HW5 – adding DMem

[Adding lw & sw instructions to CPU instruction set: Rtype (no jr), addi, beq, bne, j]

HW5 instruction set

(new inst. in **red**)

R-type

```
add  Rd, Rs, Rt        #  Rd=Rs+Rt
sub Rd, Rs, Rt         #  Rd=Rs-Rt
and Rd, Rs, Rt         #  Rd=Rs AND Rt
or Rd, Rs, Rt          #  Rd=Rs OR Rt
xor Rd, Rs, Rt         #  Rd=Rs XOR Rt
slt Rd, Rs, Rt         #  if Rs<Rt  Rd=1 else Rd=0
```

6	5	5	5		6
000000	Rs	Rt	Rd	00000	FUNCTION
OPCODE					

I-type

```
addi  Rt, Rs, imm      #  Rt=Rs+ sext(imm)
lw  Rt, imm(Rs)        #  Rt=M[Rs + imm]
sw  Rt, imm(Rs)        #  M[Rs + imm]=Rt
beq  Rs, Rt, label     #  if Rs==Rt,  PC=PC+4+ sext(imm)*4
                       #  else         PC=PC+4
bne  Rs, Rt, label     #  same as beq with cond of Rs≠Rt
```

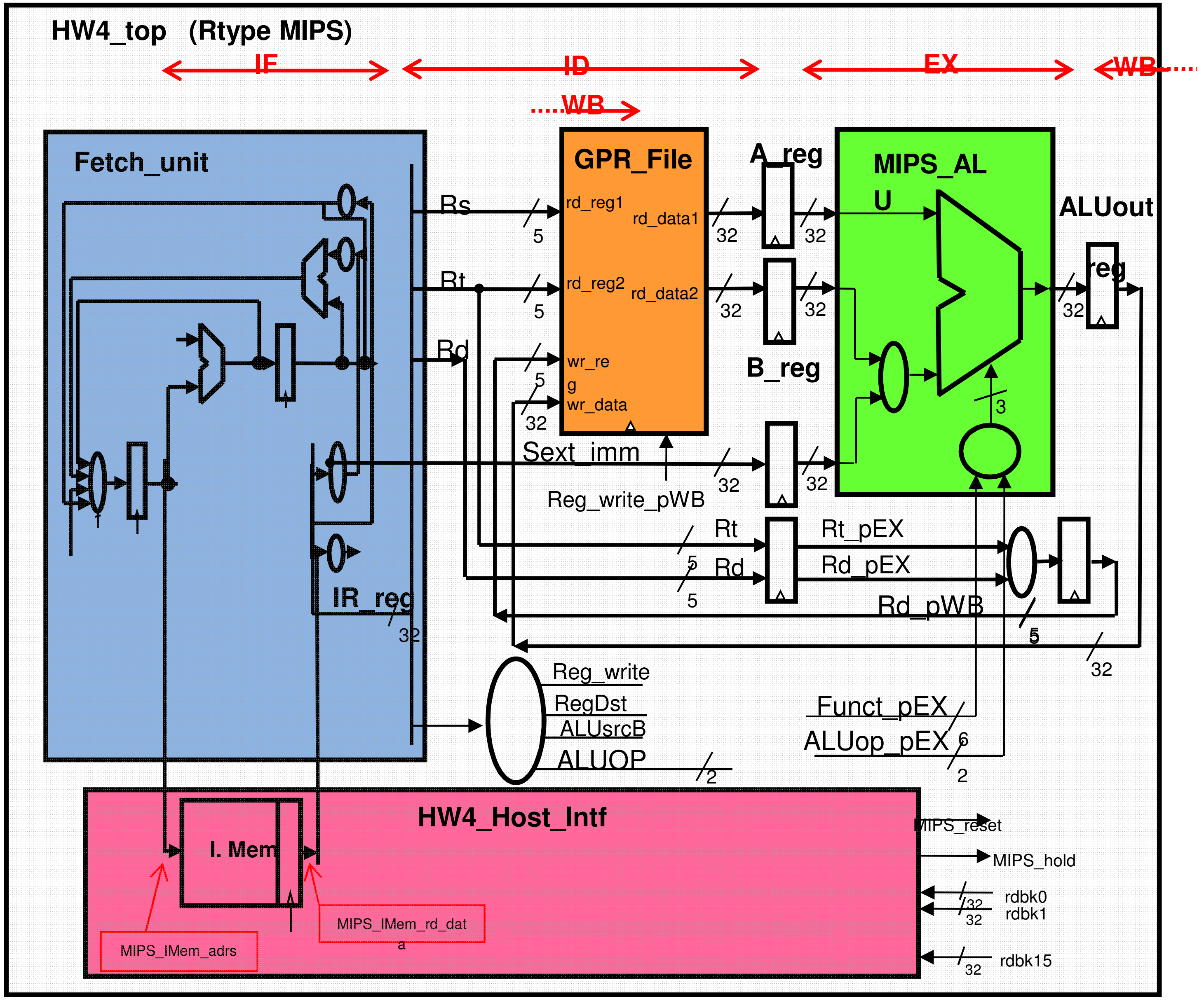
6	5	5	16
OPCODE	Rs	Rt	imm

j-type

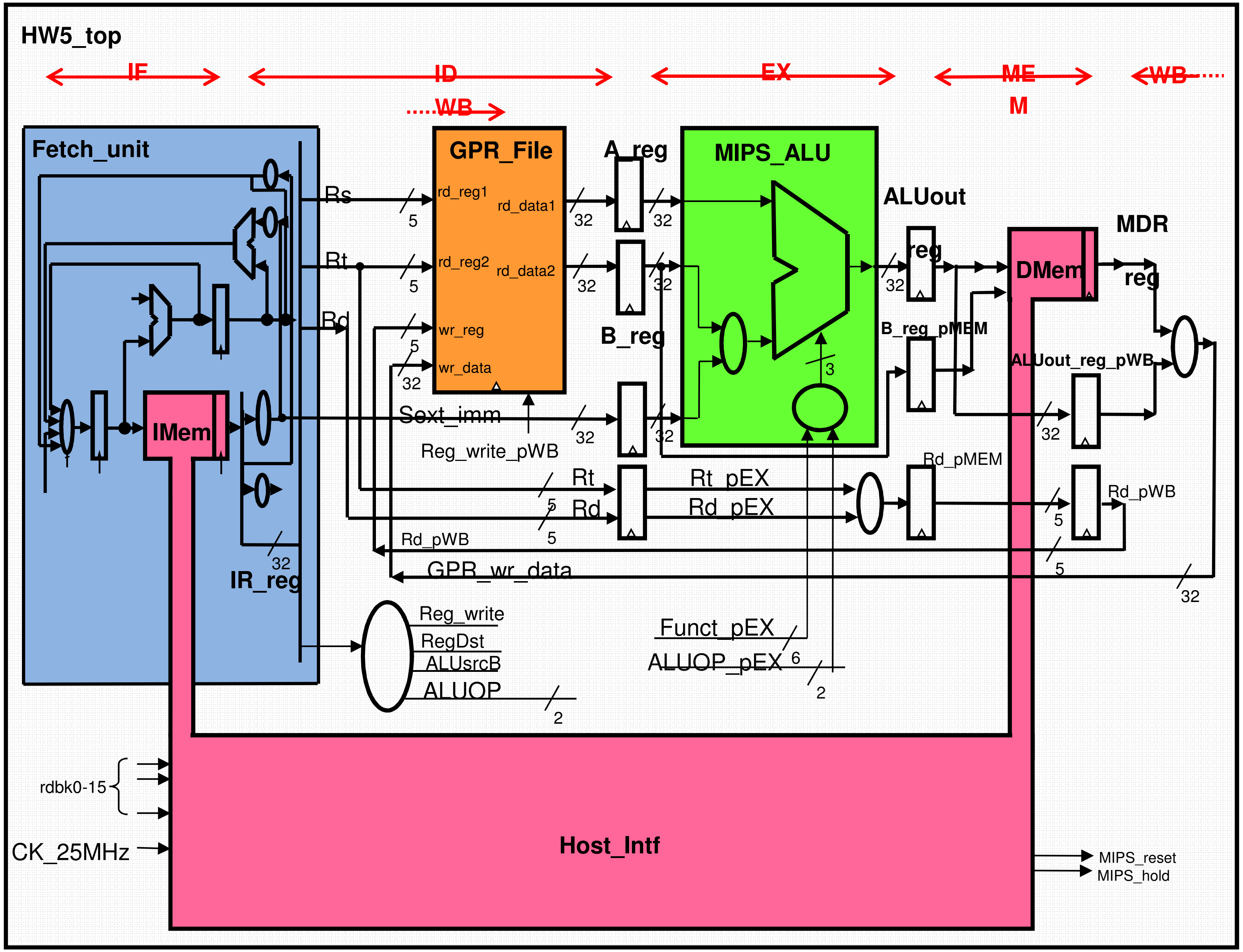
```
j    imm               #  PC= imm*4              (no sext)
```

6	26
OPCODE	26 bit imm

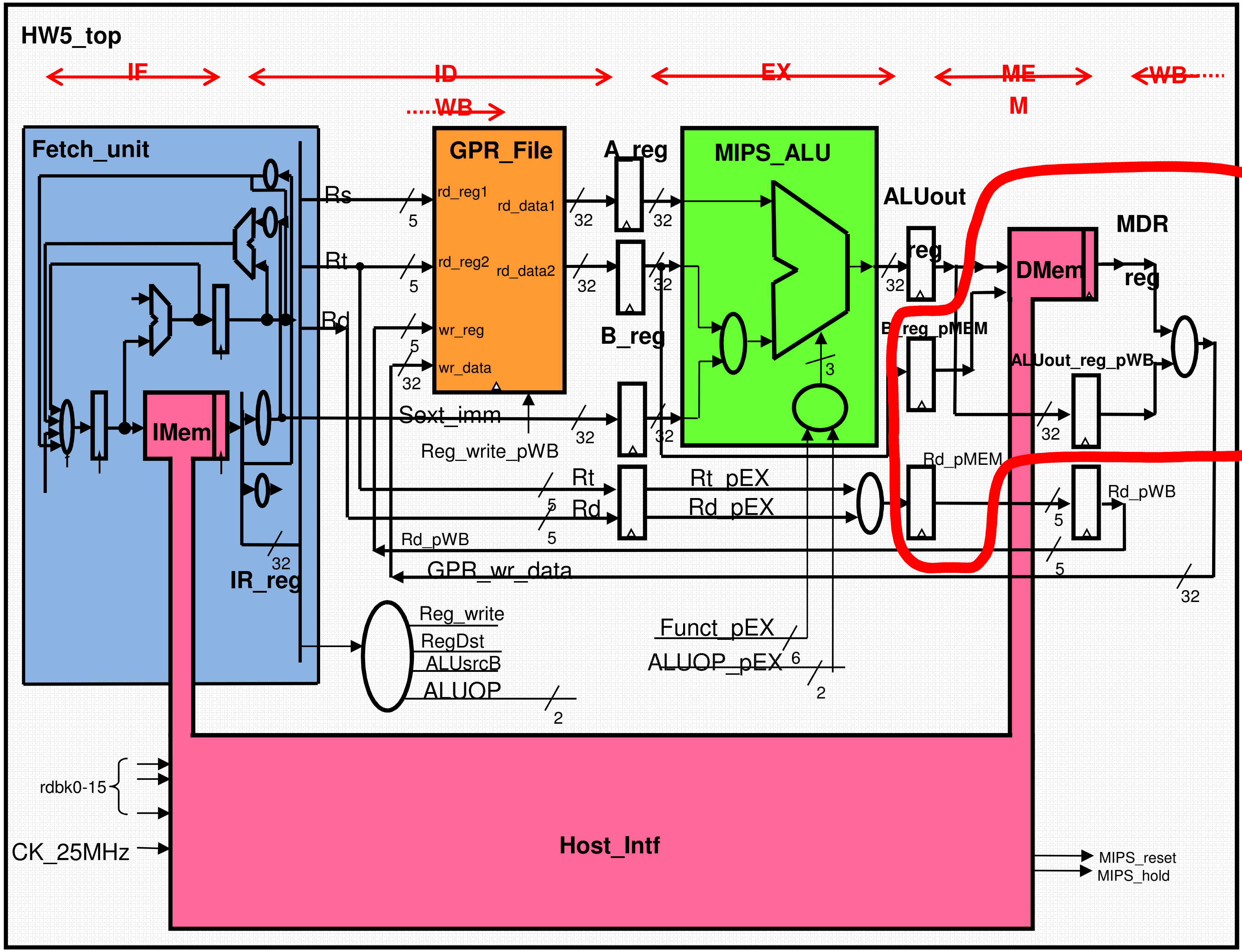
HW4_top (Rtype MIPS)



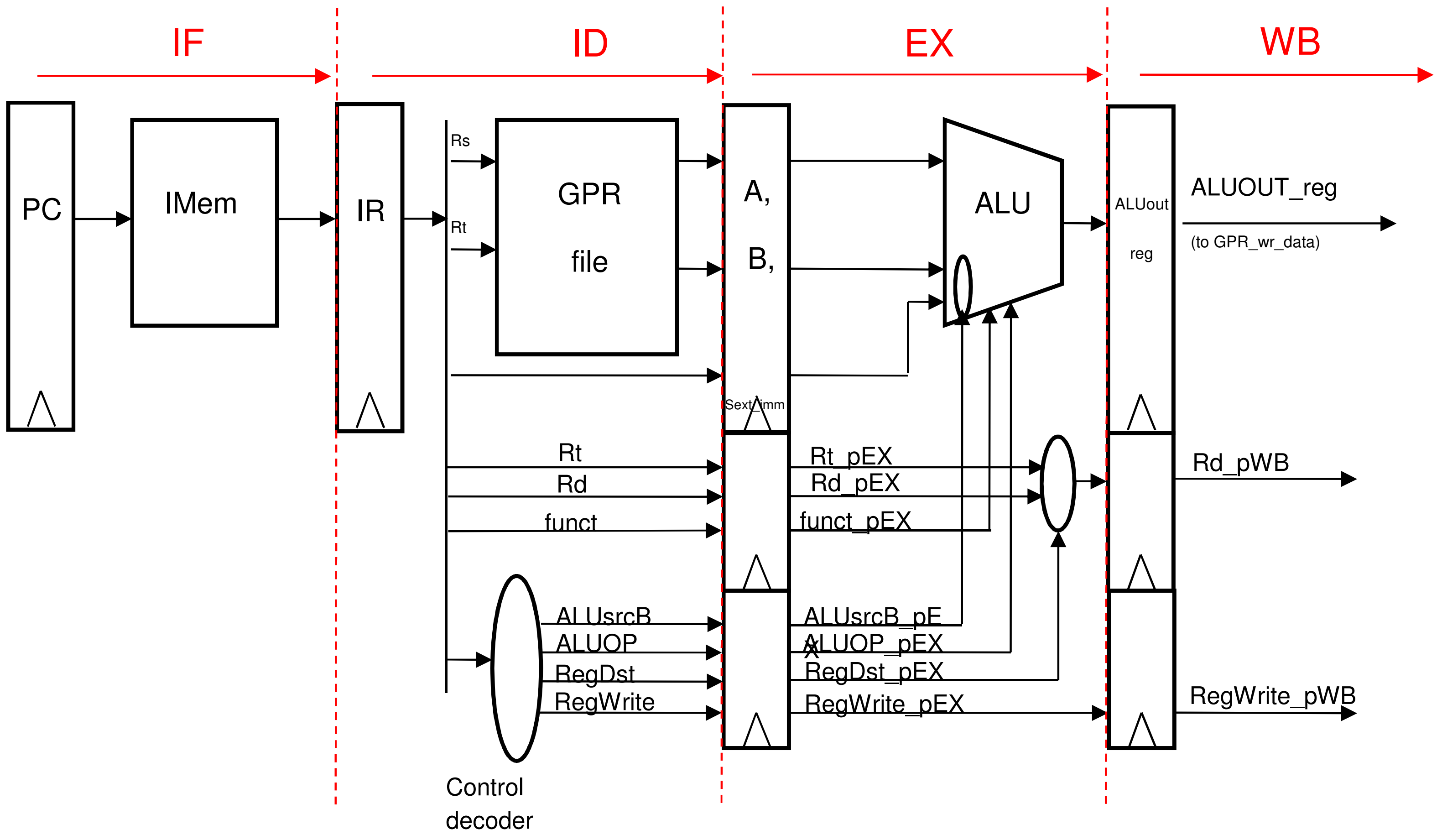
HW5_top (HW4 MIPS + lw & sw)



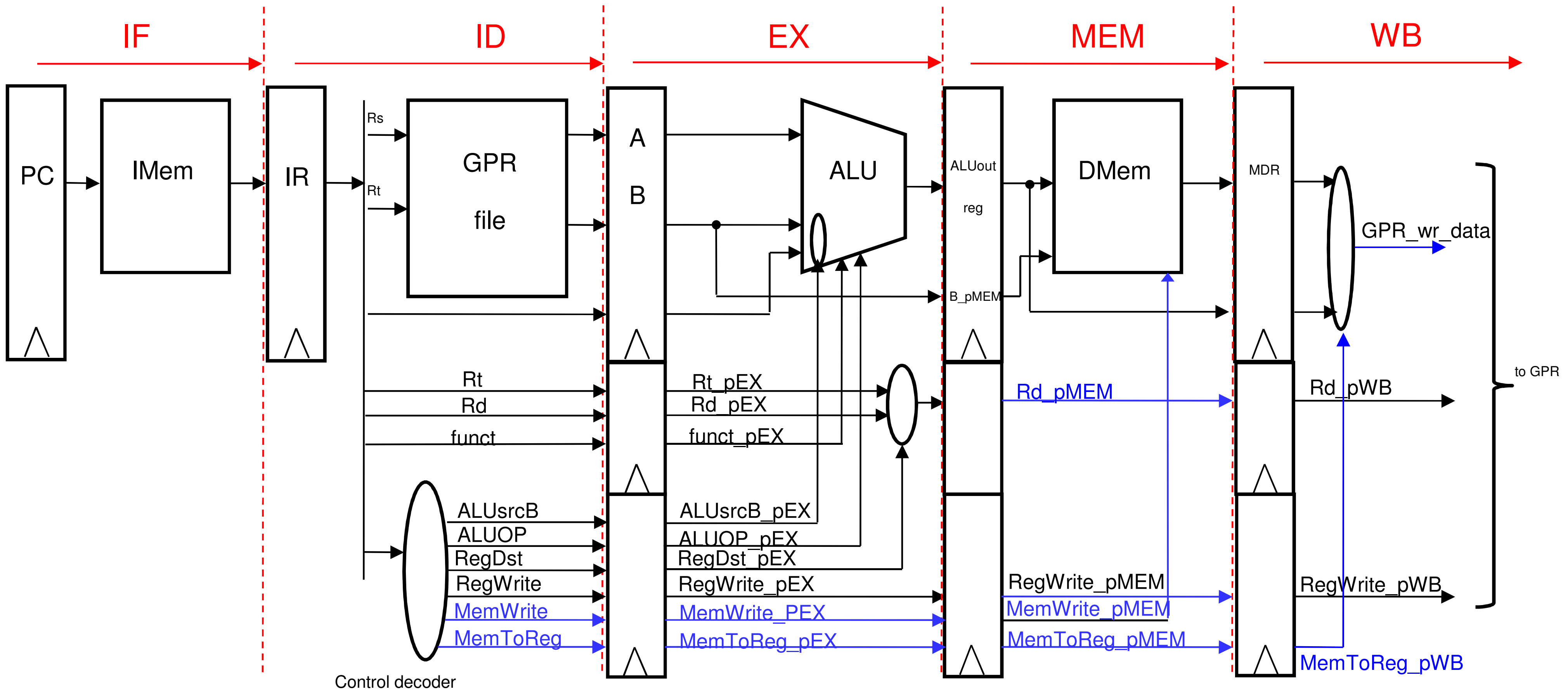
HW5_top (HW4 MIPS + lw & sw)



HW4_MIPS control scheme



HW5_MIPS control scheme



New control signals appear in blue

The HW5_top CPU has five 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.

MEM – Memory, which is the phase in which we read from the Dmem (in *lw*) or write into it (in *sw*). The read result is sampled into the MDR reg (*lw*) or into the Dmem (*sw*) at the rising edge of the clock which ends the MEM phase and starts the WB phase.

WB – Write Back, which is the final phase of the instruction. If this is an *Rtype* or *addi* instruction we write the ALUout_reg_pWB. In *lw* we write the MDR_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 execution of the instruction.

General signals in HW5_top (new signals shown in **RED**)

CK - The 25 MHz clock coming out of the Clock_driver.

RESET – coming out of the Host_Intf and is used as reset signal to all registers

HOLD –coming out of the Host_Intf and is used to freeze writing into all FFs & registers

ID phase signals in HW5_top

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

Opcode – the 6 MSBs of IR_reg. To be decoded and produce the control signals.

Rs – IR[25:21].

Rt – IR[20:16] .

Rd – IR[15:11].

Funct – IR[5:0].

sext_imm – renaming of sext_imm_pID coming out of the Fetch Unit.

GPR_rd_data1 – the 32 bit output of the rd_data1 of the GPR and input to A_reg.

GPR_rd_data2 – the 32 bit output of the rd_data2 of the GPR and input to B_reg.

Rs_equals_Rt – ‘1’ if GPR_rd_data1== GPR_rd_data2, and ‘0’ otherwise.

Used in branch instructions. That signal (renamed) is sent to the Fetch Unit.

ID control signals in HW5_top - These are created from decoding the opcode:

ALUsrcB – ‘1’ when sext_imm is used (in addi insruction).

ALUOP – a 2 bit signal. “00” maens add(addi inst.), “01” means subtract (not used),
“10” will cause the ALU to follow the Funct field.

RegDst – ‘0’ when we WB according to Rt (addi inst.) ‘1’ -according to Rd (Rtype inst.).

RegWrite – ‘1’ when we WB (Rtype or addi inst.), ‘0’ when we don’t (j, beq & bne inst.)

MemWrite – ‘1’ in sw (writing to Dmem), **MemToReg** = ‘1’ in lw (reading from DMem)

EX phase signals in HW5_top

A_reg – a 32 bit register receiving the GPR_rd_data1 signal. Its value is used in EX phase
B_reg – a 32 bit register receiving the GPR_rd_data2 signal
sext_imm_reg – a 32 bit register receiving the sext_imm coming from the Fetch Unit
Rt_pEX – Rt delayed by 1 clock cycle
Rd_pEX – Rd delayed by 1 clock cycle
ALUoutput – a 32 bit signal of the output of the ALU (renaming of ALU_out signal coming out of the MIPS_ALU component).

EX phase control signals in HW5_top

ALUsrcB_pEX – ALUsrcB delayed by 1 clock cycle.
Funct_pEX – Funct delayed by 1 clock cycle.
ALUOP_pEX – ALUOP delayed by 1 clock cycle.
RegDst_pEX – RegDst delayed by 1 clock cycle.
RegWrite_pEX – RegWrite delayed by 1 clock cycle.
MemWrite_pEX – MemWrite delayed by 1 clock cycle.
MemToReg_pEX – MemToReg delayed by 1 clock cycle.

MEM phase signals in HW5_top

B_reg_pMEM – a 32 bit register receiving the B_reg signal (i.e., B_reg delayed by 1 CK). This register has the data to be written into the DMem in sw instruction.

Rd_pMEM – the output of RegDest mux selecting to which register the CPU writes in the WB phase.

MEM phase control signals in HW5_top

MemWrite_pMEM - MemWrite_pEX delayed by 1 clock cycle.

MemToReg_pMEM – MemToReg_pEX delayed by 1 clock cycle.

RegWrite_pMEM – RegWrite_pEX delayed by 1 clock cycle.

WB phase signals in HW5_top

MDR_reg- a 32 bit register that has the data read from the memory. Rename of DMem_rd_data signal coming out of the **HW5_Host_Intf_4sim** component.

ALUout_reg_pWB - a 32 bit register that has the ALUour_reg data delayed by 1 CK cycle

GPR_wr_data - a 32 bit signal that is the output of the MemToReg mux (selecting between MDR_reg and ALUout_reg_pWB).

Rd_pWB – Rd_pMEM delayed by 1 clock cycle.

WB phase control signals in HW5_top

MemToReg_pWB – MemToReg_pMEM delayed by 1 clock cycle

RegWrite_pWB – RegWrite_pMEM delayed by 1 clock cycle.

You get a **HW5_top_4sim.empty** file in which you have all of these signals defined . You have to add your design of the HW5_MIPS, i.e., write the equations of the top file. In this vhd file we use the **Fetch_Unit**, **GPR**, **MIPS_ALU**, **Clock_Driver** and the **BYOC_Host_Intf_4sim**.

Description of the HW5_top_4sim project

1. **HW5_top_4sim.vhd** – This is your design of HW5. It uses the **GPR**, **MIPS_ALU** the updated **Fetch_Unit**, the **BYOC_Clock_driver_4sim**and the **BYOC_Host_Intf_4sim** components and all of the signals described in 2b.
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 HW4.
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.
8. **SIM_HW5_TB.vhd** - The TB vhd file prepared in advance. See the note in 9 below.
9. **SIM_HW5_TB_data.dat** – this is a data file prepared in advance that is read by the SIM_HW5_TB and used to compare the simulation results to the expected ones.
10. **SIM_HW5_program.dat** - The program file for simulation.
11. **SIM_HW5_filenames.vhd** - The actual path information of the two dat files.

NOTE: In **SIM_HW5_filenames.vhd** we specified the path of the **dat** file. You should update that according to your simulation project actual path.

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 HW5_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 See instructions in BYOC_HW5.doc. ***This should be a WORD file and not a PDF file – so remarks can be added when grading the report*****.**

Later, in the Implementation phase you will add 3 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)
- **ISE** – here you should have the HW5_top project created by the Xilinx ISE SW.
- DOCs – Same as in simulation. See instructions in BYOC_HW5.doc

Simulation report (cont)

The first part of the program we have in the IMem in HW5 simulation (addresses 400000h to 4001ACh) is very similar to the one you had in HW4 and is meant to check that you did not mess anything during the changes you did. That part is tested by the TB. The **SIM_HW5_TB.dat** file **contains test data only for this part of the IMem program**.

The rest of the IMem program, from 4001B0h till the end (400330h), is meant to test the writing and reading from the DMem. The program is given in Appendix A at the end of the HW5 document. It can also be seen in the last part of the **SIM_HW5_program.dat** file.

In order to test your design **you need to look at the simulated waveforms and decide whether it is OK or not**. You should run the simulation for 10.5us (10500ns). In the doc file you need to attach screen captures describing this part of the simulation you made, as detailed in 3.1.

Simulation report (cont)

3.1) The listed below signals should be presented in the screen capture you need to attach to your report. Show clock cycles **196-224** (following the end of the reset pulse, find i=**196-224**) and make the values of all signals readable. For this you will probably need to show clocks **196-210** and **210-224** separately. These are the signals that can help you in "testing" the DMem.

Note the some of these signals are inside the Host Intf :
CK
RESET
HOLD
i (the serial no. of the clock cycle – created by the TB)
MIPS_DMem_adrs
MIPS_DMem_wr_data
MIPS_DMem_we
MIPS_DMem_rd_data
DMem_reg0
DMem_reg1
DMem_reg2
DMem_reg3
DMem_reg4
PC_reg
IR_reg

(i=**196-224** means CK cycles from **8440** ns to **9600** ns)

Simulation report (cont)

3.2) Explain in detail what happens, i.e., what do we see here. Note that it is essential to the success of your future design that you will verify that the design does what we wanted it to do in these CK cycles.

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

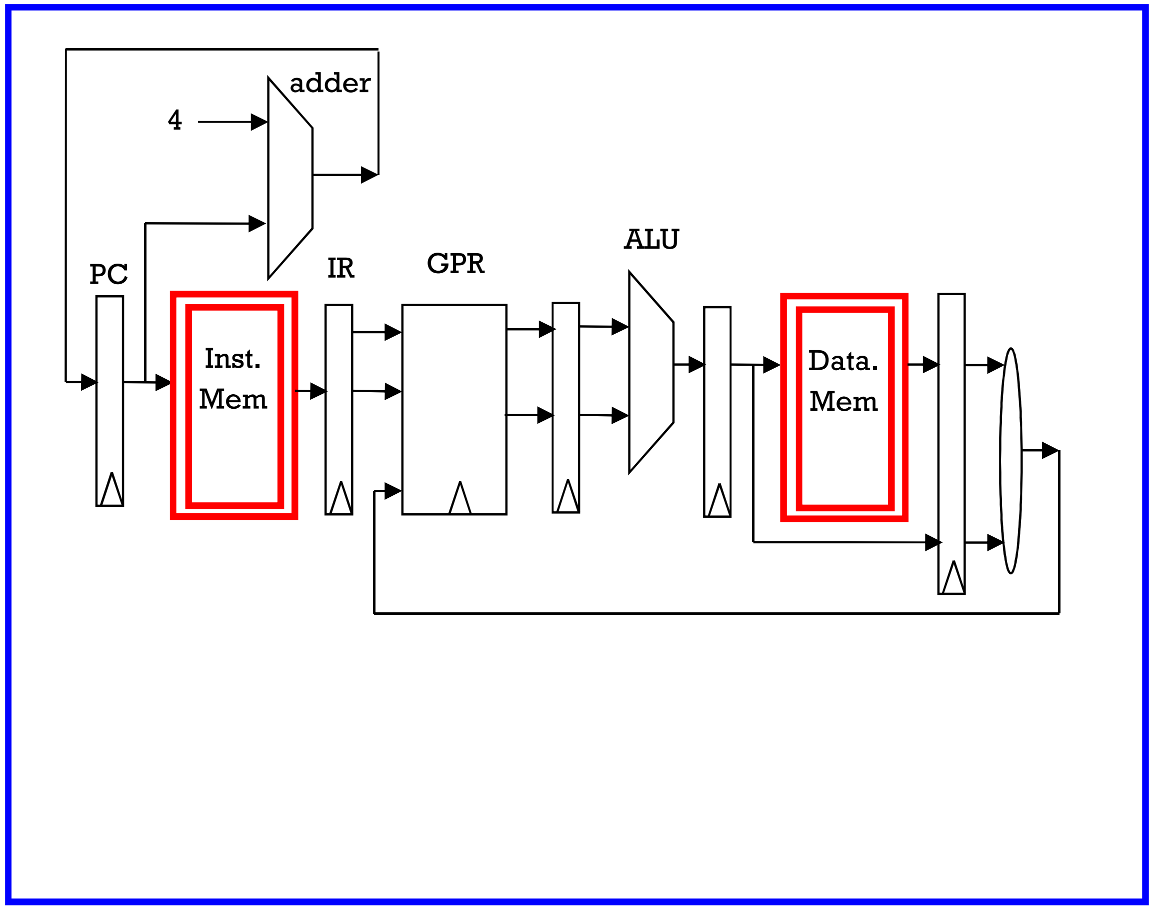
3.3) What is the latency of Rtype instruction? How many nop-s should be inserted between two consecutive Rtype instructions if the 2nd one uses the result of the 1st one?

3.4) Explain the limitation of beq that tests a register that is calculated by Rtype instruction. As an example, translate the following C if statement: for (i=0;i<10;i++) { … }
where i is register $3.

3.5) Are there any other limitations due to the pipeline structure in the instructions we implemented (Rtype, addi, beq, bne, j, lw, sw)? How can we overcome these limitations (e.g., by adding nop-s)?

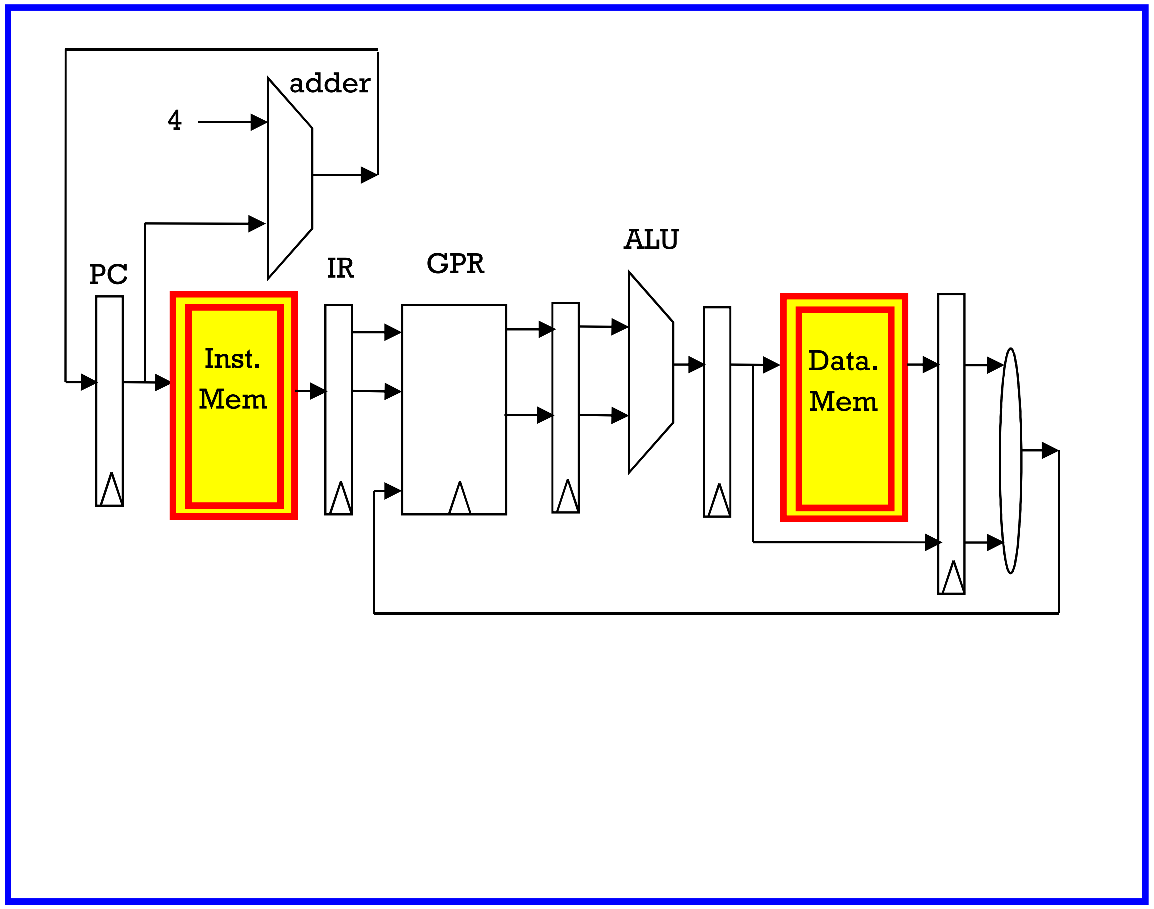
Now let’s talk on the Implementation phase

Your pipelined MIPS



All rights belong to Daniel Seidner

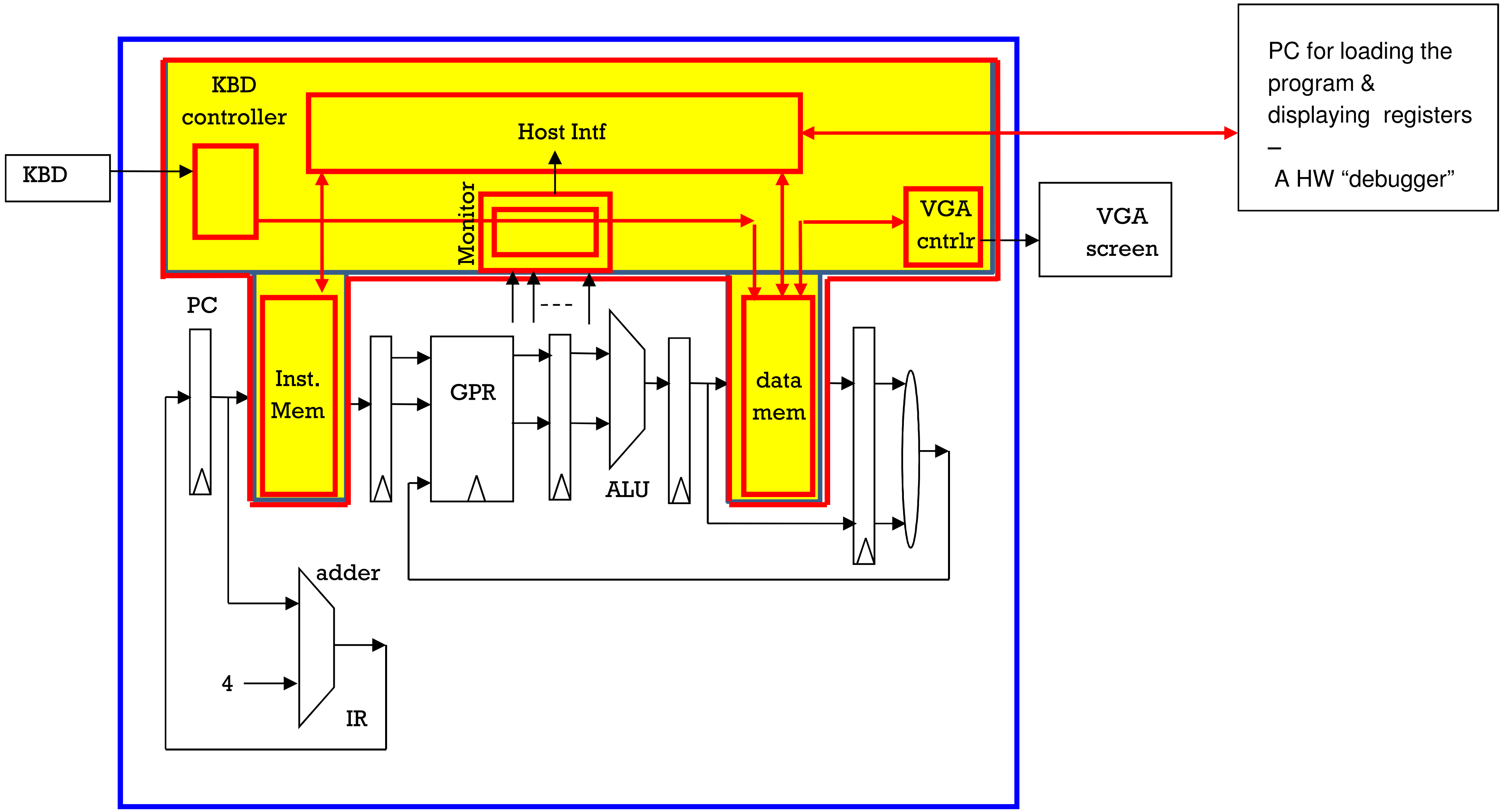
Your pipelined MIPS



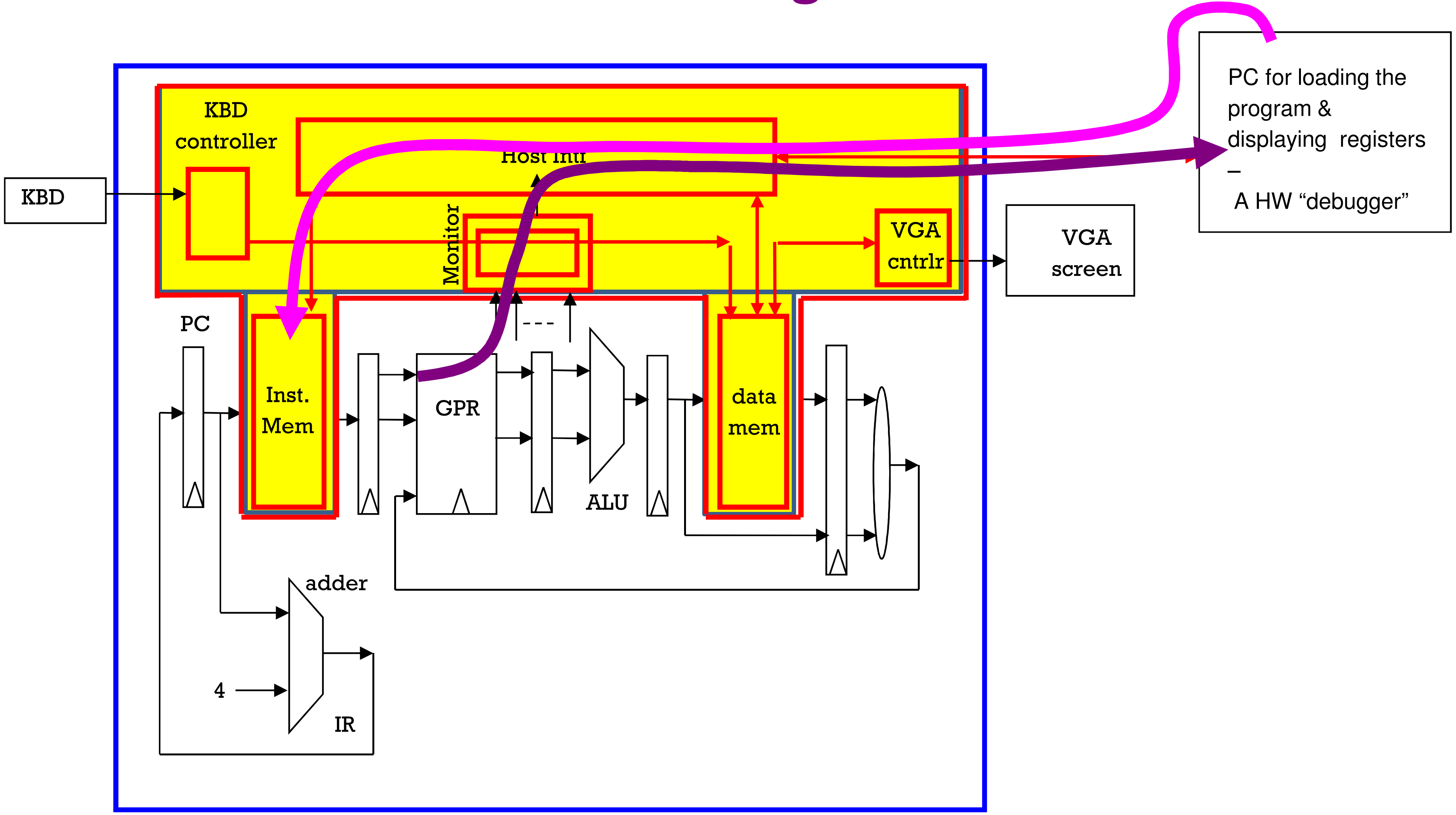
Then memories (in yellow), are part of the BYOC_Host_Intf

All rights belong to Daniel Seidner

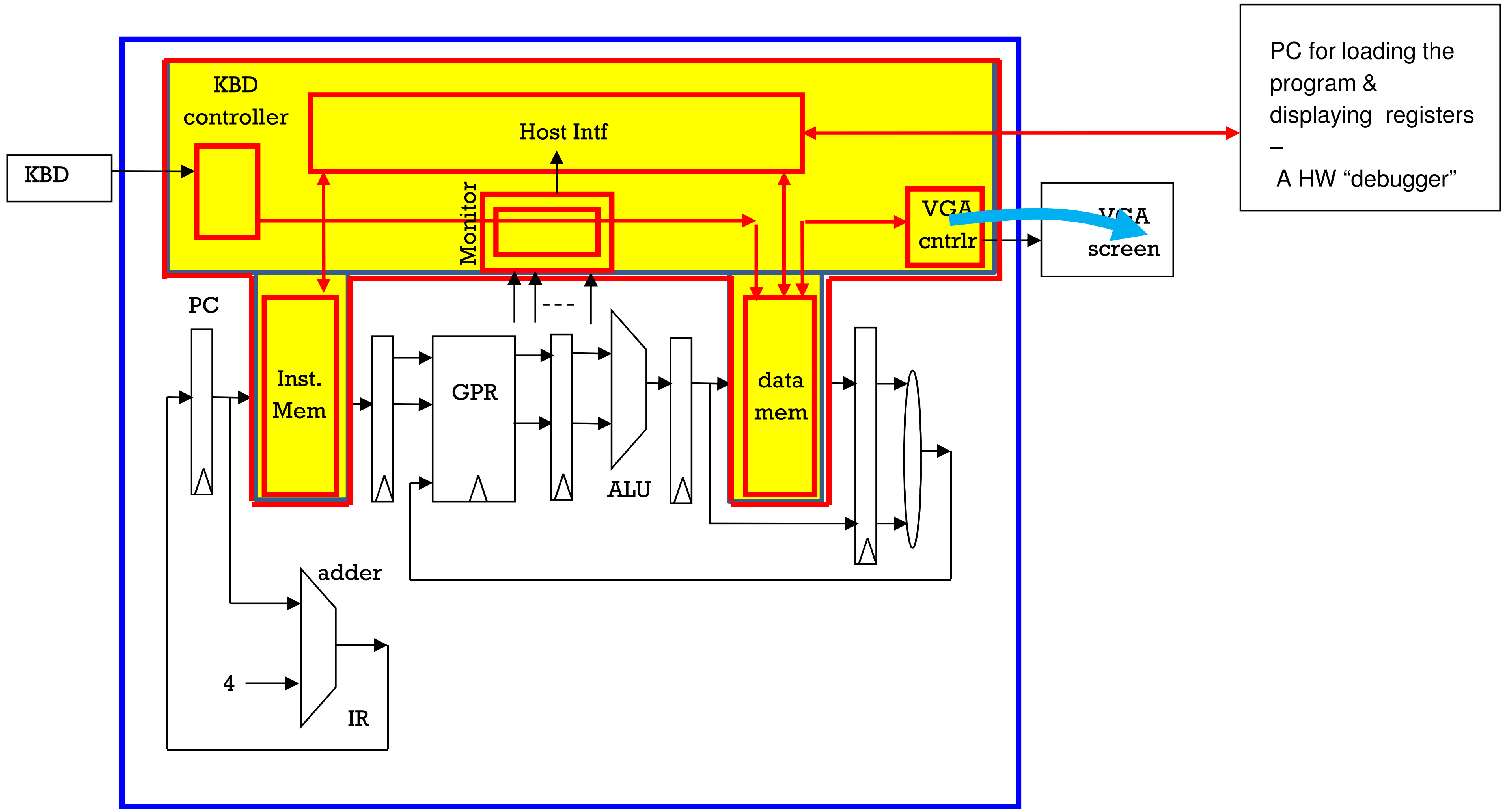
Your MIPS with the BYOC_Host_Intf



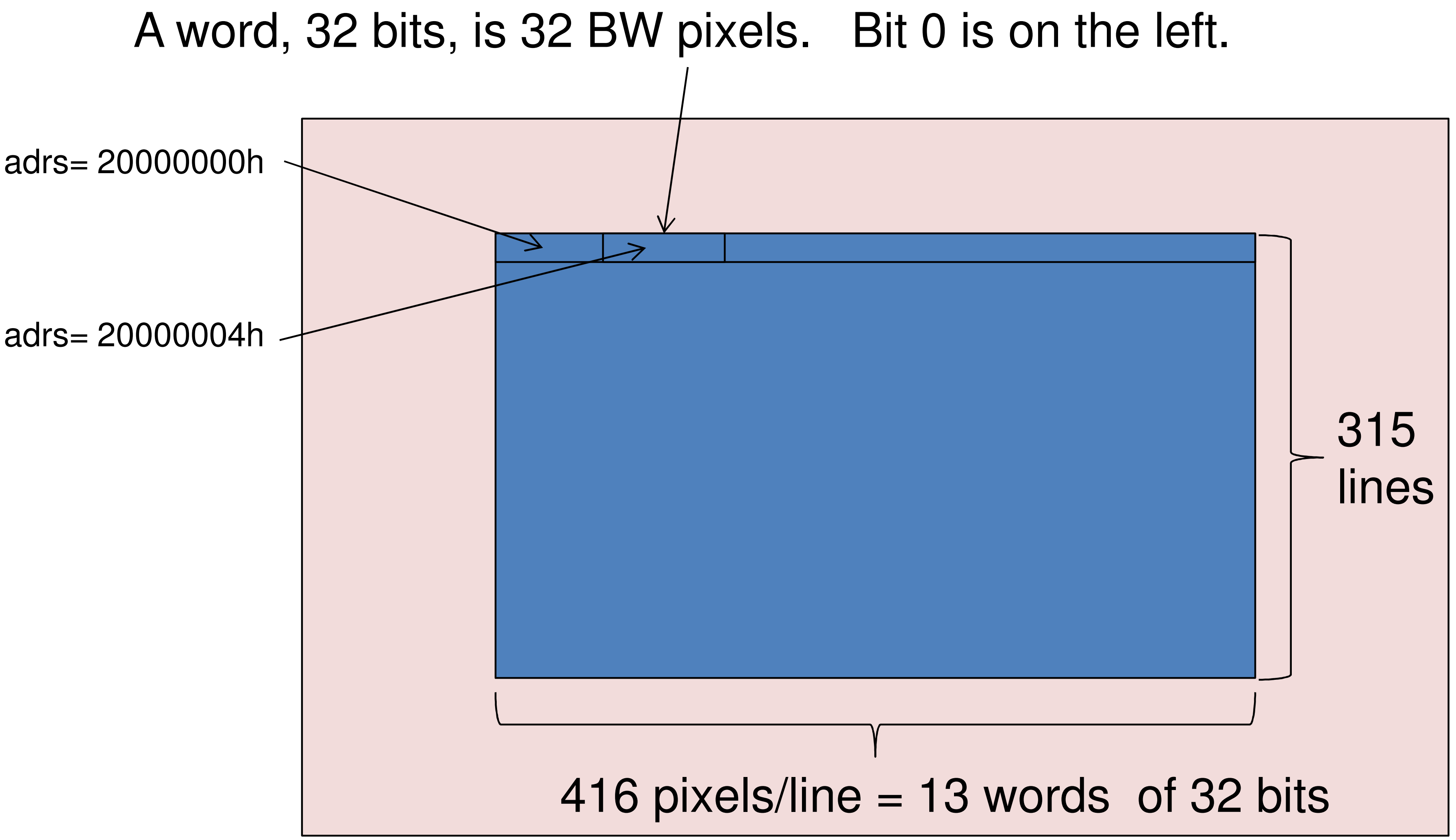
- We load the IMem via the BYOCIntf SW
- Then use it also to apply a single ck and check the rdbk signals



- We can "see" the DMem on a VGA screen
- What does this mean?



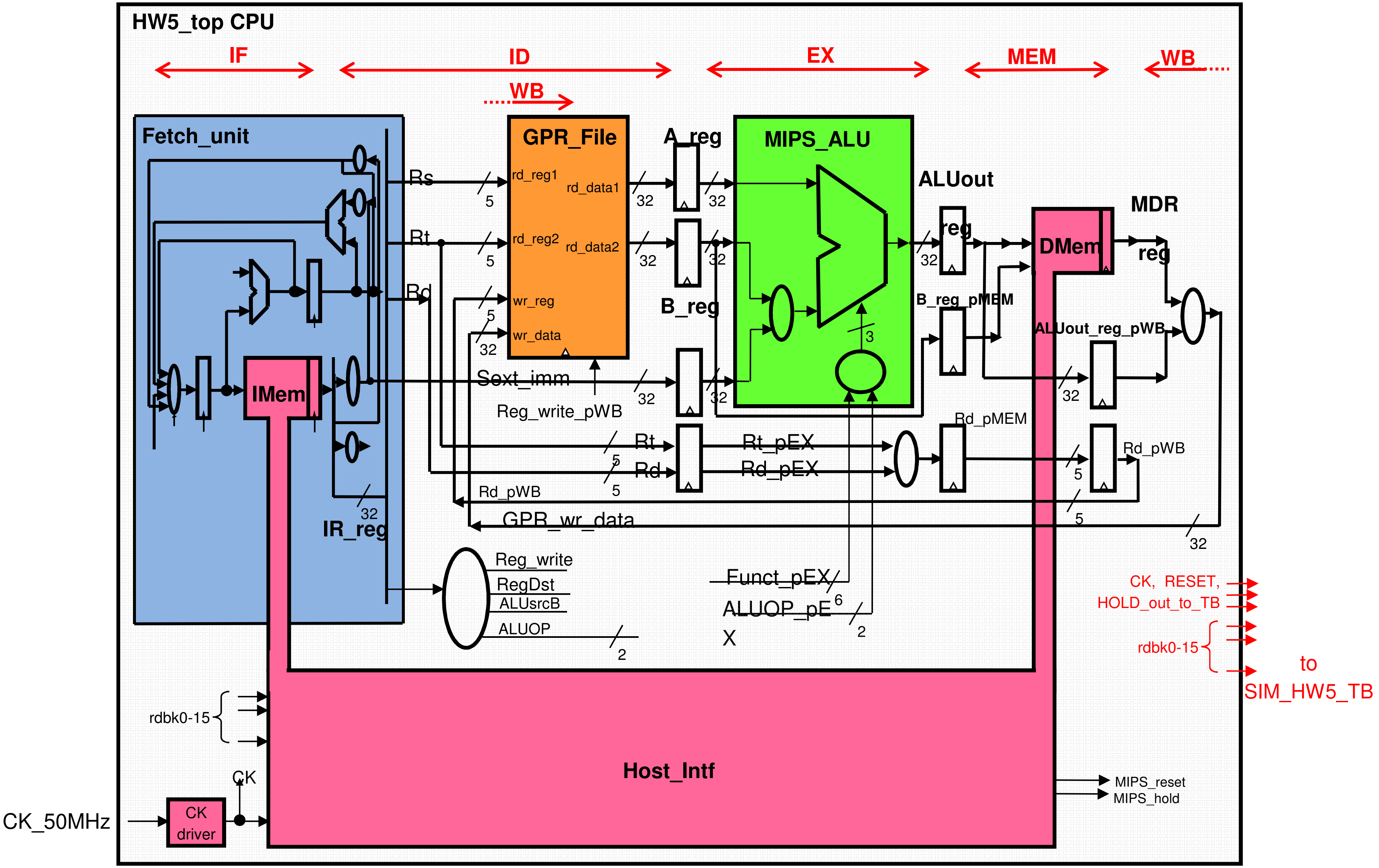
The VGA screen


A word, 32 bits, is 32 BW pixels. Bit 0 is on the left.
adrs= 20000000h
adrs= 20000004h
315 lines
416 pixels/line = 13 words of 32 bits

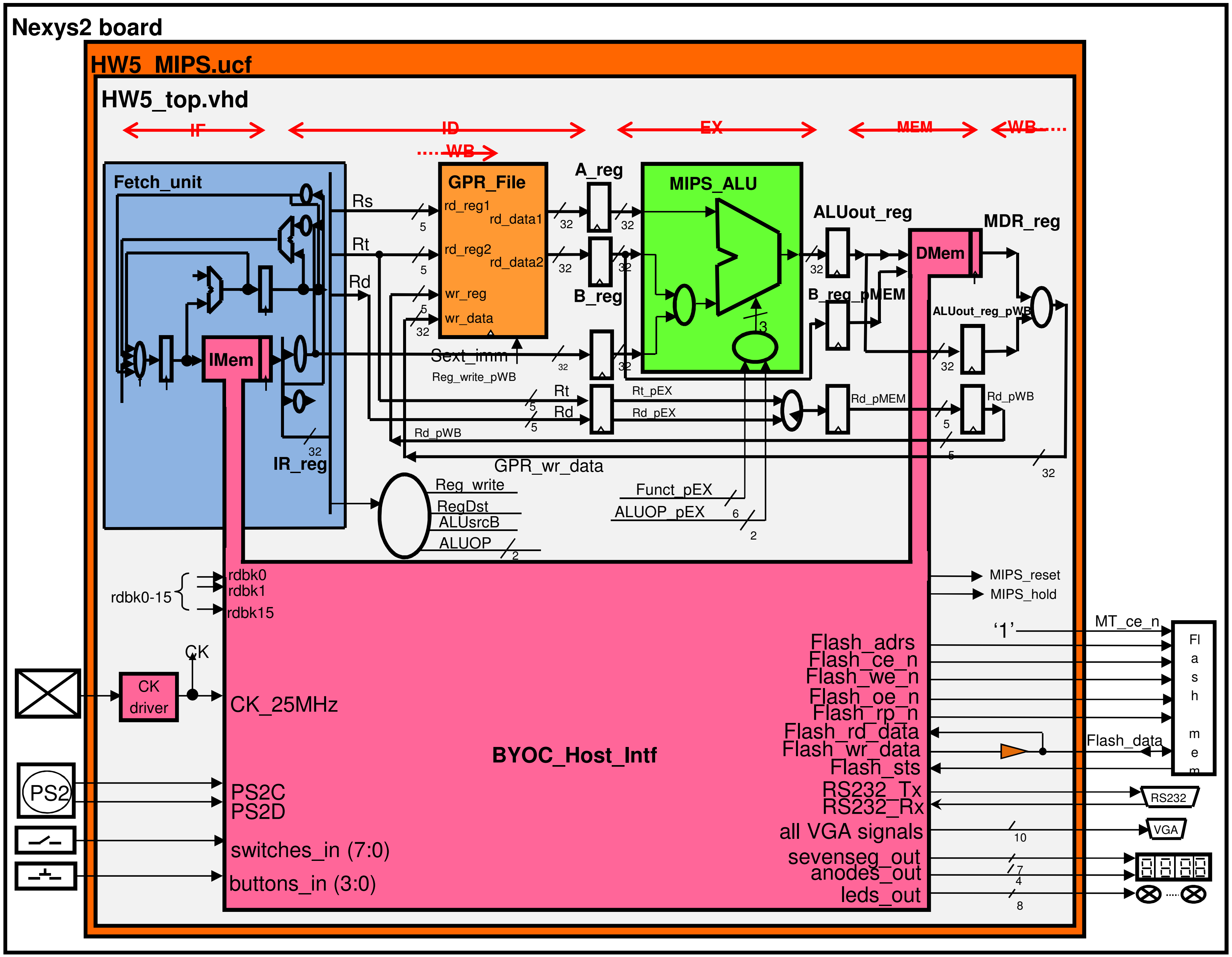
- **13x315=4095 i.e., we have 4095 words for the "screen memory"**
- **Addresses 20000000h-20003FFCh +52 (34h) means going down**

All rights belong to Daniel Seidner

The simulation version



The implementation version



```vhdl
entity HW5_top is
port  (
-- Host intf signals - Infrastructure signals [
RS232_Rx        : in STD_LOGIC;
RS232_Tx        : out STD_LOGIC;
-- VGA signals
VGA_h_sync      :  out          STD_LOGIC;
VGA_v_sync      :  out          STD_LOGIC;
VGA_red0        :  out          STD_LOGIC;
VGA_red1        :  out          STD_LOGIC;
VGA_red2        :  out          STD_LOGIC;
VGA_grn0        :  out          STD_LOGIC;
VGA_grn1        :  out          STD_LOGIC;
VGA_grn2        :  out          STD_LOGIC;
VGA_blu1        :  out          STD_LOGIC;
VGA_blu2        :  out          STD_LOGIC;
--Flash Mem signals
MT_ce_n         :  out    STD_LOGIC;-- '0' when accessing Nexys2 SDRAM –not used
Flash_adrs      :  out              STD_LOGIC_VECTOR(23 downto 1);--Flash read/write address
Flash_ce_n      :  out          STD_LOGIC;-- '0' when accessing Flash mem
Flash_we_n      :  out          STD_LOGIC;-- '0' when writing to Flash mem
Flash_oe_n      :  out          STD_LOGIC;-- '0' when reding from Flash mem
Flash_rp_n      :  out          STD_LOGIC;-- '0' when reseting Flash mem
Flash_sts       :  in           STD_LOGIC;-- '1' when Flash mem FSM is done
Flash_data      :  inout  STD_LOGIC_VECTOR(15 downto 0);--Data from/to Flash to/from IMem/DMem
--KBD signals
PS2C                       :  in           STD_LOGIC;-- PS2 keyboard clock
PS2D                       :  in           STD_LOGIC;-- PS2 keyboard data
--general signals
leds_out  : out STD_LOGIC_VECTOR(7 downto 0);-- 7=Flash_stts,6=MIPS_ck,5-0=Host_Intf vesion
CK_50MHz        : in            STD_LOGIC;
buttons_in      : in            STD_LOGIC_vector(3 downto 0);--btn0=single clock ,btn3=manual reset
switches_in     : in            STD_LOGIC_VECTOR(7 downto 0);-- to be described later
sevenseg_out    : out           STD_LOGIC_VECTOR (6 downto 0);-- to the 7 seg LEDs
anodes_out      : out           STD_LOGIC_VECTOR (3 downto 0) -- to the 7 seg LEDs
---- signals to be tested by the TB  -- REMOVED FOR IMPLEMENTATION!!
--
    );
end HW5_top;
```

All the i/o pins in the HW5_top entity were not really used in simulation (except the CK_50MHz). In implementation we use all of the i/o shown in blue

HW5_top - implementation

1. Take your **HW5_top_4sim.vhd**, remove all TB signals and rename to **HW5_top.vhd**.
   You can look at the difference between the the **HW5_top_4sim.empty** and the **HW5_top.empty** files that were given to you.

2. You should replace the **BYOC_Host_Intf_4sim.vhd** with a component that looks the same, the **BYOC_Host_Intf.ngc**, which has the infra-structure that allows the PC to load data into the IMem via the RS232 by the **BYOCInterface** SW.

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

- **BYOC.ucf** - The file listing which signal are connected to which FPGA pins in the Nexys2 board.
- **HW5_top.vhd** – This is your design of HW5
- **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after modifications of HW4
- **GPR.vhd** – your GPR File design you prepared in HW3.
- **dual_port_memory.vhd** – part of the GPR File design you prepared in HW3.
- **MIPS_ALU.vhd** – your MIPS_ALU design you prepared in HW3.
- **BYOC_Clock_driver.vhd** – the CK divider & driver we also used in HW2 & HW4.
- **BYOC_Host_Intf.ngc** - The actual infrastructure interfacing the PC.

4. Now run the Xilinx ISE SW, create a HW5_top.bit file and test it.

HW5_top – testing the implemented design

We’ll run that the **BYOCInterface** SW and 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 “**HW5_rect4.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. The **HW5_rect4** program is given in **Appendix B** at the end of the BYOC_HW5.doc document.

This time we would like to connect a VGA screen to the Nexys2 board.

- What is the value of register $2 after 122 cks?
- What happens after 126 CKs?

  (To answer thee two questions look what happens from 120 CKs till 130 CKs)
- What happens when you press the RUN button?

HW5_top – implementation 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 3 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 HW5 project 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 the instructions See instructions in BYOC_HW5.doc.
  ***This should be a WORD file and not a PDF file – so remarks can be added when grading the report*.**

In this part of the implementation report you should answer the following questions:

1. What is the value of register $2 after 122 cks?
2. What happens after 126 CKs?
   (To answer thee two questions look with the BYOCSIntf SW what happens from 120 CKs till 130 CKs)
3. What happens when you press the RUN button?
4. Explain the **HW5_rect4** program (what is the job of every register used. What is done in each loop, etc.)
5. How long does it take [in seconds] to draw a 32x32 white square when we use the draw loop of the **HW5_rect4** program?
6. Can you shorten the loop? If you can, write the code and explain.
7. Can you think of a faster way to draw the square in the same short loop? If you can, write the code and explain.

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!

Thanks for listening!