:
  - Simplify control circuit
  - Possible ISA extensions
**Instruction Set Format**

- **8 registers → 3 bits.**
  - ADD, AND:
    - Rs 00L
  - Send instruction address
  - offset
  - Execute
  - Decode (fetch operands)
  - Rs 00L

- **LD/ST:**
  - 01234567891011
  - index
  - trapvect8

- **ADD, AND:**
  - 0
  - NOT:
  - 1

- **JSRR/JMPR:**
  - 0

- **LSR/LSR:**
  - offset

- **TRAP:**
  - 000

- **LEA:**
  - 000

- **JSRR/JMPR:**
  - 0

- **LD/ST:**
  - 000

- **RET:**
  - 000

- **BR:**
  - 0

- **Instruction execution stages (1 stage = 1 processor cycle):**
  1. Send instruction address
  2. Fetch instruction
  3. Store instruction
  4. Decode (fetch operands)
  5. Compute address
  6. Fetch operands from memory
  7. Execution
  8. Write result

**Processor Architecture**

- ADD R5, R4, #3
  - LC-2