EDC: exam

Renaud PACALET

19 June 2015

You can use any document you need. Please number the different pages of your work and indicate on each page your first and last names. Write your answers in French or in English, as you wish, but avoid mixing the languages. If some extra information or hypotheses are missing to answer a question or solve a problem, decide by yourself and write down the added hypotheses or information. If you consider a question as absurd and thus decide not to answer, explain why. If you don't have time to answer a question or solve a problem but know how to, briefly explain your ideas. The first part is a set of five questions (2 points each) and the second part is a small problem (10 points).

Important advice #1: quickly go through the document and answer first the easy parts.

Important advice #2: copying verbatim the slides of the lectures or any other provided material is not considered a valid answer.

1 Questions

1.1. Consider the following natural language property:

```
If the signal EN is high and RNW is high, then,
on the next cycle, EN is low and EN cannot be
high again before VALID is high.
```

Translate the property in CTL in the two following cases expressed as assumed CTL properties:

- `AG AF VALID=1`
- `EF EG VALID=0`

1.2. In digital circuits, why is it usually considered as a good idea to freeze the content of registers when possible?

1.3. We want to design a synthesizable VHDL model of a circuit which all outputs are outputs of registers and all registers are synchronized by the same edge of the same clock. What is the minimum number of VHDL processes we need? Why?

1.4. What are VHDL resolved types? What are they used for? What shall they not be used for?

1.5. Assume the designer of the following VHDL code wanted to design a combinatorial synthesizable adder / subtracter. What do you think of her work? Identify the errors if any and, for each of them, explain why it is an error, what undesirable effect it has and finally, write down a new VHDL code with all the errors fixed.

```
signal add_not_sub: std_ulogic;
signal a, b, s: signed(31 downto 0);
...
process(a, b, s)
begin
  if add_not_sub = '1' then
    s <= a + b;
  elsif add_not_sub = '0' then
    s <= a - b;
  end if;
end process;
```

2 Design of a One Instruction Set Computer (OISC)

Surprisingly, the smallest number of instructions needed to build a computer system is... one. The resulting machine is a real CPU and it is even possible to design a compiler for it, for any classical language, like, for instance C. There are several ways to design such a weird CPU. The one we will explore is named SUBLEQ for *SUBtract and branch if Less than or EQual to zero*. It is also the name of its single instruction.

Our SUBLEQ CPU is a 32-bits CPU with 4GB address space. Figure 1 represents SUBLEQ and its environment and table 1 lists all its inputs and outputs.

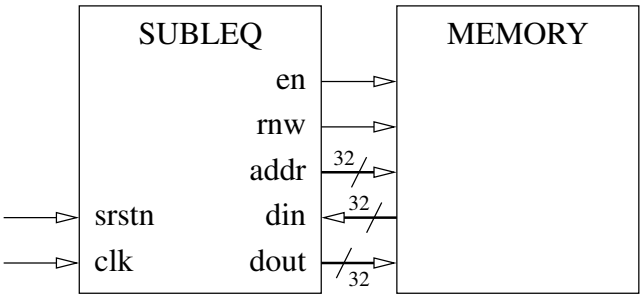

Figure 1: The SUBLEQ CPU

Note: the *MEMORY* block represented on figure 1 contains the memory of our computer but also the interface registers of the peripherals (boot ROM, keyboard, mouse, display controller, network card...) that are needed to build a real computer. In the following we do not care about the organization of the address space and we can safely consider it as a true 4GB memory, with 32 bits address and data buses. Because it is 32-bits wide, the two Least Significant Bits of the address bus are ignored by the memory and only the 30 Most Significant Bits are used to select the target 32-bits word of a memory access. It is **not** an error if the 2 LSBs are not zero but it has the same effect as if they were.

Name	Direction	Bit-width	Description
`clk`	Input	1	Global clock. SUBLEQ is synchronized on the rising edge of `clk`
`srstn`	Input	1	Global, synchronous, active low reset
`en`	Output	1	Enable. SUBLEQ asserts `en` to read or write in its address space
`rnw`	Output	1	Read-Not-Write. Asserted for reads, de-asserted for writes
`addr`	Output	32	Byte-address of read or write target. Two LSBs ignored
`din`	Input	32	Read data
`dout`	Output	32	Write data

Table 1: Interface specification of the SUBLEQ CPU

SUBLEQ is a synchronous machine (rising edge of `clk`) with synchronous reset (`srstn`). To read in its address space it puts the address on its `addr` output, asserts its `rnw` output and its `en` output. On the next rising edge of `clk` SUBLEQ can sample the read data. Writing is similar: SUBLEQ puts the address on `addr`, the data to write on `dout`, de-asserts `rnw` and asserts `en`. On the next rising edge of `clk`, the environment samples the address and the data and performs the write. These timings are summarized on figure 2.

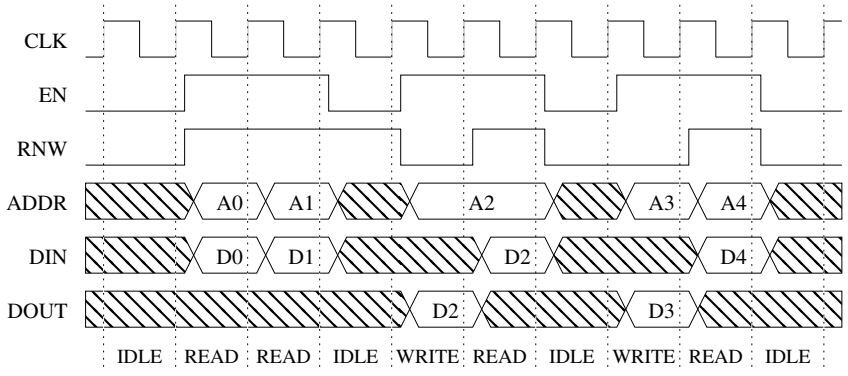

Figure 2: The timing of SUBLEQ accesses to its external address space

The unique `subleq` instruction of our CPU takes three 32-bits arguments, `a`, `b` and `c`:

```
subleq a,b,c // subleq assembly code
```

The 3 arguments are considered as 32 bits addresses in memory (with the 2 LSBs ignored by the memory). The behaviour of this instruction is:

```
memory[b] = memory[b] - memory[a]
if (memory[b] <= 0) goto c
```

That is, the instructions subtracts the memory word at address `a` to the memory word at address `b`, stores the result at address `b`, and, if the result is less or equal zero, goes to instruction at address `c`, else goes to next instruction. Each instruction in memory is coded on 3 consecutive 32-bits words, one for `a` (lowest memory address), `b` and `c` (highest address (as we have only one instruction we do not need any operation code). So, the program counter always increments by 3 words (12 bytes), unless the current `subleq` instruction jumps somewhere else.

When the CPU is reset, its program counter (also known as instruction pointer) is forced to zero. So, the first executed instruction after reset is always at addresses `0x00000000` (`a`), `0x00000004` (`b`) and `0x00000008` (`c`).

TODO (1 points): Study the 12-words (4 instructions) program represented in table 2. Words and addresses are in hexadecimal. The first address is the reset address, so it is the program that is launched after reset. Explain its behaviour.

Address	Value
0x00000000	0x00000000
0x00000004	0xffffffff
0x00000008	0x00000000
0x0000000c	0x00000000
0x00000010	0x21fa7aa8
0x00000014	0xab2ee5d0
0x00000018	0xefc51add
0x0000001c	0xe8022dcf
0x00000020	0xd970380b
0x00000024	0x7b795d18
0x00000028	0xc9dde8e6
0x0000002c	0x54d72538

Table 2: Example SUBLEQ program

TODO (1 points): Our memory contains 2 integers at addresses `x` and `y`. Design a program for SUBLEQ that adds these two integers and stores the result at address `z`.

TODO (3 points): Assume we do not care about performance (clock frequency, number of clock cycles per instruction) but only about the hardware resources. We want the smallest possible CPU. But, of course, it **must** respect the functional and interface specifications. Design a block diagram of the SUBLEQ CPU. Clearly identify the internal registers, the combinatorial parts and the interconnections between them all. Indicate the behaviour of the combinatorial parts and name the interconnections and the registers with meaningful names. If you use a control part to sequence the operations, do not detail its internals. Just show its inputs and outputs and give a state diagram.

TODO (4 points): Design, in plain synthesizable VHDL (one entity and one architecture), the small SUBLEQ CPU.

TODO (1 points): Imagine that, instead of hardware resources, you are only interested in performance. What modifications would you introduce to your slow and small version (do not code, just list the optimizations that you have in mind).