BYOC course

Assignment #4

Rtype MIPS CPU

1) The Rtype MIPS CPU and its main components

In HW3 we stated that we want 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. We built the last two components in HW3.

In this homework/lab exercise we are going to tie the GPR File, the MIPS ALU and the Fetch unit together to form an Rtype MIPS CPU.

Below we see a simplified drawing of the Rtype MIPS CPU we used in HW3.

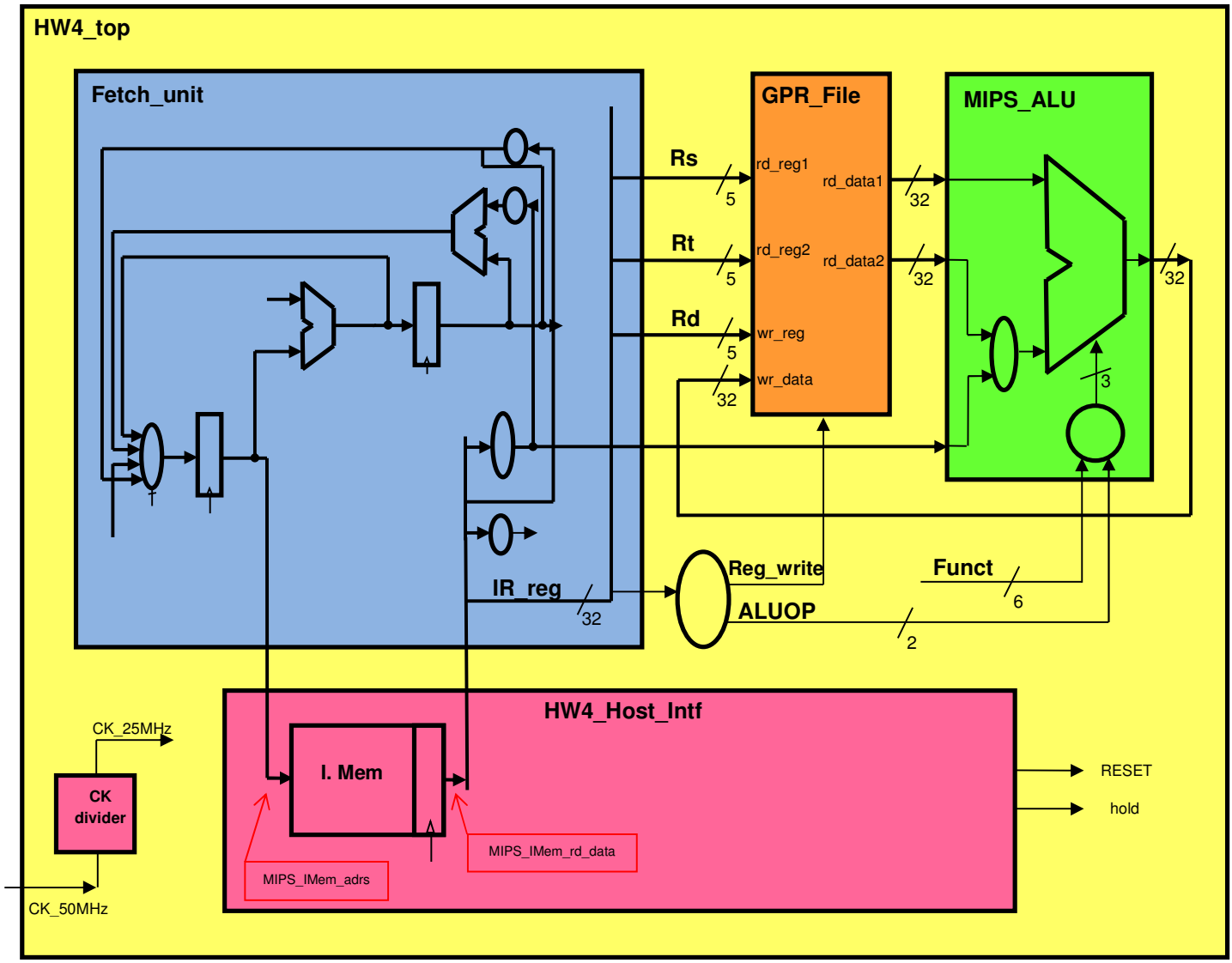


Fig. 1 – The Rtype only MIPS CPU – a simplified drawing

In HW3 we called this CPU the Rtype only MIPS. However, in the Fetch Unit we already have the ability to support jump and branch instructions. Supporting **beq** and **bne** instructions might require some minor additions. In order to make things more interesting, we will also support the **addi** instruction. Thus, this "Rtype" MIPS CPU will start running from address 400000h and preform **Rtype** instructions and also **j**, **beq**, **bne** and **addi** instructions.

Some changes in the Fetch Unit are necessary to "tailor" it into the Rtype MIPS CPU. Our design of the Rtype MIPS CPU resides in the **HW4_top.vhd**.

A more accurate description appears in Figure 2 below.

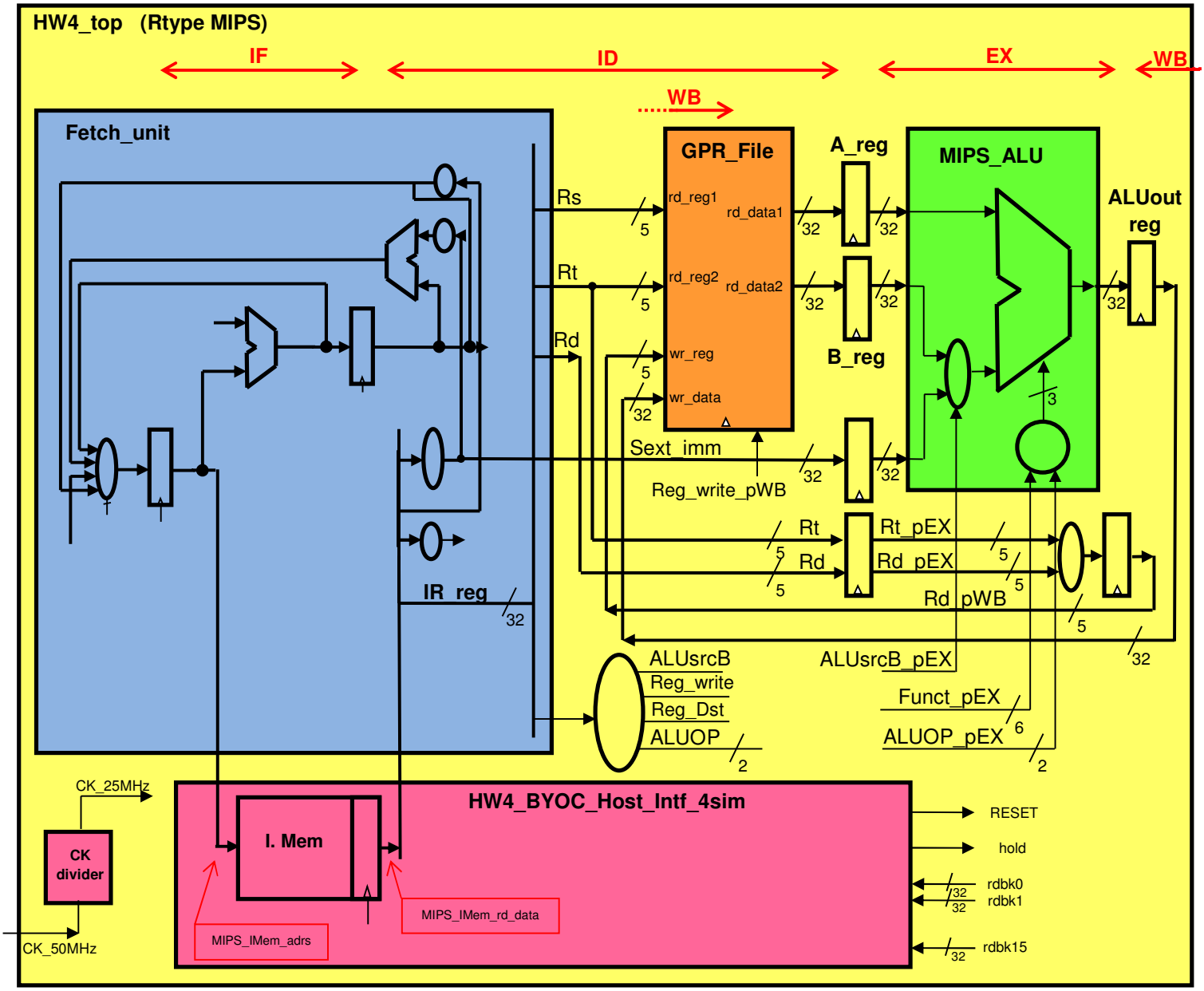


Fig. 2 – The Rtype MIPS or HW4_MIPS CPU

2) HW4 Rtype MIPS CPU – design & simulation

The HW4 Rtype MIPS CPU will have four phases.

- **IF** – Instruction Fetch, which is carried out inside the Fetch Unit producing the instruction in the IR_reg at the rising edge of the clock which ends the IF phase and starts the ID phase.
- **ID** – Instruction Decode, which is the stage in which we do the following:
  - Decode the instruction residing now at the IR_reg and decide what should be done.
    This means, we produce all control signals to be used by that instruction in all phases of this instruction – ID, ED and WB.
  - Read Rs into A_reg and Rt into B_reg

  The rising edge of the clock sampling data into the A_reg and B_reg ends the ID phase and starts the EX phase.
- **EX** – Execute, which is the phase in which the ALU calculates the result of A op B (in **Rtype** instructions) or A+sext_imm (in **addi** instructions). The result is sampled into the ALUout_reg at the rising edge of the clock which ends the EX phase and starts the WB phase.
  In this phase we also select Rs or Rd as the GPR file destination register to be written into in the Write Back phase.
- **WB** – Write Back, which is the final phase of the instruction. If this is an **Rtype** or **addi** instruction, then we write the ALUout_reg value into the GPR file. If this is a **j**, **beq** or **bne** instruction, we do nothing at that stage. The rising edge of the clock sampling data into the GPR File ends the WB phase and completes the instruction.

As explained above, the control signals are created by decoding the instruction residing in the IR_reg at the ID phase. If the control signal is supposed to influence at the EX phase, it must be delayed by 1 clock cycle. If that control signal is supposed to influence at the WB phase, it must be delayed by 2 clock cycles. You will have to handle these timing issues in order to make your design function properly.

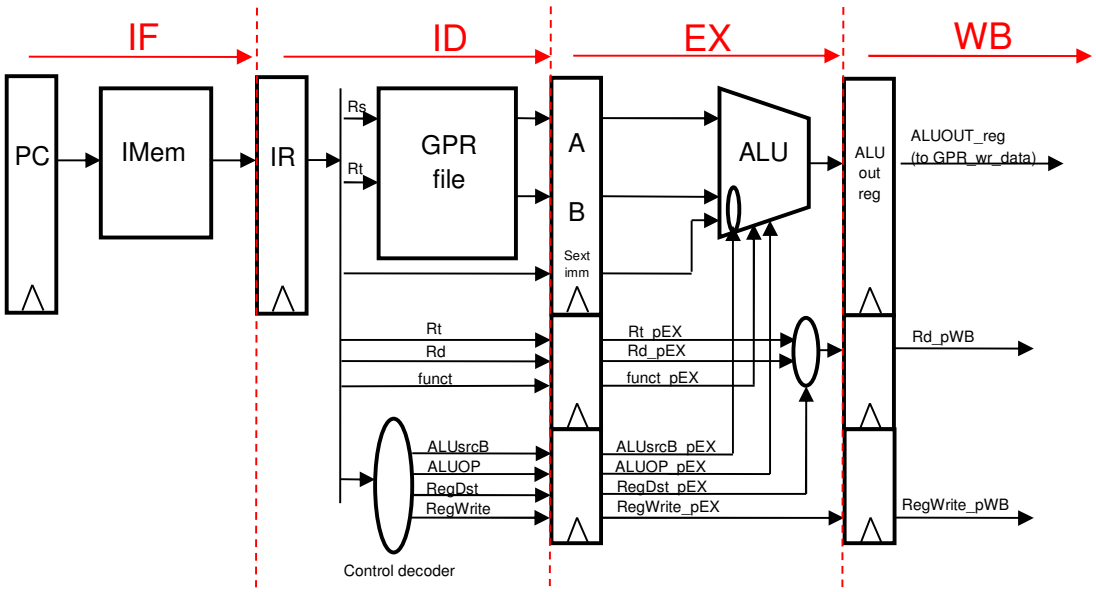


Fig. 2b – The Rtype MIPS control scheme

a. Modifications required in the Fetch Unit

We do the following changes in the Fetch_Unit entity so it will be possible to use it in the HW4_TOPdesign. See Figure 3 on the next page.

We remove all rdbk0-15 output signals from the Fetch Unit. We hope we won't need them since the Fetch Unit is already debugged and the changes we introduce are minor.

Instead we add output signals coming out of the Fetch_Unit that should be used by the rest of the CPU. These *output* signals are:

1. IR_reg_pID - This is a 32 bit signal of the IR_reg (the instruction bits). We added pID to that signal name to indicate it is the IR_reg value at the ID phase.
2. sext_imm_pID - Similarly, this is the 32 bit sext_imm signal we calculate at the ID phase. It is outputted from the Fetch Unit to be used later in the EX phase.
3. PC_reg_pIF - this is the 32 bit PC_reg we use during the IF phase for the Instruction Fetch, i.e., for reading from the IMem. It is outputted from the Fetch Unit to be used for verification purposes only (debugging).

These signals are used in the **HW4_top** entity. They also allow us testing the IR_reg and sext_imm (and the PC_reg) during simulation. Our Fetch_Unit stays the same for simulation & implementation – no changes are required when going from the simulation phase to the implementation phase. Note that for TB purposes we output the CK_out_to_TB, RESET_out_to_TB, HOLD_out_to_TB signals from the **HW4_top_4sim.vhd** which in HW4 is our top component. Therefore, when going from simulation to implementation, we will need to change the **HW4_top** and remove these signals.

Now we add an input signal to the updated Fetch_Unit.

1. We add the Rs_equals_Rt_pID signal that tells us whether to branch in beq (if it is '1') or not (if it is '0'). This signal should come from comparing the two data outputs of the GPR File which resides outside the Fetch_Unit. You should modify the PC_source signal decoder so that the **beq** and **bne** instructions are properly performed. Make sure that theh **addi** instruction is also supported.

The rest of the Fetch_Unit signals are left unchanged. See Fig. 3 below for the updated Fetch_Unit with the new signals in **RED**

When simulating our top file is **HW4_top_4sim.vhd**. In this entity we will use the **BYOC_Host_Intf_4sim.vhd** as our Host Interface circuit having the pre-loaded IMem. For implementation our top vhd file will be renamed to **HW4_top.vhd** and inside it, we will use the **BYOC_Host_Intf.ngc** file. The difference between the two Host_Intf versions is that in the sim version the Host Interface has the program already loaded inside (actually it is loaded at the beginning of the simulation). The implementation version includes the real Host_Intf mechanism allowing us to load a program from the PC, run the design in single clock mode and see the readback signals. The difference between the **HW4_top_4sim.vhd** and the **HW4_top.vhd** will be minor - removal of TB signals.

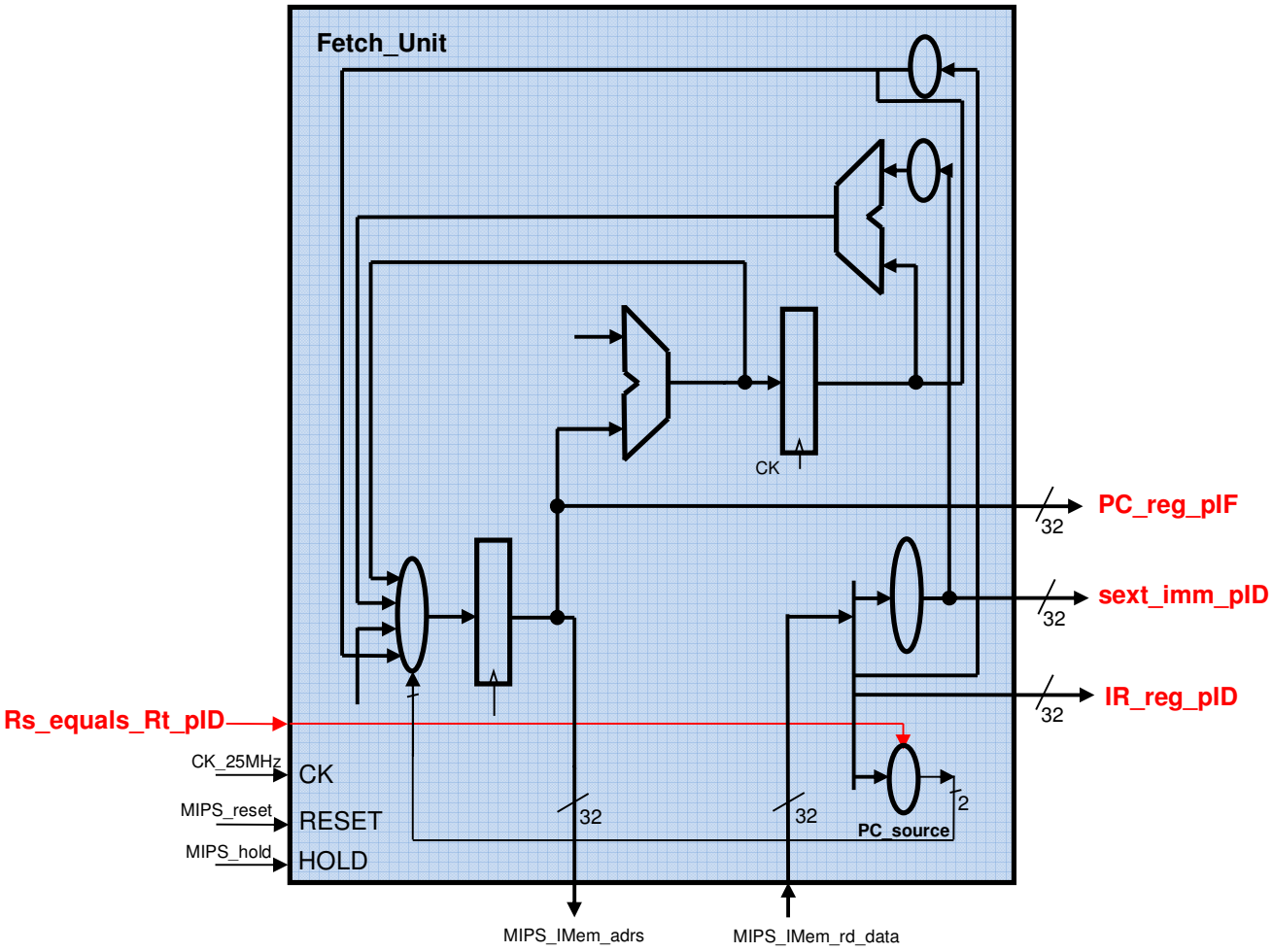


Fig. 3 – The updated Fetch_Unit (new signals – in red)

Note that in the **HW4_top_4sim.empty** file we already connected all of the components (Fetch_Unit, GPR, MIPA_ALU, BYOC_Host_Intf – those are the blue, orange, green and pink parts of Fig. 2). We also defined all of the HW4_top signals (see in section **b** below). Your job is therefore to rename it to **HW4_top_4sim.vhd** and build the missing "logic" in the **HW4_top_4sim.vhd** (which is the yellow part in Fug. 2). That "logic" is made of the registers, FFs and combinational logic forming the Rtype MIPS CPU.

b. Names & definition of signals inside the Rtype CPU (HW4_top.vhd)

You must use these exact signal names in your design.

General signals in HW4_TOP

1. CK - The 25 MHz clock coming out of the Clock_driver.
2. RESET – The MIPS_reset signal coming out of the Host_Intf and is used as reset signal to all registers
3. HOLD – The MIPS_hold signal coming out of the Host_Intf and is used to freeze writing into all FFs & registers in the Rtype MIPS design.

ID phase signals in HW4_TOP

4. IR_reg- a 32 bit register that has the instruction we read from the IMem. This signal is a rename of the IR_reg_pID signal coming out of the modified Fetch Unit
5. Opcode – the 6 MSBs of IR_reg. To be decoded and produce the necessary control signals.
6. Rs – IR[25:21].
7. Rt – IR[20:16] .
8. Rd – IR[15:11].
9. Funct – IR[5:0].
10. sext_imm – renaming of sext_imm_pID coming out of the Fetch Unit.
11. GPR_rd_data1 – the 32 bit output of the rd_data1 of the GPR and input to A_reg.
12. GPR_rd_data2 – the 32 bit output of the rd_data2 of the GPR and input to B_reg.
13. Rs_equals_Rt – '1' if GPR_rd_data1== GPR_rd_data2, and '0' otherwise. Used in branch instructions. That signal will be sent to the Fetch Unit after renaming to Rs_equals_Rt _pID.

ID control signals in HW4_TOP- These are created from decoding the opcode:

14. ALUsrcB – '1' when sext_imm is used (in addi insruction).
15. ALUOP – a 2 bit signal. "10" will cause the ALU to follow the Funct field, thus it is used for Rtype instructions. "01" will cause a subtraction (We used it for beq and bne instructions to be consistent with non-pipelined MIPS implementation. Here it is not really needed since we use Rs_equals_Rt in branch operation). "00" causes addition. We will use that combination for all other instruction (including addi where addition is definitely required)
16. RegDst – '0' when we WB according to Rt (addi inst.) '1' when we WB according to Rd (Rtype inst.) We should make sure that it is '1' in Rype instructions only.
17. RegWrite – '1' when we WB (Rtype or addi inst.), '0' when we don't (j, beq & bne inst.)

EX phase signals in HW4_TOP

18. A_reg – a 32 bit register receiving the GPR_rd_data1 signal. Its value is used in the EX phase
19. B_reg – a 32 bit register receiving the GPR_rd_data2 signal
20. sext_imm_reg – a 32 bit register receiving the sext_imm coming from the Fetch Unit
21. Rt_pEX – Rt delayed by 1 clock cycle
22. Rd_pEX – Rd delayed by 1 clock cycle
23. ALU_output – a 32 bit signal of the output of the ALU (renaming of ALU_out signal coming out of the MIPS_ALU component). It is used as the input to ALUout_reg.

EX phase control signals in HW4_TOP

24. ALUsrcB_pEX – ALUsrcB delayed by 1 clock cycle.
25. Funct_pEX – Funct delayed by 1 clock cycle.
26. ALUOP_pEX – ALUOP delayed by 1 clock cycle.
27. RegDst_pEX – RegDst delayed by 1 clock cycle.
28. RegWrite_pEX – RegWrite delayed by 1 clock cycle.

WB phase signals in HW4_TOP

29. ALUout_reg – a 32 bit register getting the ALU_output signal at its input.
30. Rd_pWB – the output of RegDst mux selecting to which register we write in WB phase.

WB phase control signals in HW4_TOP

31. RegWrite_pWB – RegWrite_pEX delayed by 1 clock cycle.

Note that there are no IF signals (actually we do have the IMem address & rd_data signals and PC_reg_pIF). This is so since all IF signals are handled within the Fetch Unit. Some of the ID phase is also handled inside the Fetch Unit. That part includes the branch and jump addresses calculation, the sign extension of the imm and the creation of the PC_source signal.

c. Names & definition of output signals from HW4_top_4sim to the TB

You need to define all output signals coming out of the HW_top_4sim entity to be tested by the TB:

1) CK_out_to_TB - a signal identical to the CK "internal" signal (i.e., the CK_25MHz signal)
2) RESET_out_to_TB - a signal identical to the RESET "internal" signal
3) HOLD_out_to_TB - a signal identical to the HOLD "internal" signal
4) rdbk0_out_to_TB to rdbk15_out_to_TB – 16 vector signals, 32 bit each that will have the data we want to check – as detailed below.

In your design you should connect the rdbk signals as follows:

ID signals:

rdbk0 => PC_reg (PC_ref_pIF from the Fetch Unit))
rdbk1 => IR_reg,
rdbk2 => sext_imm (of the ID phase)
rdbk3 => Rs, Rt, Rd, Funct (Rs= bits 28:24, Rt= bits 20:16, Rd= bits 12:8, Funct= bits 5:0)
rdbk4 => RegWrite, Rs_eq_Rt, (Reg Write= bit 28, Rs_eq_Rt=bit0)
rdbk5 => GPR_rd_data1
rdbk6 => GPR_rd_data2

EX signals:

rdbk7 => ALUsrcB_pEX, ALUOP, Funct_pEX (ALUsrcB=bit 28, ALUOP= bits 9:8, Funct_pEX = bits5:0)
rdbk8 => A_reg,
rdbk9 => B_reg,
rdbk10 => sext_imm_reg,
rdbk11 => ALU_output,

WB signals:

rdbk12 => ALUOUT_reg
rdbk13 => Rd_pWB, RegWrite_pWB, (Rd= bits 20:16, RegWrite= bit 0)

The rdbk signals are connected to the TB signals of rdbk0_out_to_TB - rdbk15_out_to_TB and also to the **BYOC_Host_Intf**. During simulation the TB we prepared reads a data file called **SIM_HW4_TB_data.dat** which contains the values we expect to get from these lines and compares the file values to the rdbk0_out_to_TB-rdbk15_out_to_TB values. The rdbk0-15 signals

to the **BYOC_Host_Intf** will be used in the implementation phase for debugging of the actual circuit.

d. Description of the HW4_top_4sim project

You need to define all signals in your design. Actually, this is already done for you in the already prepared **HW4_top_4sim.empty** file. Then, you should write the equations of all of the signals and registers and connect all of the components.

So the files we will use to run the simulation are:

The design files:

1) **HW4_top_4sim.vhd** – This is your design of HW4. It uses the **GPR**, **MIPS_ALU** the updated **Fetch_Unit**, the **BYOC_Clock_driver** and the **BYOC_Host_Intf_4sim** components and all of the signals described in 2b above.
2) **GPR.vhd** – your GPR File design you prepared in HW3.
3) **dual_port_memory.vhd** – part of the GPR File design you prepared in HW3.
4) **MIPS_ALU.vhd** – your MIPS_ALU design you prepared in HW3.
5) **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after the modifications detailed in section 2a above.

BYOC infrastructure files:

6) **BYOC_Clock_driver_4sim.vhd** – the CK divider & driver we use for simulation (also good for the Modelsim simulator)
7) **BYOC_Host_Intf_4sim.vhd** – The prepared components including the IMem and "pre-loaded" program and creating the reset & ck signals.

Simulation files:

8) **SIM_HW4_TB_for_students.vhd** - The TB vhd file prepared in advance
9) **SIM_HW4_TB_data.dat** – this is a data file prepared in advance that is read by the HW4_TB and used to compare the simulation results to the expected ones.
10) **SIM_HW4_program.dat** - The program file for simulation
11) **SIM_HW4_filenames.vhd** - The actual path information of the two dat files
    NOTE: You should update that according to your simulation project actual path.

During simulation, the TB we prepared reads a data file containing the expected signal values and compares them to the actual signal values coming out of your **HW4_top_4sim** design and reports errors to the simulation console screen.

We prepared an "empty" vhd file you should use as the skeleton for your design. It is called **HW4_top_4sim.empty**.

You should take the **GPR**, **MIPS_ALU**, **Fetch_Unit**, and the **dual_port_memory** vhd files from your previous homework exercise. The **Fetch_Unit** must be updated as explained earlier.

3) Simulation report

You should submit a single zip file for the Simulation and implementation phases. It should have two directories/folders. The first is called **Simulation**, the 2nd is called **Implementation**. In the **Simulation** folder you will have 3 sub-folder of:

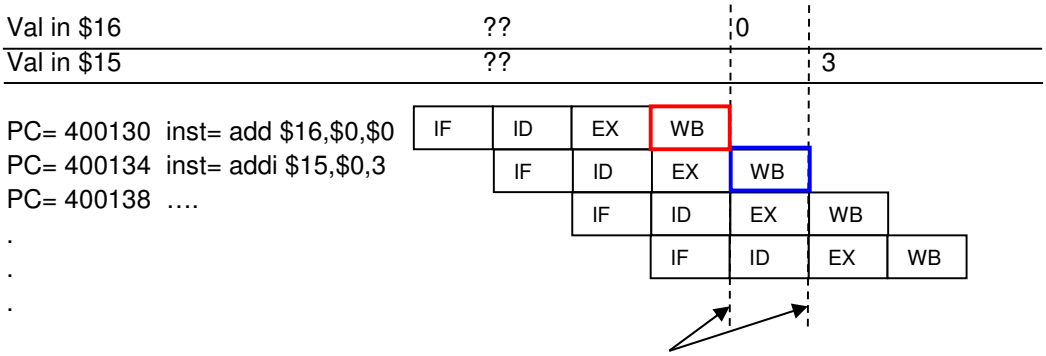
- **Src_4sim** – here you put all of the *.vhd sources and the *.dat file (to be used by the the TB)
- **Sim** – here you should have the HW4_4sim project created by the simulator you used
- **Docs** – Here you put your simulation report. The first few lines in the report will have your ID numbers (names are optional). See the instructions below for the rest of the simulation report. ***Note that this should be a WORD file and not a PDF - to allow addition of remarks while grading the report***.

In the doc file you also need to attach screen captures describing the simulation you made (see 3.1 below). You should run the simulation for 6us (6000ns).

In that doc file you need to answer the following requests and questions:

3.1) Attach screen captures showing ck cycles 70 to 118 (following the end of the reset pulse – use the i signal value of the TB to count the CK cycles) and make the values of all signals connected to rdbk0-15 readable. All of the signals mentioned in 2.c should be presented in the screen capture. The signal names should be readable as well. These are not TB_rdbk0 - TB_rdbk15 of the TB or rdbk0_out_to_TB-rdbk15_out_to_TB of HW4_top_4sim, but PC_reg, IR_reg etc. You may use the signals rdbk3_vec, rdbk4_vec, rdbk7_vec & rdbk13_vec.

3.2) Explain in details what you see in this screen capture: You should show the execution of the program using a scheme as described below. Specifically you should show the values of registers $16 and $15 and to specify the point in time where they changes (in a similar manner to the drawing below)



Put a line like that when $16 or $15 are updated

Also mark the ALUOUT_reg value at the WB stage when $16 is written to in **RED** (in a similar manner to the drawing above) on the screen capture. And mark the ALUOUT_reg value at the WB stage when $15 is written to in **BLUE**.
Also mark the IR value of branch instructions in **ORANGE** and state whether we branch or not at that instruction. You can use Excel for drawing this and attach the XL file in the Doc directory.
The purpose of this is to make you familiar in looking at the waveforms and understanding what happens in the design via simulations. This knowledge is required in HW5.

3.3) Explain in detail the changes you did in the Fetch_unit to support beq and bne instructions. Attach the relevant vhdl code.

3.4) Why do we have two nops in the following code (which is part of the code use in the simulation phase):

```
x"004000A0"   =>   x"00230820"
x"004000A4"   =>   x"00000000"
x"004000A8"   =>   x"00000000"
x"004000AC"   =>   x"00810820"
x"004000B0"   =>   x"00000000"
x"004000B4"   =>   x"00000000"
x"004000B8"   =>   x"00A10820"
x"004000BC"   =>   x"00000000"
x"004000C0"   =>   x"00000000"
x"004000C4"   =>   x"00260820"
```

3.5) The program we load into the IMem checks many functions & instructions. We do check the sign extension for example. Which of the instructions or functions of the Rtype MIPS CPU you built are <u>not</u> checked by the program we have in the IMem (the program we dis-assembled in the report of HW3 and which is part of the HW4_BYOC_Host_Intf_4sim.vhd file) ?

Later, in the Implementation phase you will add 3 sub-folders to the **Implementation** folder:

- **Src_4ISE** – here you put all of the *.vhd sources and the *.ucf file (and no TB file)
- **ISE** – here you should have the HW4_TOPproject created by the Xilinx ISE SW.
- **Docs** - Here you put your implementation report. The first few lines in the report will have your ID numbers (names are optional). See instructions in section 5 below.

4) HW4 - Rtype only MIPS CPU - implementation

After a successful simulation we want to implement the design on the Nexys2 borad. A few changes are required. First we will take our **HW4_top_4sim.vhd** file and rename it to **HW4_top.vhd**. Then we remove all of the TB signals we outputted for simulation. These are: CK_out_to_TB, RESET_out_to_TB, HOLD_out_to_TB & rdbk0_out_to_TB to rdbk15_out_to_TB. Just to be on the safe side, we also prepared the **HW4_top.empty** that is similar to the **HW4_top_4sim.empty** with the above mentioned changes. You may copy your HW4_top_4sim design to the **HW4_top.empty** we prepared for you, or use the two empty versions to see the changes needed to be done on the **HW4_top_4sim.vhd** file in order to produce the **HW4_top.vhd** one.

Now you should replace the **BYOC_Host_Intf_4sim.vhd** with a component that looks the same, the **BYOC_Host_Intf.ngc**, which inside is prepared for implementation instead of for simulation. It means that this component includes the circuitry that allows the PC to load data into the IMem instead of the **Host_Intf_4sim.vhd** which had a pre-loaded program "hardwired" inside and was used for simulation. Data will be loaded via the RS232 channel from the PC by the **BYOCInterface** SW into the program memory.

The files we will use to implement the design on the Nexys2 board are:

1) **BYOC.ucf** - The file listing which signal are connected to which FPGA pins in the Nexys2 board.
2) **HW4_top.vhd** – This is your design of HW4. It uses the **GPR**, **MIPS_ALU**, **Fetch_Unit** , **Clock_Driver** and the **BYOC_Host_Intf** components and all the signals described in 2b above.
3) **GPR.vhd** – your GPR File design you prepared in HW3.
4) **dual_port_memory.vhd** – part of the GPR File design you prepared in HW3.
5) **MIPS_ALU.vhd** – your MIPS_ALU design you prepared in HW3.
6) **Clock_driver.vhd** – the CK divider & driver we also used in HW2.
7) **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after the modifications detailed in section 2a above.
8) **BYOC_Host_Intf.ngc** – This prepared components including the implementation of Host Interface allowing us to load programs into IMem and read feedback signals in single clock mode.

In the next page we describe the connections of the rdbk signals used in the implementation phase.

In the HW4_top you should connect the rdbk signals to the BYOC_Host_intf as follows:

ID signals:
rdbk0 => PC_reg (PC_ref_pIF from the Fetch Unit))
rdbk1 => IR_reg,
rdbk2 => sext_imm (of the ID phase)
rdbk3 => Rs, Rt, Rd, Funct (Rs= bits 28:24, Rt= bits 20:16, Rd= bits 12:8, Funct= bits 5:0)
rdbk4 => RegWrite, Rs_eq_Rt, (Reg Write= bit 28, Rs_eq_Rt=bit0)
rdbk5 => GPR_rd_data1
rdbk6 => GPR_rd_data2
EX signals:
rdbk7 => ALUsrcB_pEX, ALUOP, Funct_pEX (ALUsrcB=bit 28, ALUOP= bits 9:8,
Funct_pEX = bits5:0)
rdbk8 => A_reg,
rdbk9 => B_reg,
rdbk10 => sext_imm_reg,
rdbk11 => ALU_output,
WB signals:
rdbk12 => ALUOUT_reg
rdbk13 => Rd_pWB, RegWrite_pWB, (Rd= bits 20:16, RegWrite= bit 0)

These are the exact connections we used in the simulation phase. Then they were outputted to the TB and to the **BYOC_Host_Intf**. Now they are connected only to the **BYOC_Host_intf**. Thes connections are required so that we could read the signals during actual running of the design and display them on the PC monitor.

In order to load the MIPS IMem with data, and in order to read data from desired points in the design we use the **BYOCInterface** SW. This SW can communicate with the **BYOC_Host_Intf** component via a RS232 cable connected from the PC to the Nexys2 board.

So we'll run that SW. Load the IMem. Then run the circuit in a single ck mode and check that the reading we see at the points we "hooked" to the rdbk signals are as what we expect.

The file we want to load into the IMem is called "**HW4_IMem_load.txt**". The file itself includes all the information required in order to load it into the IMem and switch to a single ck mode. Following the loading, we can run in single ck mode and see the readback values on the PC screen after each clock.

We can compare the read data to the **HW4_compare_data_for_BYOCIntf_SW.dat** which is basically the same data we used in simulation - to see whether the data we actually get in the real circuit is the same as the expected data. For this you need to choose to compare the read data to a file and choose the **HW4_compare_data_for_BYOCIntf_SW.dat** as that file.

5) Implemetation report

The **Implementation** folder of the zip file you submit should have 3 directories:

- **Src_4ISE** – here you put all of the *.vhd sources and the *.ucf file (and no TB file)
- **ISE** – here you should have the HW4_TOPproject created by the Xilinx ISE SW.
- **Docs** - Here you put your implementation report. The first few lines in the report will have your ID numbers (names are optional).
  Note that this should be a WORD file and not a PDF - to allow addition of remarks while grading the report.

The implementation report is very short. In that report you should explain why when you run the **BYOCIntf** SW and compare in the 2nd time, you get errors in some of the first instructions.

5.1) In which instructions do we see errors? (The instruction starts when we have it in the IR)

5.2) Why does that happen? List all of the reasons.

5.3) How can we eliminate this? Give a detailed explanation (VHDL code is preferred).

As part of completing this part of the course you will have to show me how you run the design on the Nexys2 board in the lab. And maybe answer some questions.

Enjoy the assignment !!

At the end of this assignment you will have a real CPU capable of running simple programs but missing a very important part – the Data Memory. We will add that part in our next assignment.