BYOC course

Assignment #2

<u>Fetch unit</u>

1) Description of the Fetch unit

Here we design the Fetch Unit of a pipelined MIPS CPU.

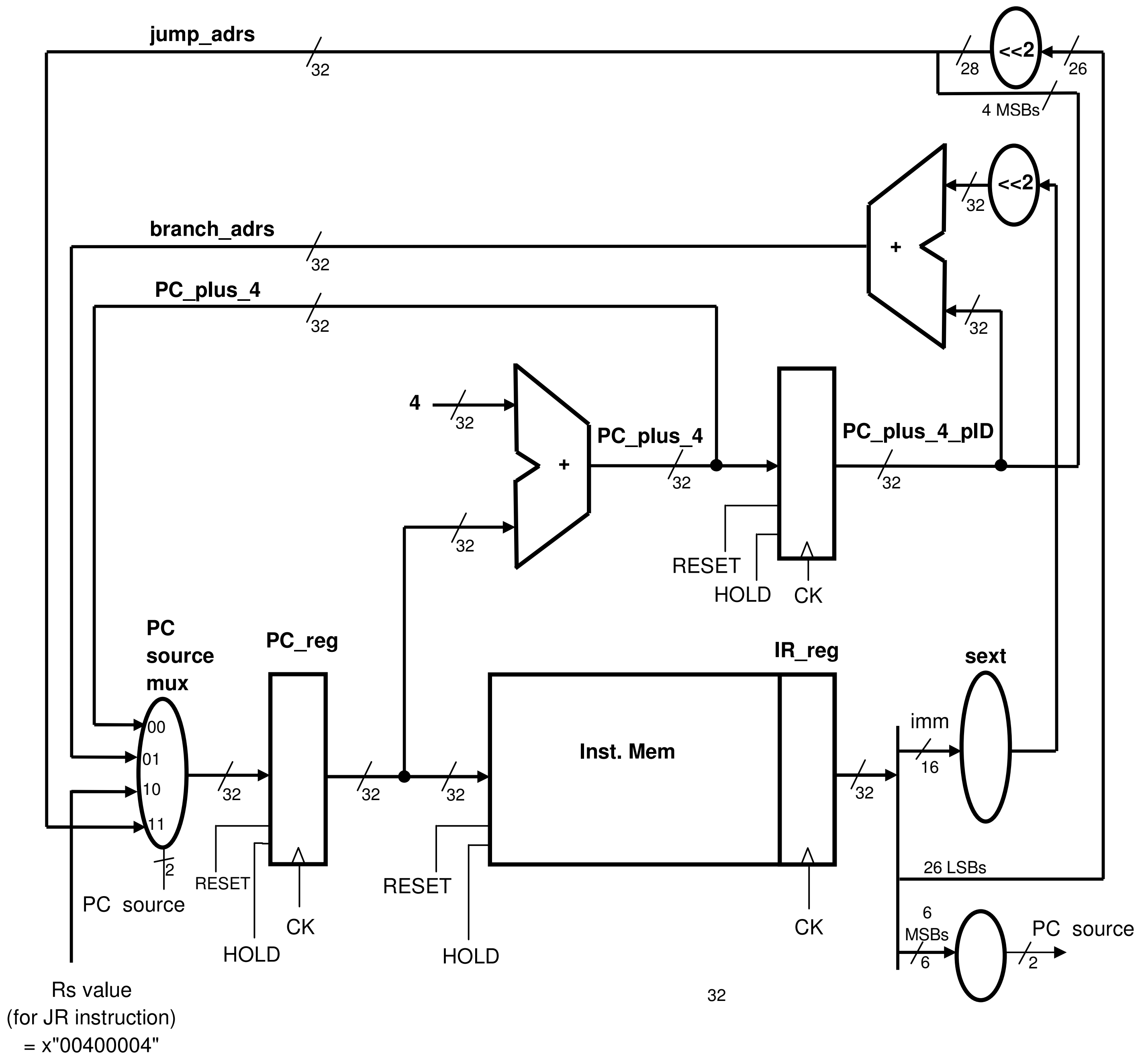

Fig. 1 – The Fetch Unit

The Fetch Unit is the part of the CPU that fetches the instruction from the Instruction Memory (IMem) into the Instruction Register (IR_reg). It also handles jumps and branches. The Fetch Unit's main components are the PC register (PC_reg) and IMem. The PC_reg is a 32 bit register that advances by 4 in every clock. Thus we should have also a 32 bit Adder that adds 4 to the current PC_reg value. In order to jump or branch, we need to input the jump address or branch address to the PC register. Thus, we have a multiplexer at the input of the PC register. This is depicted in Fig.1 above

a. Names & definition of signals inside the Fetch Unit

You must use these exact signal names in your design.

1. PC_reg – a 32 bit register. When RESET is '1', the PC_reg value becomes 0x400000. (All other registers and FFs will be cleared by RESET='1').
2. PC_plus_4 – a 32 bit signal that has the PC_reg value + 4.
3. PC_plus_4_pID – a registered version of the PC_plus_4 to be used in the ID phase. This is why we added _pID at the end of that signal name.
4. branch_adrs – a 32 bit signal which is made of PC_plus_4_pID + sext(imm)<<2. This is the address to be loaded into the PC when a successful branch is performed. Imm signal is made of the lower 16 bits of IR_reg (see IR_reg in #8 below).
5. jump_adrs – a 32 bit signal made of PC_plus_4_pID[31:28] & IR[25:0] & b"00", i.e., the jump address in words multiplied by 4. This is the address to be loaded into the PC when a jump or a jal instruction is performed.
6. jr_adrs – a 32 bit signal made of the Rs value in a JR instruction. Since we do not have a GPR file, we set the Rs value to x"00400004". In the complete CPU this will be the address to be loaded into the PC when a jr (jump register) instruction is performed.
7. PC_source – a 2 bit signal. When "00", PC_reg is loaded with PC_plus_4. When "01" it is loaded with branch_adrs, when "10" with jr_adrs (Rs value for jr instruction) and when "11", PC_reg is loaded with the jump_adrs.
   The PC_source signal is created by a decoder looking at the opcode field of the instruction residing in the IR_reg.
8. IR_reg- a 32 bit register that has the instruction we read from the IMem. This register is part of the IMem (The IMem is an already designed component we use in the Fetch Unit).
9. imm – the 16 LSBs of IR_reg
10. sext_imm – sign extension of imm to 32 bits
11. opcode – the 6 MSBs of IR_reg. We sould determine the PC_source value according to the instruction opcode (j,jal-11, beq,bne-01,jr -10, any other instruction-00).
12. HOLD – This signal is meant to freeze all registers when it is "1". It will be used later for running the design in a single clock mode. At that mode this signal will be "1" all the time except for the clock cycles in which we want to perform a single clock "step". This means that all of the registers should have a HOLD input. The IMem itself and its output register (the IR) already support that signal.

2) Fetch unit simulation project

a. Description of the Fetch_Unit projects

The Fetch_Unit you design is a new entity. In this homework assignment it is used it in two projects. The first is a simulation project meant to test the Fetch Unit design by running SW only. The second is the implementation project which should actually run on the Nexys2 board. In this section we describe the Simulation project.

The simulation project is made of 3 groups of files:

a.1. The design vhd files - In HW2 you have two main files that form the design

i. **HW2_top_4sim.vhd**
This file is the top file of HW2 design. It "connects" all of the components used together to form the design. The components used are the Fetch Unit, the BYOC_Host_Intf_4sim and the Clock_driver_4sim. The last two belong to the group of pre-prepared infrastructure files and will be the same for the simulation projects of HW2, HW4, HW5 and HW6. In these HW assignments you will always have a top file that starts with the assignment name (HW2, HW4, ... etc).

ii. **Fetch_Unit.vhd**
This is what you need to design in this HW assignment – the Fetch Unit.

a.2. Other components – Infra-structure prepared ahead of time for you by the course team
These infrastructure filenames will start with the word **BYOC**.
The infrastructure files will be used in the simulation projects of HW2, HW4, HW5 and HW6.

i. **BYOC_Clock_driver_4sim.vhd**
This small vhd file has a T-FF dividing the 50MHz clock coming from outside (in real implementation outside means the Nexys2 board, in simulation, outside means from the Test Bench "wrapping" our design and allows testing of the design). It is a modified version of the **BYOC_Clock_driver.vhd** which will be used in implementation and is modified to allow simulation with any simulator, e.g., Modelsim.

ii. **BYOCh_host_intf_4sim.vhd**
This vhd file includes the infrastructure that issues the Reset and Hold signals. It also includes the Instruction Memory [IMem] that has the program to be run in the MIPS design and the Data Memeory [DMem]. In HW2, we hook the Fetch Unit to the IMem and check whether the Program Counter increments correctly and whether the instruction data read from the IMem is correct. Thus, in HW2 we only use the IMem and will not use the DMem. The IMem part of the **BYOC_Host_Intf_4sim** is loaded automatically with the required program for HW2 when we start the simulation.
During the implementation phase, we will replace the **BYOC_Host_Intf_4sim** component with a **BYOC_Host_Intf** component that is meant to provide the same functionality. In both cases the **BYOC_Host_Intf** and **BYOC_Host_Intf_4sim** have the same i/o pins and are connected similarly in the circuit (in **HW2_top.vhd** or in **HW2_top_4sim.vhd**).

a.3. <u>Files for simulations only</u> - These files are the Test Bench and data files
These filenames will start with the word **SIM**

i. **SIM_HW2_TB.vhd**
This file includes a Test Bench [TB] design that feeds the Hw2_top_4sim with the appropriate signals (in HW2, HW4, HW5 & HW6 only the 50MHz CK is fed to the top design). It also reads feedback signals (all have suffix of "_out_to_TB"). The TB compares the feedback signals "read" from the HW2_top design during simulation, with the expected values which are read from a data file called **HW2_TB_data.dat** and reports errors to the simulator console. This eases debugging of the design.

ii. **SIM_HW2_TB_data.dat**
This is the data file having the expected readback values.

iii. **SIM_HW2_program.dat**
This file has the IMem contents which is read by the **BYOC_Host_Intf** at the beginning of simulation

iv. **SIM_HW2_filenames.vhd**
In this file we specify the path of the data files used in simulation

Figure 2 below describe the simulation project scheme:

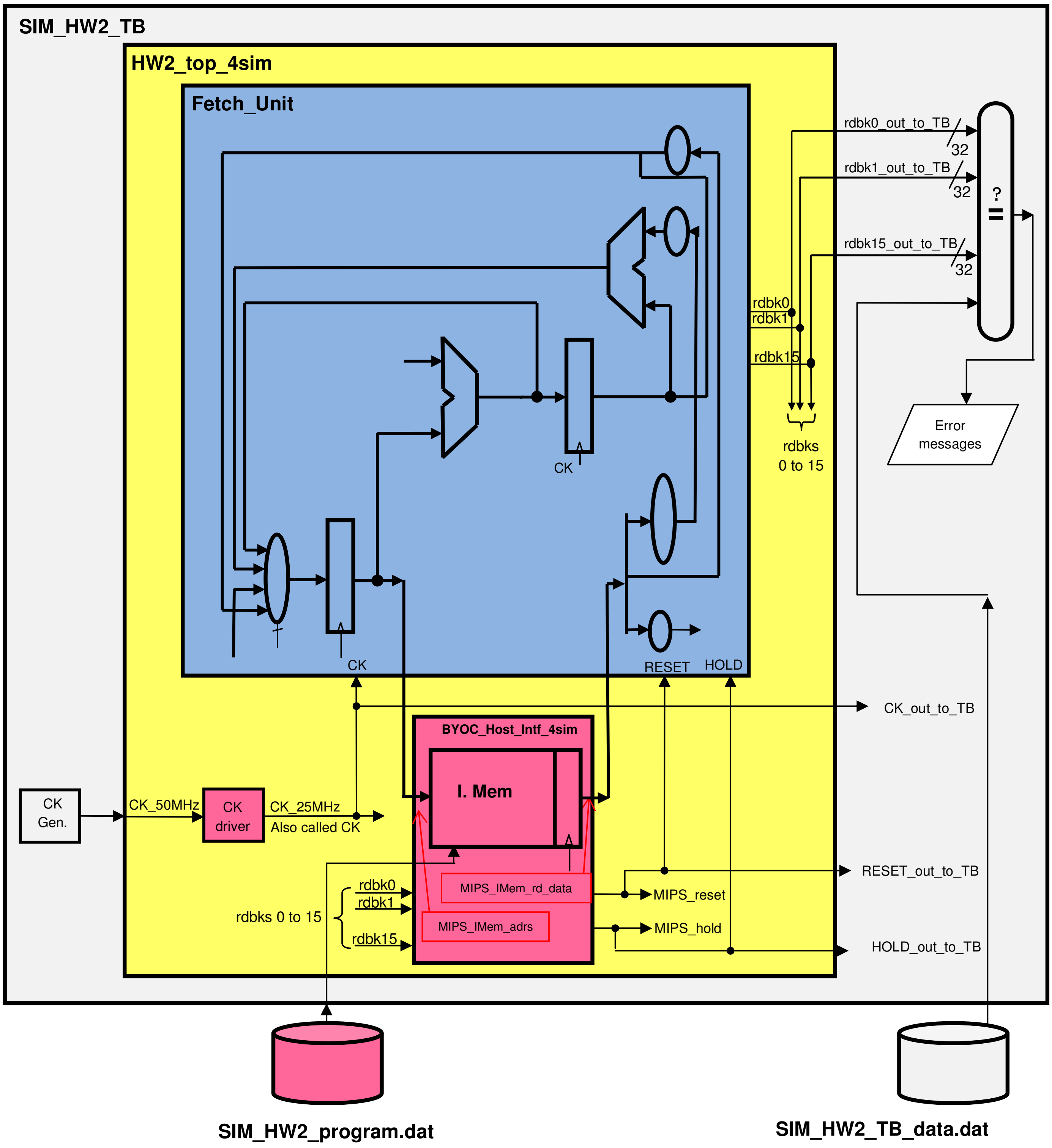

Fig. 2 – HW2 simulation project

The light blue part is the entity you design. It is called the **Fetch_Unit** and is defined in the **Fetch_Unit.vhd** file. Inside the Fetch_Unit entity you need to define all signals. Then, you should write the equations of the PC_reg, PC_plus_4 adder, branch_adrs, jump_adrs, jr_adrs, PC_source mux, sext_imm and the instruction decoder producing the PC_source signal. The MIPS instruction coding is described in Appendix A at the end of this document. Note that all of the elements shown in Fig.1 (beside the IMem) will be in the same file, the **Fetch_Unit.vhd**.
Do NOT implement these elements as entities (components)!

Your design includes all signals, registers and logic we see in Figure 1 except the IMem and IR itself. The IMem & IR are implemented for you in advance in a component called **BYOC_Host_Intf_4sim**. Another important pre-prepared component is the **BYOC_Clock_driver_4sim.vhd**. All of these 3 components (the pre-prepared two and the Fetch_Unit you designed), reside inside the design top file called **HW2_top_4sim.vhd**. It describes the connections of all of the 3 components and the signals outputted to the Test Bench. We have prepared the **HW2_top_4sim.vhd** file for you.

The **BYOC_Clock_driver_4sim.vhd** component has a T-FF that divides the 50 MHz CK input to 25 MHz output (We use CK as an abbreviation to CLOCK). In the implementation phase we use a special driver called **BUFG** inside the **BYOC_Clock_driver** that can drive many FFs and registers. The entire design is driven by the CK signal driven by the **BUFG**, i.e., this output clock of 25 MHz feeds all other components in the design. We rename it to CK and use it in the Fetch unit design.

The ISIM simulator is familiar with the special clock driver we use inside the **BYOC_Clock_driver** but the Modelsim simulator or other simulators are not. Therefore for simulation we bypassed the **BUFG** inside the **BYOC_Clock_driver_4sim.vhd** (which describes the **Clock_driver** component). In the implementation phase we will use the file that has the complete circuit with the **BUFG** inside. That file is called **Clock_driver.vhd** and has the same exact i/o pins as the **Clock_driver_4sim.vhd**.

The **BYOC_Host_Intf_4sim** component reads the required program data from a file called **SIM_HW2_program.dat** at the beginning of the simulation. Thus, for us, it has already some “program data” inside it. The 32 address lines of the IMem are called MIPS_IMem_adrs and the 32 output data lines are called MIPS_IMem_rd_data. We will drive the MIPS_IMem_adrs from the PC_reg and will direct the MIPS_IMem_rd_data lines (rename them) to the IR_reg signal.

Besides having the IMem inside it, the **BYOC_Host_Intf_4sim** component also outputs two signals that are required for the Fetch_Unit design. These are MIPS_reset and the MIPS_hold signals. You will make sure that all reset signals of all registers in your design are connected to the MIPS_reset signal. The MIPS_hold signal will drive the HOLD signal we mentioned earlier. You will make sure that every FF and register in the Fetch Unit will be “frozen” when the HOLD signal is ‘1’.

After succeeding in the simulation phase, you will replace the IMem for simulation with another version, suitable for implementation (instead of the **BYOC_Host_Intf_4sim** you will use the **BYOC_Host_Intf** component. This component has inside a mechanism enabling loading of “program” into the IMem via a RS232 cable from a PC computer as explained in the BYOC course infrastructure document). Then we will create a BIT file and load and test the design on the Nexys2 board.

These stages will be explained in detail below in the section discussing the implementation project.
Note: The **BYOC_Host_Intf_4sim** component has the same i/o pins as the **BYOC_Host_Intf** component we will use in the next phase of the project – the actual implementation phase. In that phase we use the real **BYOC_Host_Intf** component that has an IMem that can be loaded with any program via the PC. That component therefore includes a hidden mechanism that can communicate with the PC via RS232 interface. The hidden mechanism has additional features. It is connected to the 8 switches and 4 of the push-buttons existing on the Nexys2 board. There are more hidden mechanisms inside the **BYOC_Host_Intf** component we will need to use later in the course. This means that the **BYOC_Host_Intf** component has additional i/o pins which we did not mention yet. We built the **BYOC_Host_Intf_4sim** component with the same i/o pins as the **BYOC_Host_Intf** component. So we have these additional signals also in our simulation design although we will not use them in the simulation phase. Take a look in the BYOC course infrastructure document describing the **BYOC_Host_Intf** component.

The Test Bench (TB) design "wraps" the **HW2_top_4sim** design and supply the required test-signals. In our case that is the CK_50MHz signal only. The TB should also look at some output signals from the **Fetch_Unit_4sim** (we make sure that they go through the **HW2_top_4sim**) so it can test them and verify their correctness. When running simulation, you can see on the screen waveforms of all signals in the design, "internal" and "external". However, the TB needs to get the signals we want to check as outputs from the design under test. Some of these signals are issued by the **BYOC_Host_Intf_4sim**. Thus we will output the following signals from the **Fetch_Unit** and from the **BYOC_Host_Intf_4sim** designs:

1) CK_out_to_TB - The MIPS_ck signal (coming from the **clock_driver**)
2) RESET_out_to_TB - The MIPS_reset signal (coming from the **BYOC_Host_Intf**)
3) HOLD_out_to_TB - The MIPS_hold signal (coming from the **BYOC_Host_Intf**)
4) rdbk0_out_to_TB to rdbk15_out_to_TB – These are 16 vectors of 32 bit each signals. We will connect these signals to all the points in the circuit we want to test.

(As a preparation for the implementation phase, we will also output these 16 vectors of 32 bit each to the **BYOC_Host_Intf**. These rdbk signals will be used later for reading signals from the implemented **Fetch_Unit** during testing/debugging).

We connected the rdbk signals as follows:

rdbk0	=>	PC_reg,
rdbk1	=>	PC_plus_4,
rdbk2	=>	branch_adrs,
rdbk3	=>	jr_adrs,
rdbk4	=>	jump_adrs,
rdbk5	=>	PC_plus_4_pID,
rdbk6	=>	IR_reg.
rdbk7	=>	PC_source (bits 1:0),
rdbk8	=>	output of PC_mux (32 bit signal),
rdbk9 – 15	=>	0x00000000.

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 **HW2_top_4sim** tested design. It reports errors to the simulation console screen. This will make it easier for you to debug your design. The name of the dat file that has the expected values is **SIM_HW2_TB_dat.dat**. You must update the actual path of this file in the **SIM_HW2_filenames.vhd** otherwise the simulator will fail reading it and abort simulation.

So the files we require to have in order to run the simulation are:

Group #1 – The design files

1) **HW2_top_4sim.vhd** – The pre-prepared top file
2) **Fetch_Unit.vhd** - your design implementing figure 1

Group #2 – The infrastructure files

3) **BYOC_Clock_driver_4sim.vhd** – The pre-prepared component that divides the clock from 50MHz to 25MHz
4) **BYOC_Host_Intf_4sim.vhd** – The pre-prepared component including the IMem and creating the reset & hold signals

Group #3 – The simulation files

5) **SIM_HW2_TB.vhd** - The TB vhd file prepared in advance.
6) **SIM_HW2_TB_data.dat** - The data file read by the TB during simulation.
7) **SIM_HW2_program.dat** - The program file loaded into IMem by the **BYOC_Host_Intf** at the beginning of simulation.
8) **SIM_HW2_filenames.vhd** - The actual path information of the two dat files so that the simulator can access these files during simulation.

We even prepared an “empty” **Fetch_Unit.empty** file in which we already did the following:

- Defined the I/O pins of the Fetch_Unit design
- Defined all necessary internal signal inside the Fetch_Unit design
- Connected other signals, e.g., rdbk signals to be outputted to the TB or **BYOC_Host_Intf**

You now have to write the equations describing the logic circuitry of Figure 1. For that we also added remarks listing the required functionality inside the “empty” file.

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** directory you should have should have 3 sub-directories:

- **Src_4sim** – here you put all of the *.vhd sources and the *.dat file (to be used by the the TB) [You should change the name of the Fetch_Unit.empty to Fetch_Unit.vhd]
- **Sim** – here you should have the HW2 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.

An important note:
Your files time-stamp should reflect your project timeline! That is, the grader should be able to see the steps you did in reaching the successful simulation.
If all of your files (source files or project file) have the same time stamp (e.g., after downloading from a server or from a MAC computer) you lose the timeline and points will be deducted generously.
If you discover an error later in the HW (e.g., at the implementation stage) you should keep the old files in Src_4sim_old1,2,.. etc., and the new ones in Src_4_sim. Then you should run the simulation again.
These instructions are relevant to all next HW reports, i.e., HW2-HW6.

Answer question 3.1 below and then run the simulation to 5500 nS and reply all other questions.

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

3.1) Draw on paper a block diagram describing the inside of HW2_top entity. Specify all connections of the sub components between themselves and between them and the i/o pins of the HW2_top entity. [This is meant to make you familiar with the i/o pin names, the signals, the port mapping of the components, i.e., to understand what is connected to what]

3.2) You need to attach a doc file with screen captures describing the simulation you made. All signals mentioned in section 1a above should be presented in the screen capture. Show at least the 1st 10 ck cycles following the end of the reset pulse and make the values of all signals readable.

3.3) Say we have a value V in location Adrs in the IMem. What will be the expected values of the PC_reg and PC_plus_4_pID signals when the IR_reg signal has the value V? Adrs? Adrs-4? Adrs+4? Other? Explain your reply.

3.4) In the IMem we have the value of a bne instruction (x"14000002") in address x”400014”. Write down all of the fields of this instruction (opcode, Rs, ..., etc.) Do we expect to actually branch? Who checks the condition?

3.5) Did we branch in the simulation? Check the simulation and write down the sequence of the instructions (the PC_reg values) from address x”400008” until 4 addresses after reaching the branch target. Did the branch occur in the address we expect it to occur in? Explain what happened.
Following this write the assembly code of the smallest possible loop and explain how it works.

Later, in the Implementation phase you will add 2 sub-folders to the **Implementation** folder.
These will be:

- **Src_4ISE** – here you put all of the *.vhd sources and the *.ucf file (and no TB file or dat files)
- **ISE** – here you should have the HW2 project created by the Xilinx ISE SW.

4) <u>Fetch unit implementation project</u>

a. Description of the Fetch_Unit implementation project

Only after a successful simulation we continue in implementing the design on the Nexys2 board. The implementation project is made also of 3 groups of files:

a.4. <u>The design vhd files</u>

In HW2 you have two main files that form the design

i. **HW2_top.vhd**
   This file is the top file of HW2 design. It is similar to the **HW2_top_4sim.vhd** file except of the removal of all signals outputted to the TB. The changes are described in detail below.

ii. **Fetch_Unit.vhd**
   This is the Fetch_Unit you designed and tested in the simulation phase.

a.5. <u>Other components</u> – Infra-structure prepared ahead of time for you by the course team

These infrastructure filenames will start with **BYOC**.

The infrastructure files will be used in the implemenation projects of HW2, HW4, HW5 and HW6.

i. **BYOC_clock_driver.vhd**
   This small vhd file has a T-FF dividing the 50MHz clock coming from the Nexys2 board, and the appropriate clock driver **BUFG** that can drive all of the design FFs and registers

ii. **BYOCh_host_intf.ngc**
   This pre-compiled file includes the infrastructure that issues the Reset and Hold signals, the IMem and the DMem and the appropriate mechanism to access them from the PC. When running the design, we use the **BYOCh_host_intf** to run the design in a single clock mode and read all readback signals from the design. We display these signals on the PC screen and can verify correctness of our design.

a.6. <u>Files for implementation only</u>

Now we do not need the TB files at all and we remove all TB related files. We do need a special file for implementation, the **BYOC.ucf** file.

i. **BYOC.ucf**
   This file specifies the connection between the HW2_top design and the FPGA i/o pins. Since we wil use the same exact i/o pins in our top of HW2, HW4, HW5 and HW6, in all of these designs we will use the same **BYOC.ucf** file.
   The initials **ucf** stand for User Constraints File. The ucf file also specifies timing constraints. All of these were already pre-prepared for you and you need only to include the **BYOC.ucf** file in your design sources.

Figure 3 below describe the simulation project scheme:

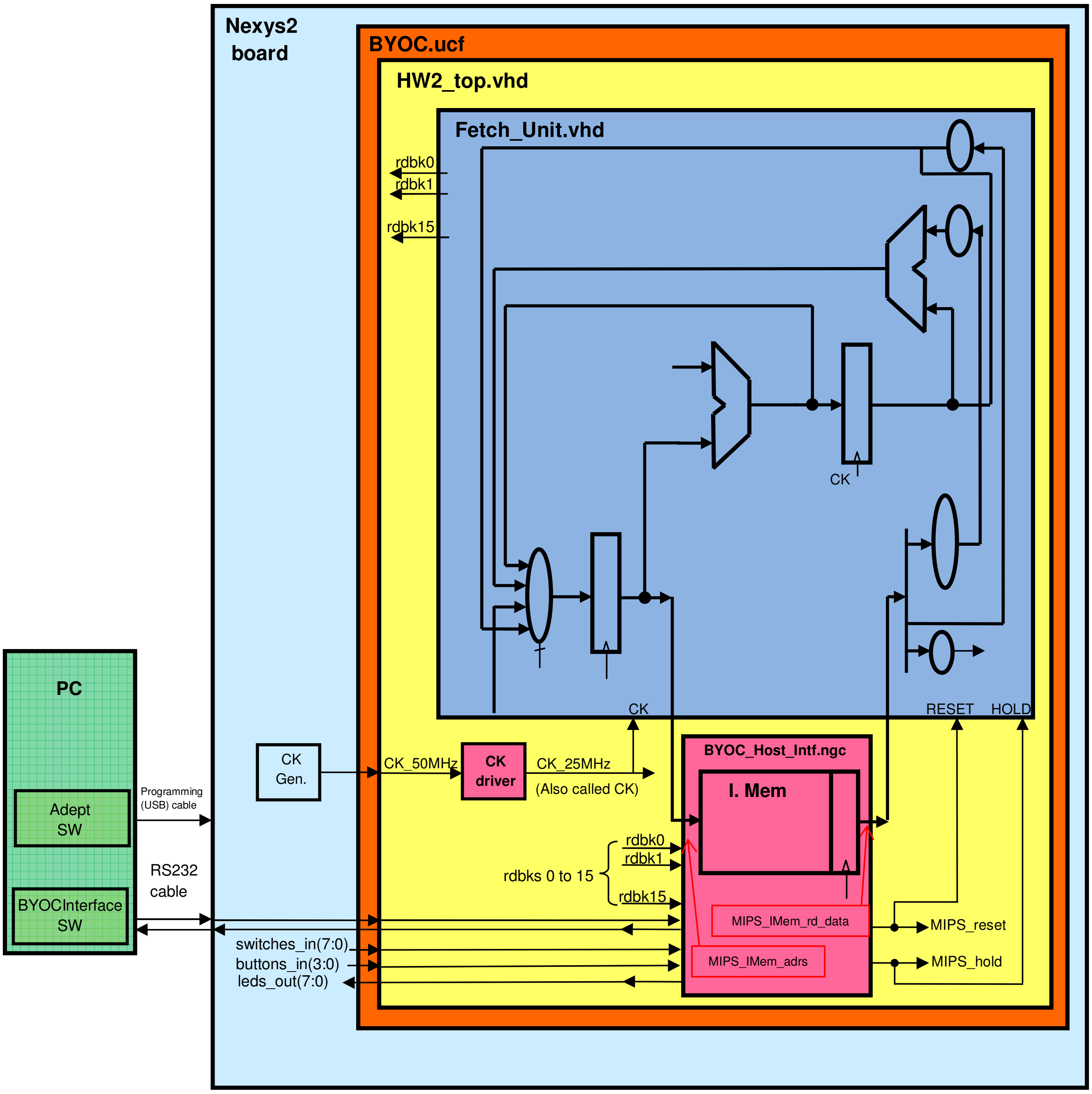

Fig. 3 – HW2 implementation scheme

A few changes are required to get to this scheme from the simulation one. First we will take our **HW2_TOP_4sim.vhd** file and rename it to **HW2_top.vhd** as well as removing the all of the signals we outputted for simulation. This removal of the signals we outputted from **HW2_top_4sim** towards the TB is a must. We do not have enough i/o pins for these 16 readback vectors of 32 bit each in the FPGA!

We also need to replace the **BYOC_Clock_driver_4sim.vhd** with the **BYOC_clock_driver.vhd** file.

The **HW2_top** component is somehow connected to the real world – i.e., to the Nexys2 board. It resides "inside" the board. We "connect" the i/o pins of the **HW2_top.vhd** to Nexys2 board by including the **BYOC.ucf** file in the project

The lfiles we need to have in order to "compile" our project are therefore:

Group #1 – The design files
1) **HW2_top.vhd** – The top file after renaming and removal of all signals outputted to the TB
2) **Fetch_Unit.vhd** - your design implementing figure 1

Group #2 – The infrastructure files
3) **BYOC_clock_driver.vhd** – The pre-prepared component that divides the clock from 50MHz to 25MHz that has a **BUFG** driver inside
4) **BYOC_Host_Intf.ngc** – The pre-compiled component including the IMem and creating the reset & hold signals. It also includes the infrastructure interfacing to the PC which is required to load the IMem and run the implemented design in a single-clock mode.

Group #3 – The implementation files
5) **BYOC.ucf** - The file listing which signals are connected to which FPGA pins in the Nexys2 board.

When we have these 5 files we can compile them on the Xilinx ISE SW and produce a **HW2_top.bit** file which is the file we load into the FPGA in order to configure it to implement our design.

When we have the **HW2_top.bit**, we can load it into the FPGA inside the Nexys2 board by connecting a USB cable from an external PC into the mini-USB input on the Nexys2 board and activating the **Adept** SW application that supports "loading" a design into the Nexys2 board. The Nexys2 board and the **Adept** SW were developed by a company called **Digilent**.

OK. We loaded the design. But what about loading the IMem with data? The design we created has an empty Instruction Memory. Furthermore, say we loaded the design with some program and run it. How do we know it really works? We need some means or tools to be hooked to the signals inside the FPGA (or to direct them to external pins in the FPGA so we can really connect to them) and "read" the values of the signals and display them somewhere (on the screen as waveforms or as data values). How do we do that?

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 and then run the circuit in a single ck mode. After each single clock, we will check that the reading we see at the points we "hooked" the rdbk signals to are as we expect.

We connected the rdbk signals as follows:

rdbk0	=>	PC_reg,
rdbk1	=>	PC_plus_4,
rdbk2	=>	branch_adrs,
rdbk3	=>	jr_adrs,
rdbk4	=>	jump_adrs,
rdbk5	=>	PC_plus_4_pID,
rdbk6	=>	IR_reg.
rdbk7	=>	PC_source (bits 1:0),
rdbk8	=>	output of PC_mux (32 bit signal).

In order to "teach" the **BYOCInterface** SW the names of these signals we have a "**labels.txt**" file in which we write the name of the signals we hooked to in the same order (The name of the signal connected to rdbk0 is written in the first line of the **labels.txt** file. The name of the signal connected to rdbk1 in is the 2nd line, etc.).

The file we want to load into the IMem is called "**HW2_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 also see the PC value on the Nexys2 board 7 segments leds.

We can also ask the **BYOCInterface** to compare the values on the screen to the expected values by selecting a comparison to a file called "**HW2_compare_data_for_BYOCIntf_SW.dat**". If we do that and run the circuit in a single clock mode we will see errors on the screen in **red**.

5) Implametation 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 Implementation directory you should have 2 sub-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 HW2 project created by the Xilinx ISE SW.

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 complete Fetch Unit which is the basis for our next designs.

6) 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)