Digital System

Microprocessor project: Digital watch

Fabrice Ben Hamouda, Yoann Bourse, Hang Zhou

2009-2010: Semester 1
# Table of contents

1. **Netlists and simulator**
   - Netlists
   - Specifications of the simulator
   - Circuit watch

2. **Global functioning**

3. **Microprocessor**
   - Instruction set - specifications
   - Assembly translator
   - Netlist realisation
   - Assembly watch

4. **Results**
   - Compilation
   - Results overview
Netlist language

- Very **simple**
- Inspired by the course and the instructions
- Easily manipulable by the teachers

**Examples**

```plaintext
s, r = FullAdd(a, b, c) {
    s = a ^ b ^ c
    r = (a & b) | ((a ^ b) & c)
    inst o = clk2 ()    # useless
}
```

```plaintext
o = clk2 () {
    o = Z(c)
    c = ~ Z(o)
}
```
**Drawback:** no high-level functions (loops, recursion...)

⇒ **Solution:** PHP as a Netlist generator

- Easy mix of PHP code and netlist language
- Netlists are true overview of real circuits
Example

```php
m = main(<=?=listV("a",0,$m)?>,<=?=listV("s",0,$n)?>){
    inst m = mux3(<=?=listV("a",0,$m)?>,<=?=listV("s",0,$n)?>)
}
```

Gives for $m = 3$ and $n = 5$

```php
m = main(a[2],a[1],a[0], s[4],s[3],s[2],s[1],s[0])
{
    inst m = mux3(a[2],a[1],a[0], s[4],s[3],s[2],s[1],s[0])
}
```
**Example**

```php
m = main(<?=listV("a",0,$m)? >,<?=listV("s",0,$n)? >){
    inst m = mux3(<?=listV("a",0,$m)? >,<?=listV("s",0,$n)? >)
}
```

**Gives for $ m = 3 and $ n = 5**

```php
m = main(a[2],a[1],a[0], s[4],s[3],s[2],s[1],s[0]){
    inst m = mux3(a[2],a[1],a[0], s[4],s[3],s[2],s[1],s[0])
}
```
Specifications of the simulator

Ability to set cycle frequency.
Two modes:

- Simple interpretation
- Compilation towards C++: circuits run like a program

Outputs:

- Truthtable
- Seven segments interface
- Timing diagram
Circuit watch

1st watch: circuit watch (huge circuit)
⇒ Demonstration
Close to MIPS in order to use our OCAML Compiler.

- Harvard architecture
  (instruction bus separated from data bus).
- One instruction per clock cycle.
Close to MIPS in order to use our OCAML Compiler.

- Harvard architecture
  (instruction bus separated from data bus).
- One instruction per clock cycle.

**Bus length, register length/number, ...**

All is 16!
General presentation

- 16 registers of 16 bits
- 16 bits instructions
- Up to 64 K instructions (ROM) (16 bits address length)
- Up to $2 \times 64$ KB of data RAM (16 bits address/data length)
Special instructions/features

- Up to 8 (±2 for the timer) output ports and 8 (±2 for the timer) input ports of 16 bits
  → controlled by input/output instructions.
Special instructions/features

- Up to 8 (+2 for the timer) output ports and 8 (+2 for the timer) input ports of 16 bits → controlled by input/output instructions.
- Sleep instruction
Special instructions/features

- Up to 8 (+2 for the timer) output ports and 8 (+2 for the timer) input ports of 16 bits
  → controlled by input/output instructions.

- **Sleep** instruction

- One 16 bits clock timer
  - Incremented each clock cycle.
  - Timer period controlled by an output port.
  - Wake up microprocessor each time it reaches its period.
## External interfaces

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>real 16 bits input port 0-7</td>
</tr>
<tr>
<td>8</td>
<td>16 bits timer <em>period</em></td>
</tr>
<tr>
<td>9</td>
<td>16 bits timer</td>
</tr>
</tbody>
</table>
# MIPS comparison

<table>
<thead>
<tr>
<th></th>
<th>MIPS</th>
<th>Our microprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>(32 \times 32) bits</td>
<td>(16 \times 16) bits</td>
</tr>
<tr>
<td>ALU</td>
<td>(+,, -,, \times,, \div,, \mod)</td>
<td>(+,, -)</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>(\oplus)</td>
</tr>
<tr>
<td>Conditional branch</td>
<td>on a comparison</td>
<td>last result (zero, carry)</td>
</tr>
<tr>
<td>Special instructions</td>
<td>syscall</td>
<td>input, output, sleep</td>
</tr>
<tr>
<td>Special features</td>
<td></td>
<td>16 bits timer</td>
</tr>
</tbody>
</table>
Specifications

ROM
- <= 64K x 16bits instructions

address

instructions
- jmp, la

data

RAM
- <= 64K x 16bits data

address, data

ALU
- 16x16bits registers
- carry, zero, 1 bit registers

Outputs
- 8+2 x 16bits

Inputs
- 8+2 x 16bits

control unit

TIMER
- 16bits

timer period
- set
- reset

Instruction sleep

Assembly translator

Translates assembly code into machine language (integer instructions)

- Replace labels
- Pseudoinstructions: \( addui \) (add a register with a literal)
- Simulation in C
Example

li     W0  5  ; W0 <- 5

output 0 W0  ; output W0=5 on port 0

input W1 1  ; read port 1 in W1

addu W1 W1 W0 ; W1 <- W1 + W0

output 0 W1  ; output result on port 0
Assembly watch

2\textsuperscript{nd} watch: assembly watch (slower)
⇒ Demonstration
Compile OCAML on our microprocessor thanks to MIni MOdules (MIMO) project:

- Add some OCAML functions (sleep, inputs, outputs)
- Software multiplication, division, modulo
- Software heap management
- Conditional branch conversion
- Terminal recursion management
Compilation II

Drawbacks:

- not very well optimized
- longer code due to use of stack
  (manual watch: only registers !)
- big assembly file (1299 line in generated assembly watch instead of 129 in manual assembly watch)
Advantages of OCAML (high level language):

- easy function call
- automatic heap/stack management (for big project)
- transparent use of software operation (multiplication, division)
- easy debugging (just use OCAML executable)

3rd watch: OCAML watch

⇒ Demonstration
## Results overview

<table>
<thead>
<tr>
<th></th>
<th>Compilation Mode</th>
<th>Interpreting Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netlist</td>
<td>$6.0 \times 10^6$</td>
<td>$1.5 \times 10^6$</td>
</tr>
<tr>
<td>Asm</td>
<td>$0.96 \times 10^6$</td>
<td>$0.09 \times 10^6$</td>
</tr>
<tr>
<td>Ocaml</td>
<td>$0.96 \times 10^6$</td>
<td>$0.10 \times 10^6$</td>
</tr>
</tbody>
</table>

Speed Comparison between Different Modes and Different Watches

Number of cycles simulated per minute