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 functionning**
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

```
o = clk2(){
    o=Z(c)
    c=~ Z(o)
}
```

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

PHP Netlist language

Drawback: no high-level functions (loops, recursion...)

$\Rightarrow$ **Solution:** PHP as a Netlist generator

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

PHP Netlist language (example)

Example

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

PHP Netlist language (example)

Example

```
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

```
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

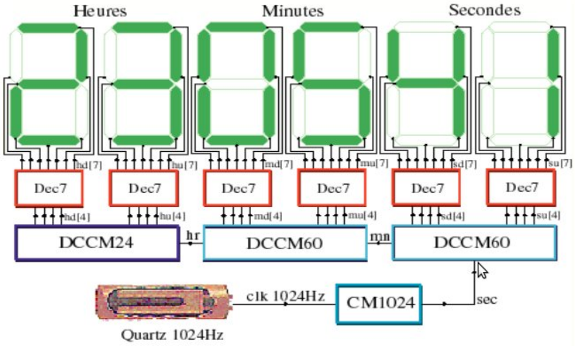
Ability to set cycle frequency.
Two modes:

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

Outputs:

- Truthtable
- Seven segments interface
- Timing diagram

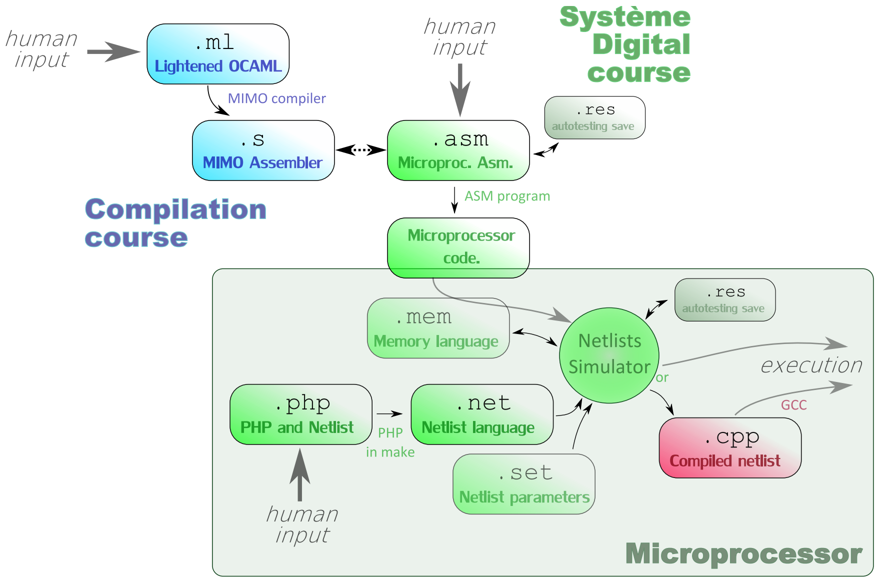
Circuit watch



1^{st} watch: circuit watch (huge circuit)

$\Rightarrow$ Demonstration

Global functionning



General presentation

Close to MIPS in order to use our OCAML Compiler.

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

General presentation

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 2×64 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
  $\longrightarrow$ 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
  $\longrightarrow$ 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
  $\longrightarrow$ 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.

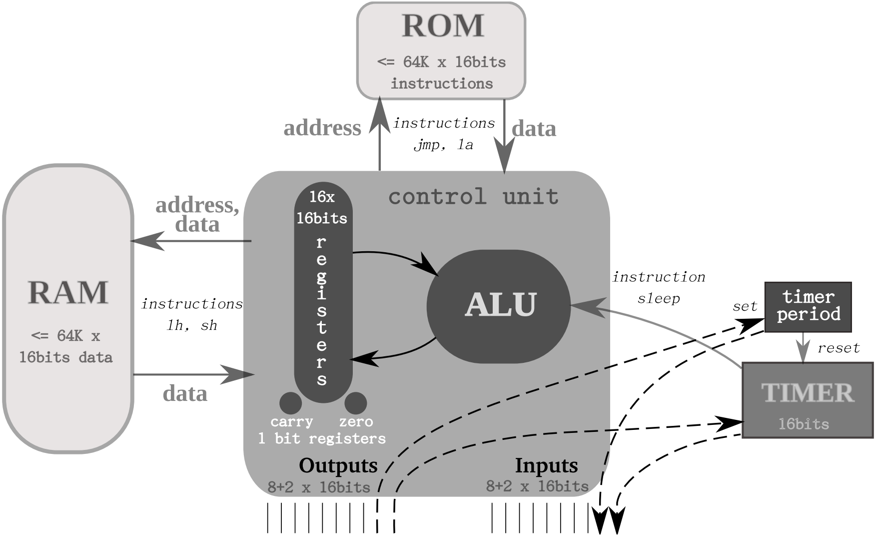
External interfaces

	Input	Output
0-7	real 16 bits input port 0-7	real 16 bits output port 0-7
8	16 bits timer *period*	
9	16 bits timer	

MIPS comparison

	MIPS	**Our microprocessor**
Registers	32×32 bits	16×16 bits
ALU	$+ \; - \; \times \; \div$ mod $\& \; \| \; \oplus \; \ll \; \gg$ not	$+ \; -$ $\& \; \| \; \oplus \; \ll 1$ not
Conditional branch	on a comparision	last result (zero, carry)
Special instructions	syscall	input, output, sleep
Special features		16 bits timer

Specifications



Assembly translator

Translates assembly code in machine language (integer instructions)

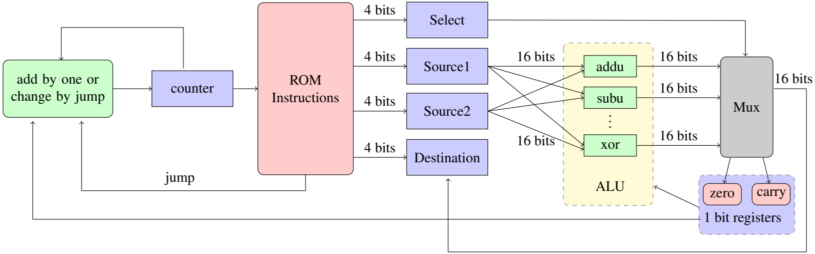
- Replace labels
- Pseudoinstructions : (*addui* : add a register with a literal)
- Simulation in C

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
```

Netlist realisation



Assembly watch

2^{nd} watch: assembly watch (slower)

$\Rightarrow$ Demonstration

Compilation I

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)

Compilation III

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)

3^{rd} watch: OCAML watch
$\Rightarrow$ Demonstration

Results overview

	Compilation Mode	Interpreting Mode
Netlist	6.0×10^6	1.5×10^6
Asm	0.96×10^6	0.09×10^6
Ocaml	0.96×10^6	0.10×10^6

Speed Comparison between Different Modes and Different Watches
Number of cycles simulated per minute