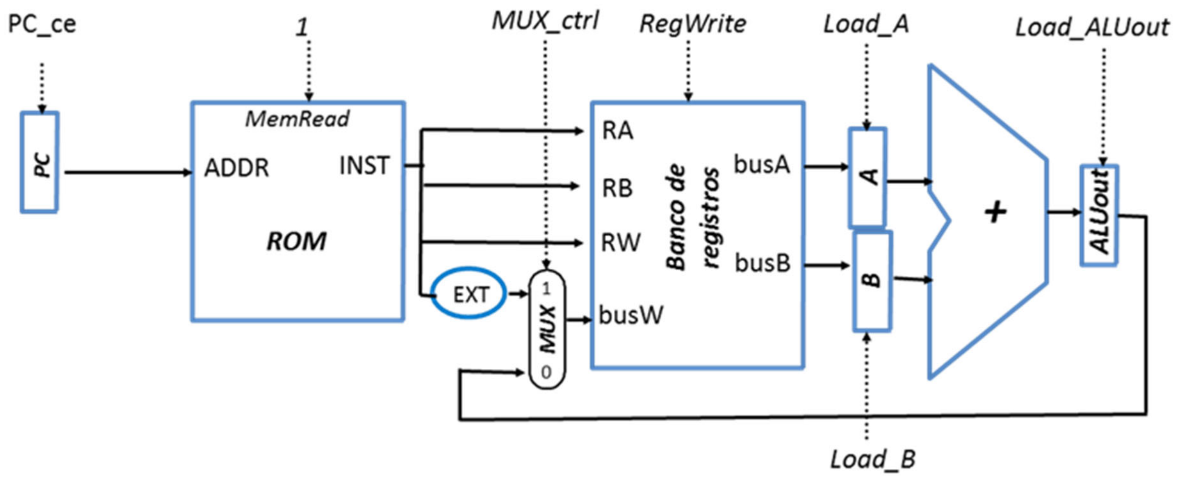
DESIGN OF A PROCESSOR WITH A REGISTER BANK

Lab Assigment #2



Computer Architecture and Organization 2
Grado Ingeniería de Computadores – 2º curso

Luis M. Ramos Alejandro Valero José Luis Briz Javier Resano
luisma@unizar.es alvabre@unizar.es briz@unizar.es briz@unizar.es jresano@unizar.es

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

1 SUMMARY

In this second session you will learn how to work with a hardware description language and a professional simulation environment based on test benches and schedules. You will design a very simple functional processor with a register bank and an adder. You will program this processor to perform a series of writes and adds. Then you will perform a timing analysis to find out the maximum frequency at which the processor can operate, and you will visualize the execution of the instructions in a timeline.

***Take a seat at the lab** and place your pre-lab work (Sec. 2.1.) on the table, so that the instructor can see it. Before starting to work, log in to Moodle from one of the lab computers and **check in at the attendance control of the session**.*

*The practice is considered completed when the processor is working properly and you have submitted the vhdl **desing** plus **results of section 3** through Moodle.*

2 DESIGNING A PROCESSOR WITH A REGISTER BANK

2.1 PRE-LAB: DESIGN ON PAPER

a) Download the companion files[1] and study the inputs and outputs of the VHDL modules that you are going to use in the practice. Draw on a paper the following components: a **BR32** register bank *(BReg_delays.vhd*), a ROM (*ROM_I_delay.vhd*), a counter (*counter_delay.vhd*), a 32-bit adder (*adder32_delay.vhd*), a multiplexer (*mux2_1_delay.vhd*), three registers *A, B* and *ALUout (REGISTER_delay.vhd)*, and a control unit (*UC_Mov_add.vhd*) that generates the two control signals: ***RegWr,*** to write to the register bank; ***MUX_ctrl***, to choose what to write to the register bank, ***PC_ce***, to make the pc advance, and the register load signals. Please check out the schematic of the data path which appears at ther front cover of this Guide: there are only a few details missing (such as the bits used in some connections).

b) Interconnect the components, taking into account that:
   - One instruction will be written to each ROM address.
   - The counter will be used to address the ROM.
   - The adder will calculate the sum of the two registers read from **BR32**.
   - The multiplexer will select the appropriate data to write to the **BR32**.
   - The processor must support the following instructions:

```
mov K,rd          ; SignExt(K)-> BR(rd);pc++     K is a 16-bit immediate
add ra,rb,rd      ; BR(ra) + BR(rb)-> BR(rd); pc++
halt              ; execution stops
```

c) Defines the instruction format and instruction coding.
   - Pay attention to where we have encoded K in the data path skeleton we give you in *Mov_add.vhd*
   - The field for the target record must be the same in `mov` and `add.`
   - In principle you can place the operation code wherever you want, but it would be convenient to place it in the heaviest bits of the instruction format.

[1] Please find them in Moodle or logging into your hendrix account and copy them from /misc/practicas/aoc2, then run 'unzip AOC2_p2.zip'.

d) Draw the automaton of the control unit following a **Moore** scheme. As there are intermediate registers, each instruction will be executed in several cycles. Specify the RTL actions at each state (you do not need to indicate the value of the control signals).

e) Following the above instructions write an assembly program (*NIA.asm*) that stores in registers r1 - r7 the values obtained inwebdiis.unizar.es/~luisma/aoc2/nia1.php from your NIA and calculates the sum of all of them in r0.

2.2 LABORATORY WORK: DESIGN IN VHDL

a) Copy the provided vhd files and follow the instructions of the GHDL workflow that you will find in Moodle.

b) Simulate the testbench *BR_test.vhd* as long as necessary to be able to see all tests. Open the simulation waveform with GTKWave, and load the schedule configuration *br_testbench.gtkw* (*File / Read Save File*).

c) Check that the various read and write operations work correctly. Locate the propagation delays $d_{RA\rightarrow busA}$ and d_{busA} in the chronogram.

d) Open the *Mov_add.vhd* processor with the text editor of your choice. Do not modify the file name, just **fill in the header information with your data**. The components are already defined and instantiated, but you must connect them.

e) Describe in VHDL the UC you have designed. Follow the scheme of *UC_mov_add.vhd*. Do not modify the file name, just **fill in the header information with your data.** Assign a descriptive name to each state, specify which signals are activated in each state (Moore type UC), and define the next state based on the current state and the operation code.

f) Translate your program from section 2.1.d to **hexadecimal** and program the ROM. To do this edit *ROM_I_delay.vhd* and replace the 0x00000000 with the necessary instructions.

g) Simulate with GHDL the test *Mov_Add_test.vhd* the time needed to execute your instructions. Open the simulation with GTKWave and load the schedule configuration *mov_add_testbench.gtkw*. Check the execution of the instructions cycle by cycle. Are the instructions executing as you expected? Are they performing the correct writes to BR? Note that the modules you are using have associated delays, you can see their delays in the code, or in the temporal analysis section.

h) Finally, submit your circuit through the Moodle resource 'Practical Delivery 2'. You must also submit screenshots of the automaton of the control unit and the execution schedule.

3 TEMPORAL ANALYSIS

a) The VHDL modules have the following delays assigned to them: d_{REG} = 5 ns; $tsetup_{REG}$ = 1 ns; d_{ROM} = 8 ns; d_{CONT} = 6 ns; $tsetup_{CONT}$ = 1 ns; $d_{RA\rightarrow busA}$ = 7 ns; d_{busA} = 5 ns; $tsetup_{RW}$= $_{tsetup(RegWr)}$= 3 ns; d_{SUM} = 26 ns; $d_{MUX\_X\rightarrow Z}$ = 2 ns; $d_{MUX\_S\rightarrow Z}$ = 3 ns; d_{UC} = 3 ns. The *setup times* not shown are assumed to be negligible and we do not model them.

b) Identify the critical path of the multicycle processor and its delay (*dmax*). What is its maximum frequency (MHz) of operation (*fmax*)?

c) Open *Mov_add_test.vhd* and set the cycle time to *fmax* by editing the line:
constant CLK_period : time := 100 ns;. Check that everything is still working correctly.

d) How long does it take the processor to execute your code (tex)?

e) When you have all the calculations done enter them through the Moodle resource 'Temporal analysis practice 2'.