BYOC course

Assignment #3

Prep for Rtype only MIPS CPU

1) The Rtype only MIPS CPU and its main components

We would like to design part of the MIPS CPU which is capable of running simple programs with Rtype instructions only. There are 3 main parts involved. These are the Fetch Unit from HW2, the GPR File and the MIPS ALU.

In this homework/lab exercise we will design the GPR File and the MIPS ALU. In the next exercise we will tie the GPR File, the MIPS ALU and the Fetch unit together to form a Rtype MIPS CPU.

Below we see a simplified drawing of the Rtype MIPS CPU.

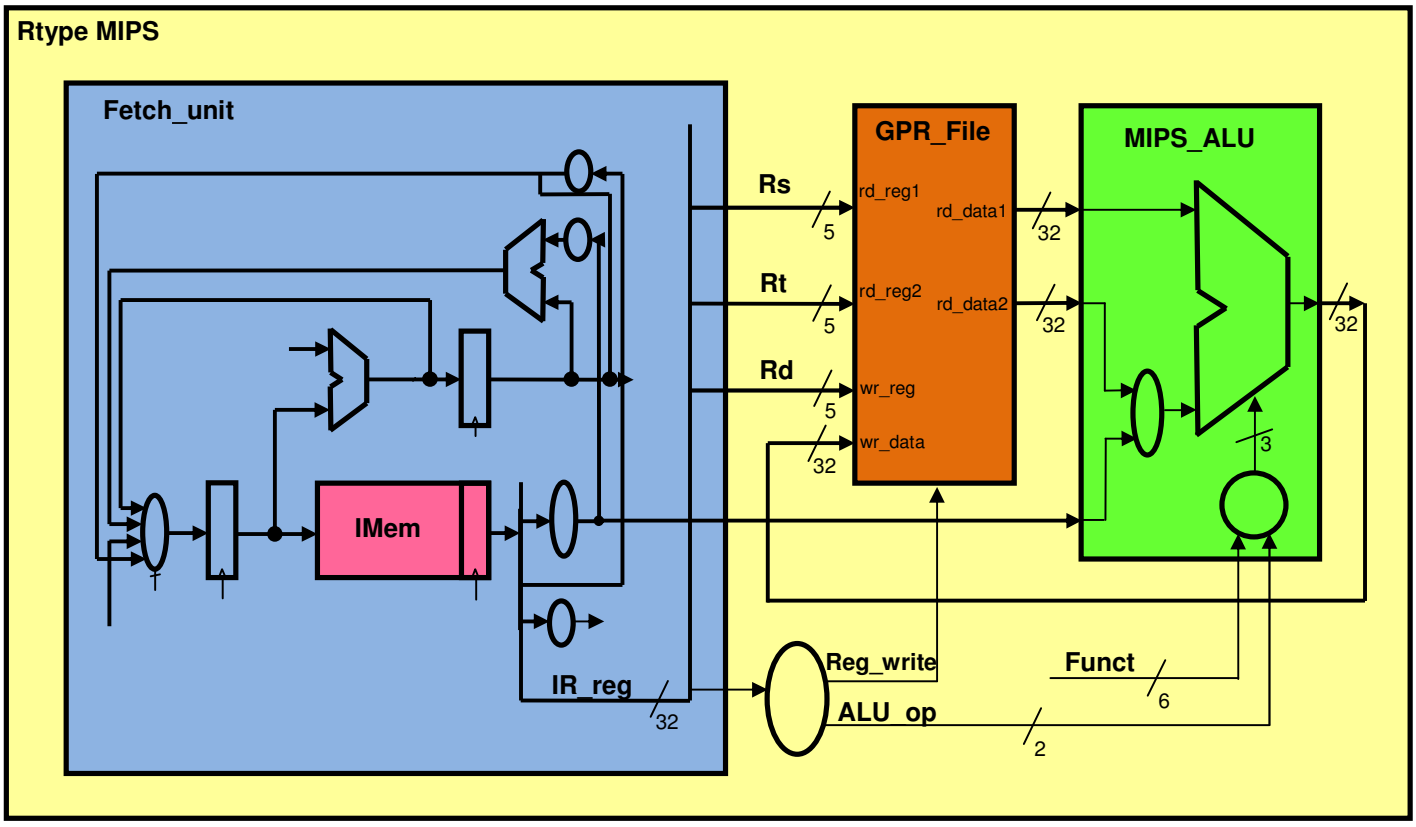

Fig. 1 – The Rtype only MIPS CPU – a simplified drawing

2) GPR – design & simulation

The inside of the GPR File is made of a dual port memory. That memory does not have a register at its output as we had in the IMem we used in the Fetch Unit of HW2. Only the writing process of the memory is triggered by the rising edge of the clock. When “wr_en”=’1’ and there is a rising edge in “wr_clk”, then the “wr_data” is written into the “wr_address” location of the memory. The reading from the dual port memory is a combinationl process.

Note that “wr_address” and “rd_address” are integers, and if your address signals is a STD_LOGIC_VECTOR signal you need to use the function conv_integer(your signal vector name) to convert the STD_LOGIC_VECTOR value to integer in order to set a value to the wr_address or rd_address.

We give you a vhd file called **single_port_memory.vhd** and you need to manipulate it to become a **dual_port_memory.vhd**. The skeleton of the **dual_port_memory.vhd** is given in the **dual_port_memory.empty** file so that you will use the signal names we decided on.

The outside to the GPR File is described in the skeleton file **GPR.empty**. In this file we implement the following:

2.1) Although the dual port memory we use has address 0 and so we can write data into that address and read data from that address, we will make sure that when we read from read_reg1=0, we will get rd_data1=x”00000000”.

2.2) Similarly, when reading from read_reg2=0, we will get rd_data2=x”00000000”.

2.3) We will add a GPR_hold input to the GPR file. When this input is ‘1’ there should not be a write operation at the rising edge of the clock even if the RegWrite signal is ‘1’.

So you need to prepare the files:

- **dual_port_memory.vhd** – that describes the dual port memory in which only the writing is synchronous (activated by the rising edge of the clock)
- **GPR.vhd** – that “wraps” the dual_port_memory component of 32 addresses of 32 bits each and performs what was requested in 2.1 and 2.2 above

To ease the design for you the GPR.vhd content is depicted in Fig. 2 below.

Now you can run a simulation and check your design with the additional three files of:

- **SIM_GPR_TB.vhd** - the TestBench file we prepared ahead of time
- **SIM_GPR_TB_data.dat** - the TestBench testing data file we prepared ahead of time
- **SIM_HW3_GPR_filenames.vhd** – In this file we specify the path of the data files used in simulation

With these 5 files you need to run the simulation and verify your design works fine. Note that you need to update the **SIM_HW3_GPR_filenames.vhd** with the actual path of the **SIM_GPR_TB_data.dat** file.

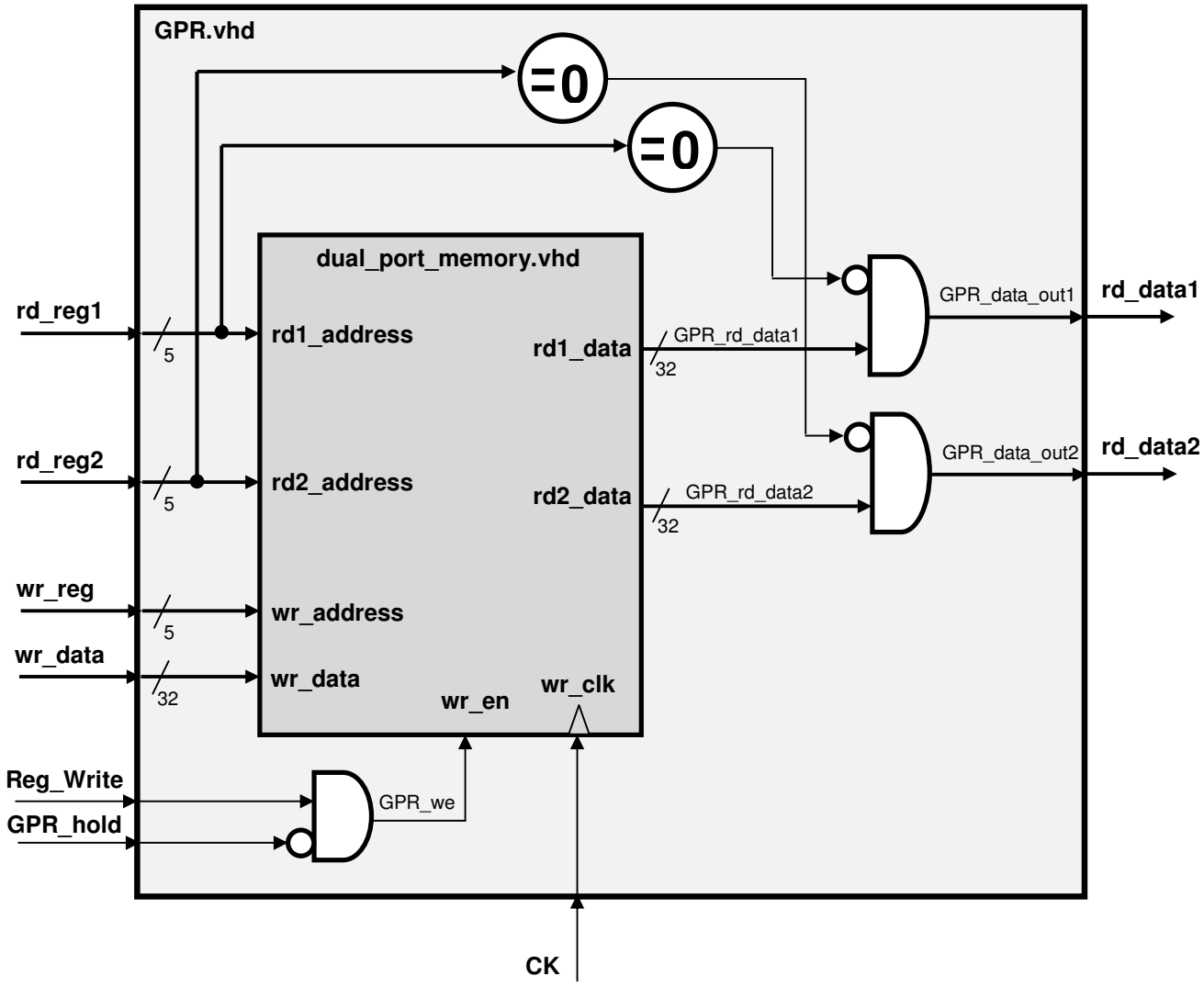

Fig. 2 – The inside of the GPR.vhd

You should submit a zip file of the entire GPR_File simulation project – see detailed instructions at section 4 of this document.

Also you need to attach a doc file with screen captures describing the simulation you made.
All i/o signals of GPR entity should be presented in the screen capture. Show the 2nd session of writing into the GPR File (clock cycles 46 to 55 = 920ns to 1100ns) and make sure that the values of all signals are readable. Explain what is seen in the rd_data1 and rd_data2 outputs of the GPR_File in these clock cycles.

3) MIPS ALU – design & simulation

The MIPS ALU is a combinational circuit. No FFs are involved. We "added" to the ALU also the ALU_src_B multiplexer and also the logic control that issues the ALU_cmd signal. The ALU_cmd is a 3 bit signal vector which determines what is the calculation done by the ALU.
If the ALU_cmd is "010", the ALU performs an addition. If ALU_cmd is "110", the ALU performs a subtraction. Here is the list of operation done by the ALU according to the ALU_cmd bits:

- ALU_cmd="000" => A and B
- ALU_cmd="001" => A or B
- ALU_cmd="010" => A + B
- ALU_cmd="011" => A xor B
- ALU_cmd="100" => A nand B - not used
- ALU_cmd="101" => A nor B - not used
- ALU_cmd="110" => A - B
- ALU_cmd="111" => SLT. 1 if A<B, 0 if not. A & B are considered 2's complement numbers

The logic that drives the ALU_cmd gets the 2 bit signal vector called ALUOP. When ALUOP="00", the ALU performs addition. When ALUOP="01", the ALU performs subtraction. When ALUOP="10", the ALU operation is determined by the 6 bit vector called Funct (function) that comes from the 6 LSBs of the IR reg. Here is the list of the Funct codes:

- Funct=" 100000" => ADD
- Funct=" 100010" => SUB
- Funct=" 100100" => AND
- Funct=" 100101" => OR
- Funct=" 100110" => XOR
- Funct=" 101010" => SLT

In all other cases we request to perform ADD.

The srcB mux selects what will be fed into the B input of the ALU. If ALUsrcB='0', we input the B_in data into the ALU B input. If ALUsrcB='1', we input the sext_imm data into the ALU B input.

We prepared a **MIPS_ALU.empty** file for your convenience.
You need to add all of the logic described above. When done, you should run a simulation using the the additional three files of:

- **SIM_MIPS_ALU_TB.vhd** - the TestBench file we prepared ahead of time
- **SIM_MIPS_ALU_TB_data.dat** - the TestBench Data file we prepared ahead of time
- **SIM_HW3_ALU_filenames.vhd** – In this file we specify the path of the data files used in simulation

With these 4 files you need to run the simulation and verify your design works fine. Note that you need to update the **SIM_HW3_ALU_filenames.vhd** with the actual path of the **SIM_MIPS_ALU_TB_data.dat** file.

You should submit a zip file of the entire MIPS_ALU simulation project– see detailed instructions at section 4 of this document. Also you need to attach a doc file with screen captures describing

the entire simulation you made – till 1200 ns. All i/o signals of the MIPS_ALU entity should be presented in the screen capture.

4) HW3 report

You should submit a single zip file for the Simulation of both entities. It should have three directories/folders. The first is called **GPR_File**, the 2nd is called **MIPS_ALU**, the 3rd is called **Disassembly**.

In the **GPR_File** directory you will have the following 3 sub-directories:

- **GPR_File_Src** - with all of your simulation sources
- **GPR_File_Sim** - with the simulation project
- **GPR_File_Docs** - Add a doc file with screen capture of the simulation showing the waveforms of the TB signal and the Console window. All i/o signals of GPR entity should be presented in the screen capture. Show the 2nd session of writing into the GPR File (clock cycles 46 to 55 = 920ns to 1100ns) and make sure that the values of all signals are readable. Explain in detail what do we see in rd_data1 and rd_data2 in these 10 clock cycles. The first few lines in the report will have your ID numbers (names are optional).

In the **MIPS_ALU** directory you will have the following 3 sub-directories:

- **MIPS_ALU_Src** - with all of your simulation sources
- **MIPS_ALU_Sim** - with the simulation project
- **MIPS_ALU_Docs** - Add a doc file with screen capture of the simulation showing the waveforms of the TB signal and the Console window. The screen captures should have the entire simulation you made (from its start to its end – till 1200 ns), and all of the MIPS_ALU i/o signals. No need to see the values of the signals, just the total picture and the console with a "Test Pass" message. The first few lines in the report will have your ID numbers (names are optional).

In the **Disassembly** directory you should have a doc file in which you disassemble a MIPS binary code and some explanations (answer questions).
See the questions in the file 18.1_MIPS_binary_code_for_disassembly_v4.docx

Note that the binary MIPS code you need to disassemble is the program we will be using in HW3. You need this disassembled code to understand what is done in HW3. That binary program to be disassembled appears in a Word file called MIPS_binary_code_for_disassembly_v2.doc and also in a text file called MIPS_binary_code_for_disassembly_v2.txt.

Use this file and add your disassembled code. See the appendix at the end of the document for MIPS instructions coding. Also explain in detail what is done by this code. Also explain how this code tests the GPR_file and ALU parts of a MIPS CPU.

At the end of this assignment you will have the necessary building blocks for our next assignment, HW4 – the "Rtype" MIPS CPU.

Enjoy the assignment !!

5) Appendix A – MIPS instructions coding

a. Codes of the Opcode fields - IR(31 downto 26)

sw =[101011]=43
lw =[100011]=35
lui =[001111]=15
ori =[001101]=13
addi =[001000]=8
beq =[000100]=4
bne =[000101]=5
j =[000010]=2
jal =[000011]=3
R-type =[000000]=0

b. Function field codes for RType instructions – IR(5 downto 0)

add =[100000]=32
sub =[100010]=34
and =[100100]=36
or =[100101]=37
xor =[100110]=38
slt =[101010]=42
jr =[001000]=8

Rs, Rt and Rd fields have a 5 bit binary number of the register (0-31)