Y35PES

Programming for Embedded Systems

Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz  nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes
How a Processor Works (1)

- Essentially, a microprocessor (CPU) is nothing but a complex finite-state machine.
- Basically, it consists of a counter (Program Counter, PC), registers (Instruction Register IR and Accumulator ACC) and instruction decoding / execution units (Arithmetical / Logical Unit, ALU and Instruction Decoding Unit, IDU).
How a Processor Works (2)

Instruction Format:

- Opcode (instruction code) – determines what is to be done
- Operands (implicit or explicit, direct or indirect) – specifies what or where is the data

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operands</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td><code>ld r0, 0x3E</code></td>
<td>0x00F03E</td>
</tr>
<tr>
<td>0x03</td>
<td><code>mov r1, r0</code></td>
<td>0x01F1F0</td>
</tr>
<tr>
<td>0x06</td>
<td><code>add r1, 0x01</code></td>
<td>0x0AF101</td>
</tr>
<tr>
<td>0x09</td>
<td><code>rol r1, 0x03</code></td>
<td>0x10F103</td>
</tr>
</tbody>
</table>
How a Processor Works (3)

**Instruction fetch**
- CPU outputs the PC value to the address bus
- Memory outputs contents of the current address to the data bus
- Instruction is stored into Instruction Register (IR)
How a Processor Works (4)

Instruction decode

- IR outputs the instruction opcode to the IDU
- IDU decodes the instruction, providing input signals to the ALU
How a Processor Works (5)

PC Incrementation

- The PC outputs its current value to the address bus
- ALU increments the value and writes it back to the PC
How a Processor Works (6)

Operand Fetch (Optional)
- IR outputs the instruction operand (address) to the address bus
- Memory provides the operand, which is stored into the ACC
How a Processor Works (7)

Instruction Execute
• ALU executes the instruction and stores the result into the ACC
How a Processor Works (8)

Result Write-Back (Optional)
• IR outputs the result address to the address bus
• ACC provides the result, storing it into the memory
**RISC x CISC**

Both terms are not very well defined nowadays, as the original technical meaning of the word RISC (*Reduced Instruction Set Computer*) had been blurred by the marketing. Technically, a RISC processor should (more or less) comply with following statements:

- All instructions are of the same fixed length
- All operations should be executed in single-cycle only
- Execution pipeline is used
- Load – store architecture is used – all operations are performed on registers only, the only allowed memory operations are load and store
- No microcode (hard-wired instructions)
- A wide pool of multi-purpose registers available – ideally orthogonal

**CISC (*Complex Instruction Set Computer*)** is basically the opposite. More complex instructions are utilized, operations performed directly on memory are allowed, and a limited amount of accumulator / index registers is available.

CISC machines are generally slower and more complex than their RISC counterparts. Whoever, today CPUs are essentially a mix of both architectures.
Pipelining (1)

- To make a CPU go faster by utilizing idle components, the pipelining was invented.
- Basically, pipelining means that while a previous instruction is being executed, another is already being fetched (instruction pre-fetch), yet another decoded etc...

**Classic RISC five-stage working cycle with a five-stage pipeline:**

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
</tr>
</tbody>
</table>

```plaintext
IF   ID   EX   MA   WB
IF   ID   EX   MA   WB
IF   ID   EX   MA   WB
IF   ID   EX   MA   WB
IF   ID   EX   MA   WB
```
Pipelining (2)

Pipelining Problems:

- Instruction independency is required for a pipeline to work efficiently.
- To operate at full speed, our pipeline needs to run 4 subsequent independent instructions while the first one is being completed.
- When consecutive instructions are interdependent (branching instructions), a solution needs to be found.
  - **Bubbles** (dummy solution) – A branch instruction is followed by (number of stages – 1) nops
  - **Branch Prediction** – A method of keeping track of which path was taken by a particular branch instruction in the past, and following that path the next time the same instruction is encountered. A history table is maintained, indicating how often a branch at a given address is taken or not taken.
  - **Speculative Execution** – An arbitrary execution path is chosen. If the execution goes down the wrong way, the intermediate results are scrapped / undone.
Pipelining Pros and Cons:

- Pipelining makes a processor run faster by concurrent utilization of its modules, without the need to increase the clock speed.
- Its performance depends greatly on the structure of the code being executed (instruction interdependencies, branching etc.), making the overall processor performance somewhat unpredictable. A processor without the instruction pipeline has a stable instruction throughput, while the pipeline makes the performance vary.
- Pipelined processors are more complex and a little more expensive to manufacture than their pipeline-less counterparts.
A Short List of Important Terms

- **Endian** – the order in which a multi-byte number is stored in the memory. Little endian means that the least significant byte (little) is being stored at the lowest address. Big endian is the opposite.

- **Microcode** – a way to simplify a processor design by introducing so-called “microinstructions”. More complex instructions are executed as a set of microinstructions. Obviously, it is slower than a “hard-wired” architecture.

- **Predicated Instructions** – conditionally executed instructions. Their execution is predestined by a condition register status. If required conditions are not met, respective instruction is omitted.

- **Superscalar Architecture** – a CPU architecture allowing simultaneous execution of multiple instructions under certain conditions.

- **Stack Frame** - A segment of a stack which holds parameters, local variables, previous stack frame pointer and a return address of a function.

- **Orthogonal Register Architecture** – Usually, usage of some of the registers is somewhat restricted. If all registers are multi-purpose and an arbitrary set of registers may be used as operands with an arbitrary instruction, the register architecture is referred to as orthogonal.
Microprocessor x Microcontroller x SOC

- A **microprocessor** is a stand-alone CPU with very limited range of peripherals. Usually, microprocessor designs are focused on performance rather than connectivity, with mother-boards providing most of the interfaces. Their usage in Embedded Systems is very limited.

- A **microcontroller (MCU)** is a processor equipped with a wide range of various HW peripherals / interfaces. Its performance is usually inferior to a CPU, but it is designed to ease its interconnection with the surrounding environment. MCUs are often designed to comply with industrial temperature / mechanical ratings. They are widely used in Embedded Systems.

- A **SOC (System–On–Chip)** is a MCU equipped with FLASH / RAM, internal oscillator and PLL and all other props necessary to form a computing system. A SOC is essentially a chip that only requires a power supply, everything else needed to form a computing system is integrated on-chip. SOCs are often hard to expand, they are used mainly as a base of simple single-purpose modules.
Lecture Summary – Essential Things to Remember

- A **processor** is basically a **finite-state machine**, comprising of Program Counter, ALU, a set of registers and **instruction decoding and control logic**.

- Basically, a **RISC** processor working cycle comprises of five parts: Instruction fetching, instruction decoding, execution, memory access and register write-back.

- **Pipelining** is a CPU speed-up technique, focused on **concurrent utilization of all vital CPU modules**, essentially making the CPU process multiple instructions at a time.

- A **microprocessor** is a **high-performance chip** with **limited range of peripherals**. A **microcontroller** is a somewhat lesser–oomph processor, equipped with **wide range of peripherals**. A **SOC (System-On-Chip)** is a **stand-alone computing system** integrated on a **single chip**.