HW6 – The final MIPS

Part I – Adding new instructions
Part II – Adding Data Forwarding
Part III – Adding Branch Forwarding

HW6 – The final MIPS

Part I

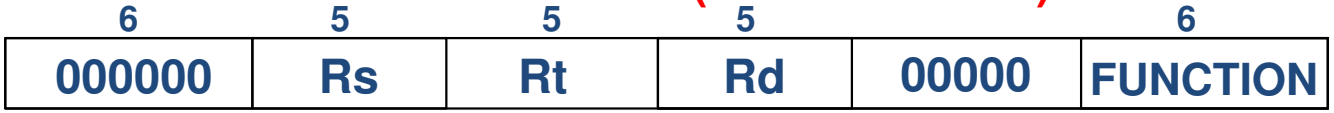
Adding lui, ori, jr, jal instructions to the CPU instruction set of Rtype, addi, beq, bne, j, lw, sw

HW6 instruction set

(new inst. in **red**)

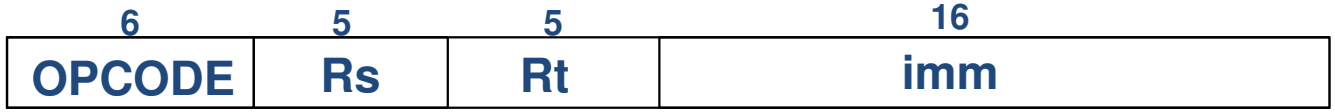
R-type

```
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
jr Rs                #  PC=Rs    (note that Rd=0)
```



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
ori  Rt, Rs, imm     #  Rt=Rs OR imm     (no sext)
lui   Rt, imm        #  Rt= imm<<16   (no sext)
```

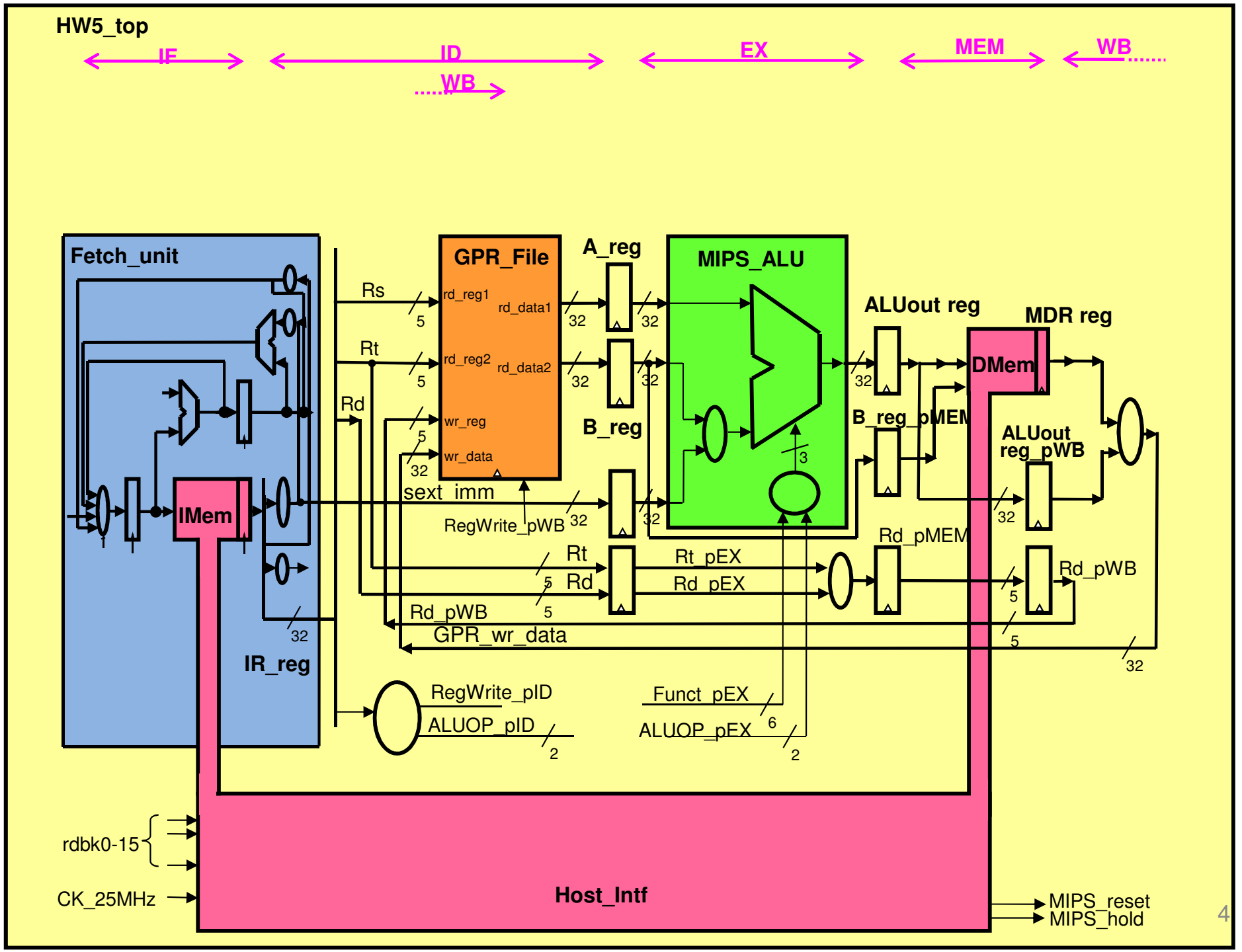


---

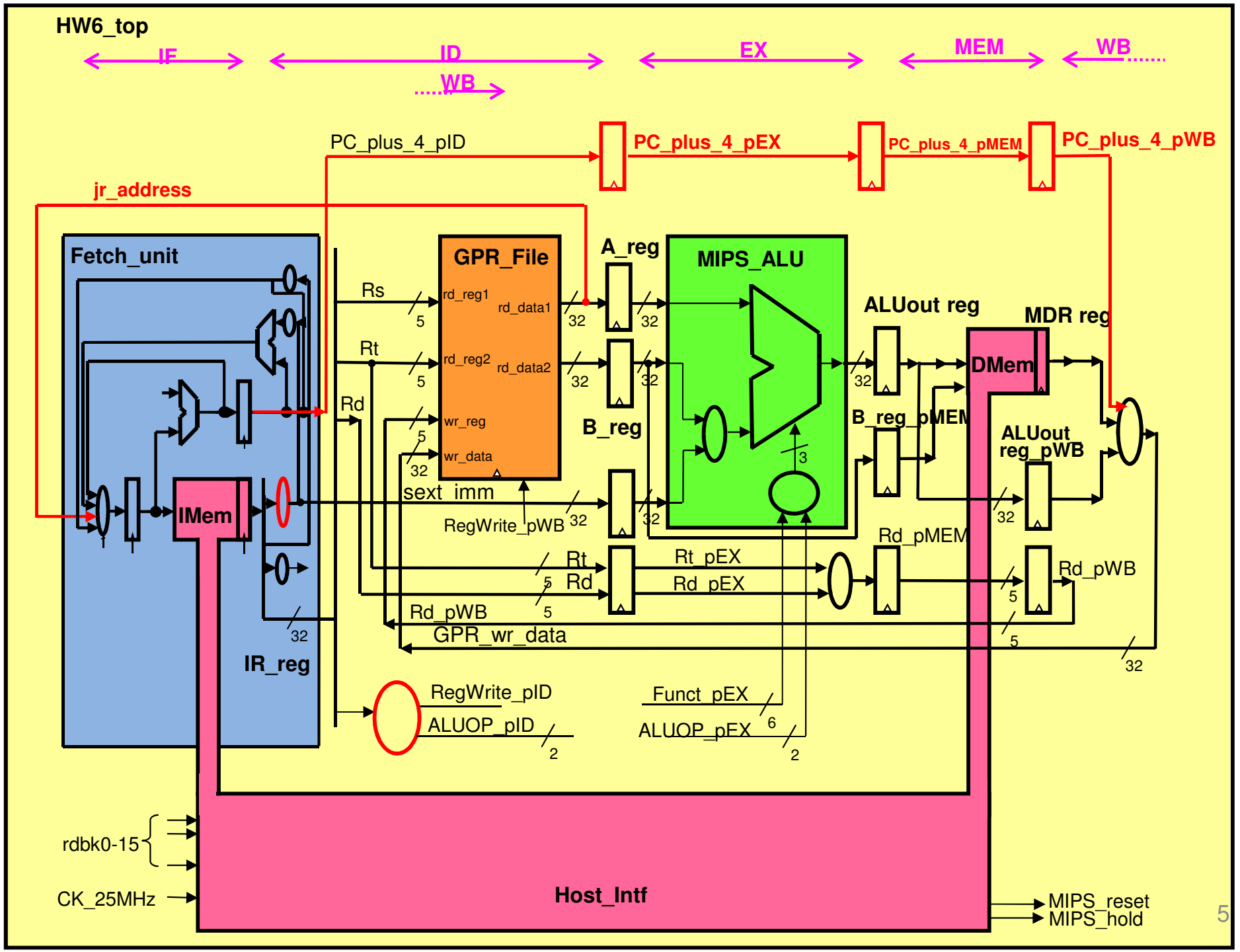
j-type

```
j     imm            #  PC= imm*4                    (no sext)
jal  imm             #  PC= imm*4,  $31=PC+4    (no sext)
```

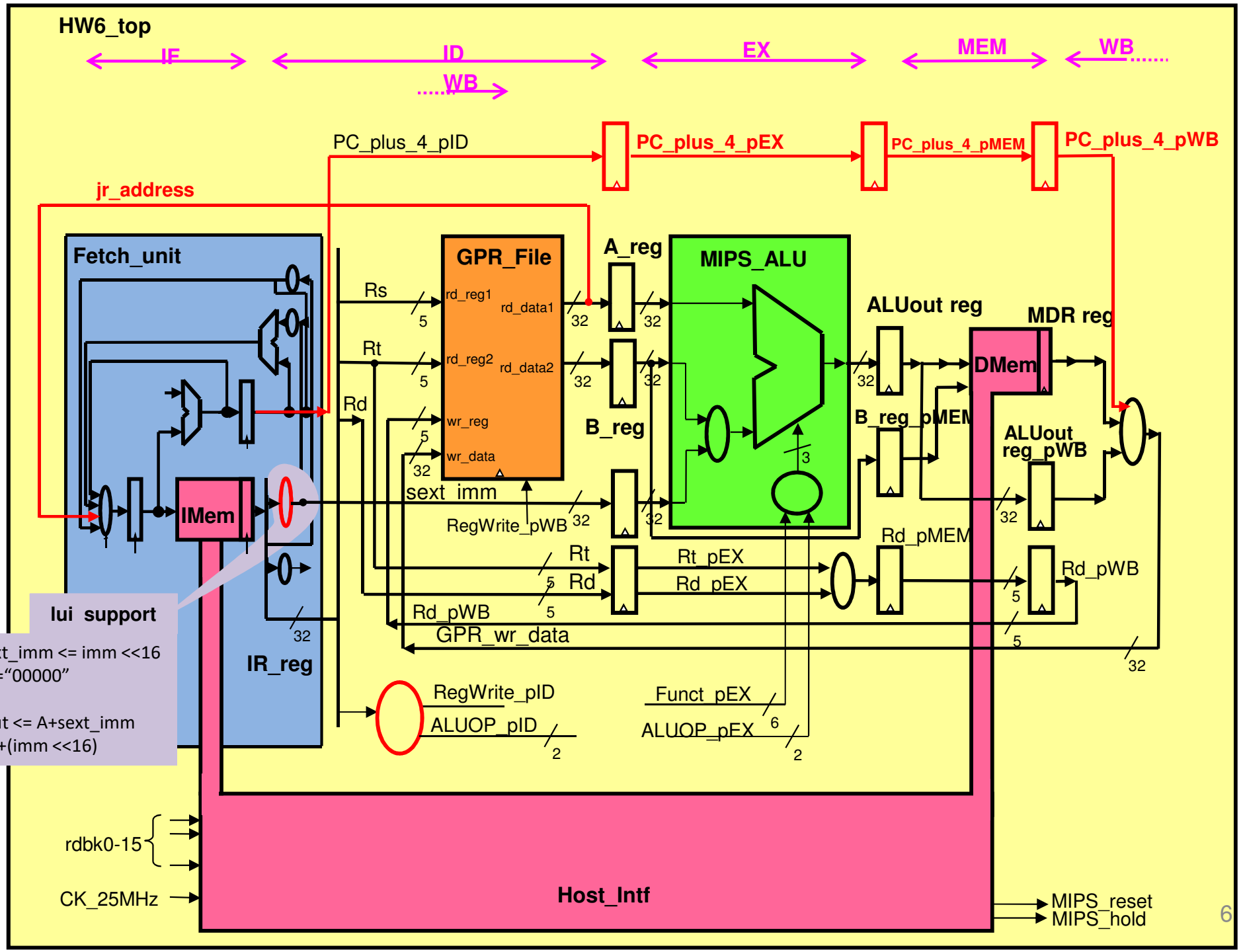
HW5_top


HW5_top
IF
ID
WB
EX
MEM
WB
Fetch_unit
IMem
IR_reg
GPR_File
rd_reg1
rd_reg2
wr_reg
wr_data
rd_data1
rd_data2
A_reg
B_reg
MIPS_ALU
ALUout reg
MDR reg
DMem
B_reg_pMEM
ALUout reg_pWB
Rs
Rt
Rd
5
32
sext_imm
RegWrite_pWB
Rt_pEX
Rd_pEX
Rd_pMEM
Rd_pWB
GPR_wr_data
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
6
2
3
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold

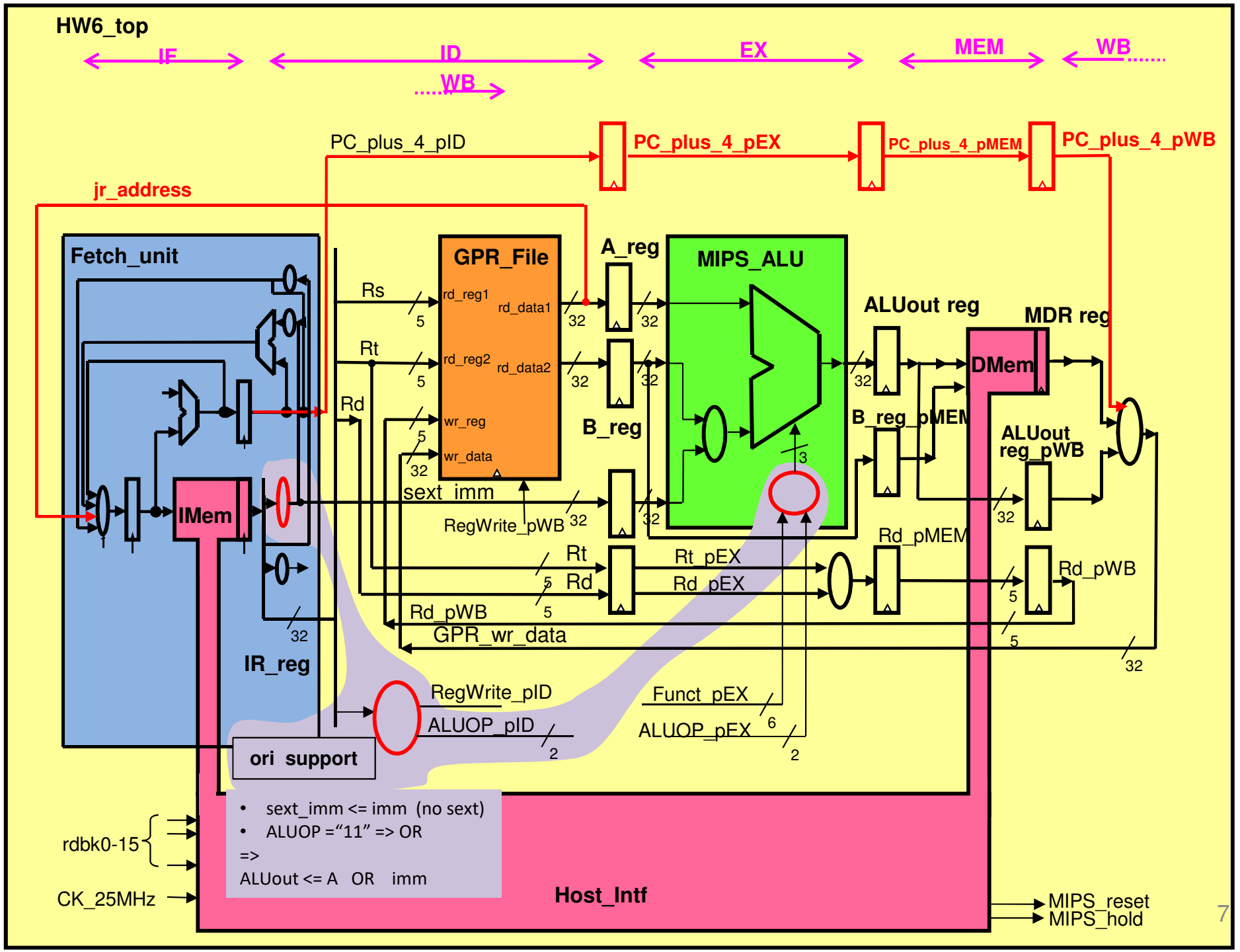
HW6_top


HW6_top
IF
ID
WB
EX
MEM
WB
PC_plus_4_pID
PC_plus_4_pEX
PC_plus_4_pMEM
PC_plus_4_pWB
jr_address
Fetch_unit
IMem
IR_reg
GPR_File
rd_reg1
rd_reg2
wr_reg
wr_data
rd_data1
rd_data2
Rs
Rt
Rd
5
32
A_reg
B_reg
MIPS_ALU
3
ALUout reg
MDR reg
DMem
B_reg_pMEM
ALUout reg_pWB
sext_imm
RegWrite_pWB
Rd_pMEM
Rt_pEX
Rd_pEX
Rd_pWB
GPR_wr_data
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
6
2
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold

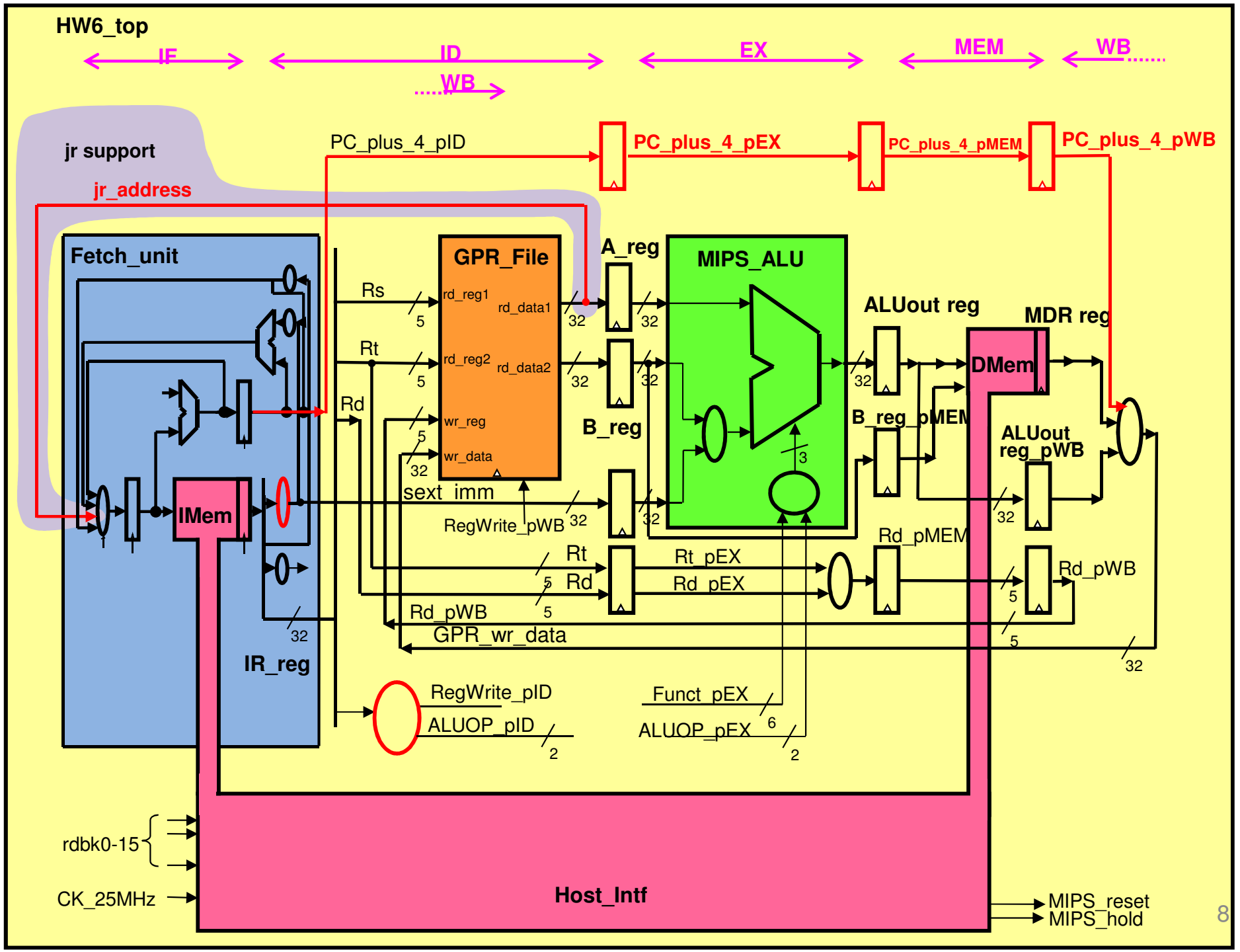
HW6_top


HW6_top
IF
ID
WB
EX
MEM
WB
PC_plus_4_pID
PC_plus_4_pEX
PC_plus_4_pMEM
PC_plus_4_pWB
jr_address
Fetch_unit
IMem
IR_reg
GPR_File
rd_reg1
rd_reg2
wr_reg
wr_data
rd_data1
rd_data2
Rs
Rt
Rd
5
32
A_reg
B_reg
MIPS_ALU
3
ALUout reg
MDR reg
DMem
B_reg_pMEM
ALUout reg_pWB
sext_imm
RegWrite_pWB
Rd_pMEM
Rt_pEX
Rd_pEX
Rd_pWB
GPR_wr_data
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
6
2
lui support
• sext_imm <= imm <<16
• Rs="00000"
=>
ALUout <= A+sext_imm
i.e., 0+(imm <<16)
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold

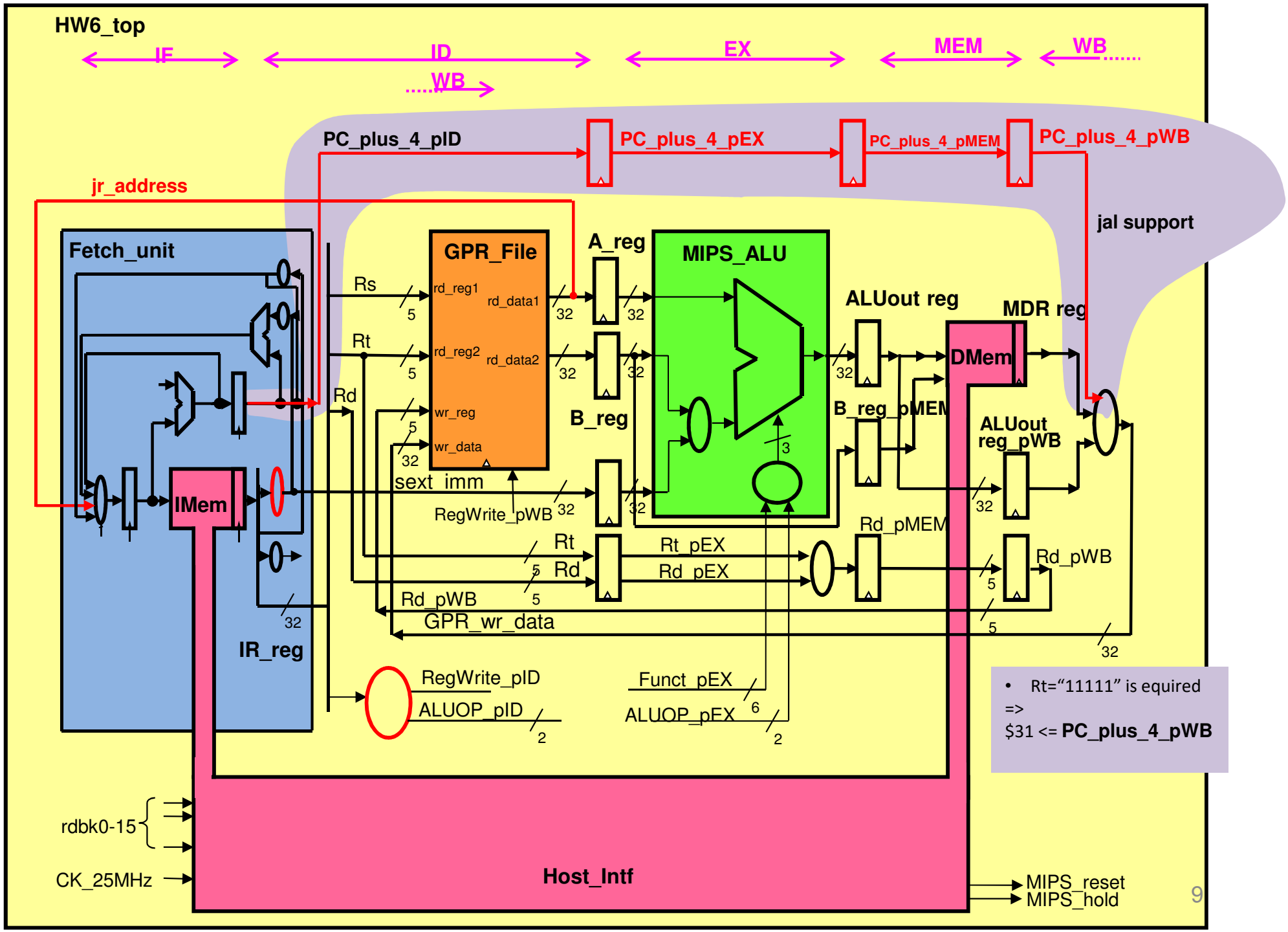
HW6_top


HW6_top
IF
ID
WB
EX
MEM
WB
PC_plus_4_pID
PC_plus_4_pEX
PC_plus_4_pMEM
PC_plus_4_pWB
jr_address
Fetch_unit
GPR_File
rd_reg1
rd_reg2
wr_reg
wr_data
rd_data1
rd_data2
A_reg
B_reg
MIPS_ALU
ALUout reg
MDR reg
DMem
B_reg_pMEM
ALUout reg_pWB
IMem
Rs
Rt
Rd
sext_imm
RegWrite_pWB
Rt_pEX
Rd_pEX
Rd_pMEM
Rd_pWB
GPR_wr_data
IR_reg
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
ori support
• sext_imm <= imm (no sext)
• ALUOP ="11" => OR
=>
ALUout <= A OR imm
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold

HW6_top


HW6_top
IF
ID
WB
EX
MEM
WB
jr support
jr_address
PC_plus_4_pID
PC_plus_4_pEX
PC_plus_4_pMEM
PC_plus_4_pWB
Fetch_unit
GPR_File
rd_reg1
rd_data1
rd_reg2
rd_data2
wr_reg
wr_data
A_reg
MIPS_ALU
ALUout reg
MDR reg
DMem
B_reg_pMEM
ALUout reg_pWB
B_reg
IMem
IR_reg
Rs
Rt
Rd
sext_imm
RegWrite_pWB
Rt_pEX
Rd_pEX
Rd_pMEM
Rd_pWB
GPR_wr_data
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold

HW6_top


HW6_top
IF
ID
WB
EX
MEM
WB
PC_plus_4_pID
PC_plus_4_pEX
PC_plus_4_pMEM
PC_plus_4_pWB
jr_address
jal support
Fetch_unit
GPR_File
A_reg
MIPS_ALU
ALUout reg
MDR reg
DMem
Rs
Rt
Rd
rd_reg1
rd_reg2
wr_reg
wr_data
rd_data1
rd_data2
B_reg
B_reg_pMEM
ALUout reg_pWB
IMem
sext_imm
RegWrite_pWB
Rd_pMEM
Rt_pEX
Rd_pEX
Rd_pWB
GPR_wr_data
IR_reg
RegWrite_pID
ALUOP_pID
Funct_pEX
ALUOP_pEX
rdbk0-15
CK_25MHz
Host_Intf
MIPS_reset
MIPS_hold
• Rt="11111" is equired
=>
$31 <= PC_plus_4_pWB

General signals in HW6_top

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

IF phase signals in HW6_top

PC_plus_4_pID – PC_plus_4_pID_out from the Fetch Unit – for jal support

ID phase signals in HW6_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.

jr_address – connected to the new **jr_adrs_in** pin in the Fetch Unit – for jr inst.

ID control signals in HW6_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)

JAL – '1' in jal . Used to control the expanded MemToReg mux

EX phase signals in HW6_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).

PC_plus_4_pEX – PC_plus_4_pID delayed by 1 clock cycle – for jal

EX phase control signals in HW6_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 – MemToRegdelayed by 1 clock cycle.

JAL_pEX – JAL delayed by 1 clock cycle.

MEM phase signals in HW6_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 – to which reg the CPU writes in the WB phase.

PC_plus_4_pMEM – PC_plus_4_pEX delayed by 1 clock cycle – for jal

MEM phase control signals in HW6_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.

JAL_pMEM – JAL_pEX delayed by 1 clock cycle.

WB phase signals in HW6_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
Rd_pWB – Rd_pMEM delayed by 1 clock cycle.
PC_plus_4_pWB – PC_plus_4_pMEM delayed by 1 clock cycle – for jal

WB phase control signals in HW6_top

MemToReg_pWB – MemToReg_pMEM delayed by 1 clock cycle
RegWrite_pWB – RegWrite_pMEM delayed by 1 clock cycle.
JAL_pWB – JAL_pMEM delayed by 1 clock cycle.

You get a **HW6_top_4sim.empty** file in which you have all of these signals defined . You have to add your design of the HW6_top entity, i.e., write the equations of the top file (and rename it to **HW6_top_4sim.vhd**).
In this vhd file we use the **Fetch_Unit**, **GPR**, **MIPS_ALU**, **BYOC_Clock_Driver** and the **BYOC_Host_Intf_4sim**.

Fetch_Unit & **MIPS_ALU** are in red to remind you they should be modified

Description of the HW6_top_4sim project

1. **HW6_top_4sim.vhd** – This is your design of HW6. It uses the **GPR**, the updated **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 **and updated for HW6!!**
5. **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after the modifications done in HW4 **and the additional modification in HW6!!**
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_HW6_TB.vhd** - The TB vhd file prepared in advance. See the note in 9 below.
9. **SIM_HW6_TB_no_fwdng.dat** – this is a data file prepared in advance that is read by the SIM_HW6_TB and used to compare the simulation results to the expected ones.
10. **SIM_HW6_program.dat** - The program file for simulation.
11. **SIM_HW6_filenames.vhd** - The actual path information of the two dat files.

NOTE: Inside **SIM_HW6_TB_filenames.vhd**, we specified the **path** of and the **name of the dat file** to which the design is compared. You should update that according to your sim project actual path and according to the design tested (no fwd, data fwd, branch fwd).

Simulation report

You should submit a single zip file for the Simulation and implementation phases. It should have **four** directories/folders. The first is called **Simulation1**, the 2nd is called **Simulation2,** the 3rd is called **Simulation 3**, the 4th is called **Implementation**. In the Simulation folders you will have 3 sub-folder of:

- **Src_4sim** – here you put all of the *.vhd sources for simulation
- **Sim** – here you should have the HW6 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 in BYOC_HW6.doc. ***This should be a WORD file and not a PDF file – so remarks can be added when grading the report*.**

Simulation1 will have the "no forwarding" design of **part I** where you add the lui, ori, jr and jal instructions. You should answer the questions in **Appendix A** and insert these to your report of **Simulation1**.
You should use the **SIM_HW6_TB_data_no_fwding.dat** file for the simulation.

Simulation report (cont.)

Simulation2 will have the "data forwarding" design of part II where you add the data forwarding to the design of **Simulation1**. You should answer the questions in **Appendix B** and insert these to your report of **Simulation2**.
You should use the **SIM_HW6_TB_data_wt_data_fwding.dat** file for the simulation.

Simulation3 will have the "branch forwarding" design of part III where you add Branch Forwarding to the design of **Simulation2**. You should answer the questions in **Appendix C** and insert these to your report of **Simulation3**.
You should use the **SIM_HW6_TB_data_wt_data_and_branch_fwding.dat** file for the simulation.

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)
- **ISE** – here you should have the HW6_MIPS project created by the Xilinx ISE SW.

HW6 – The final MIPS

Part II

Adding Data Forwarding

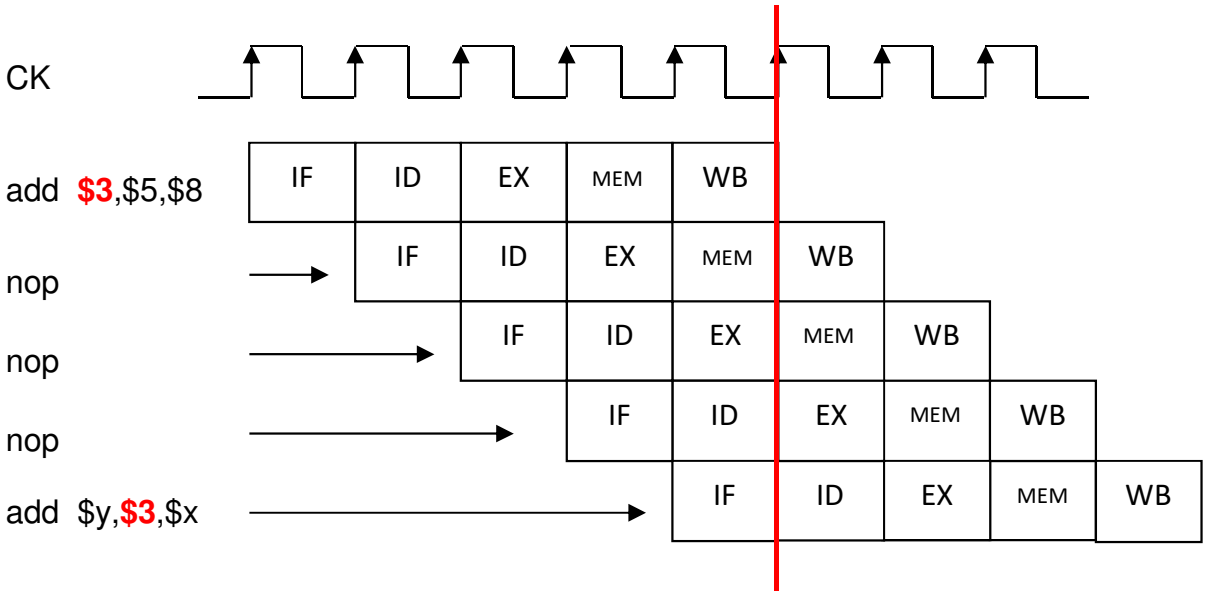


Fig. 2 – The pipelined MIPS latency

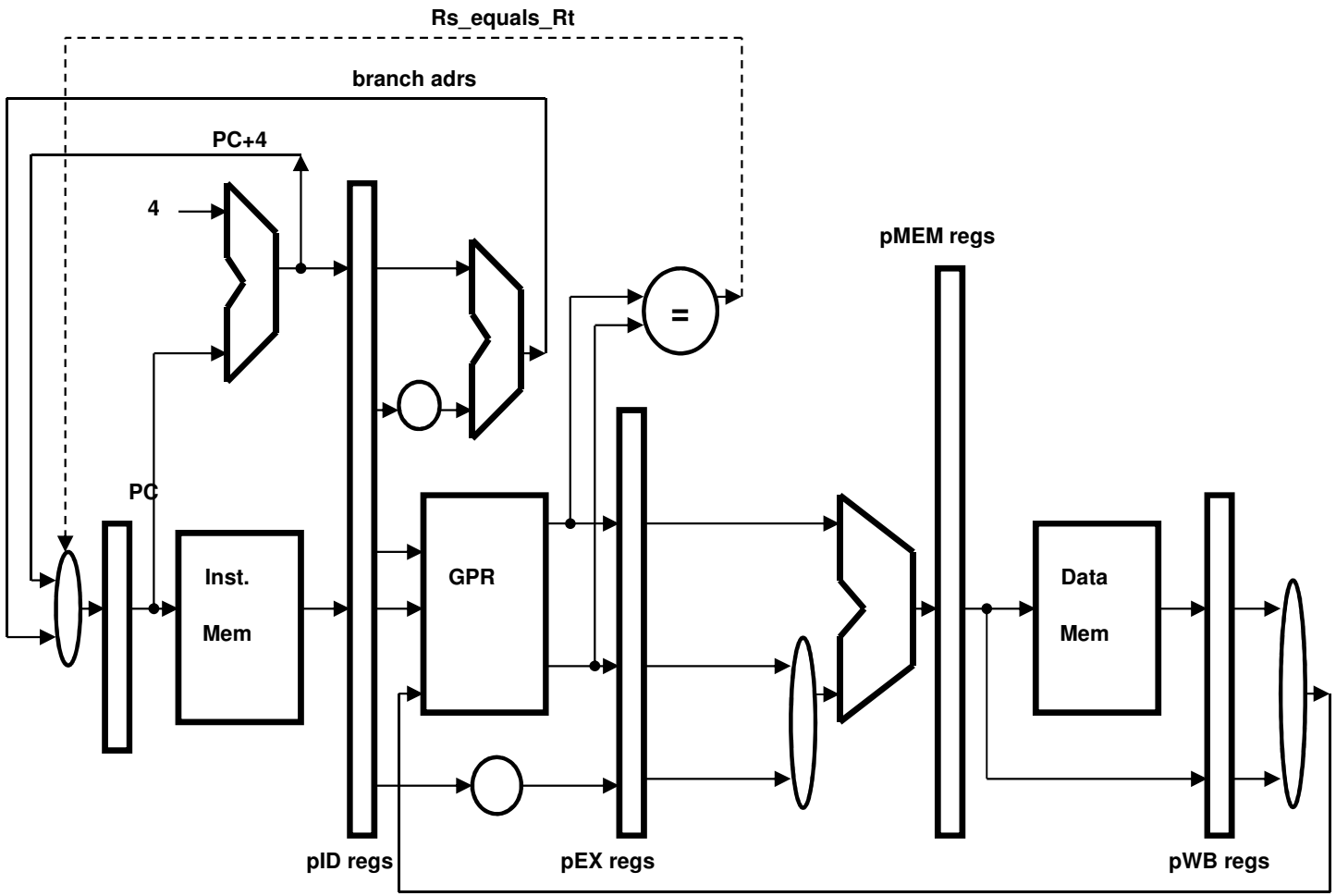


Fig. 4 – MIPS data path (part) with no forwarding

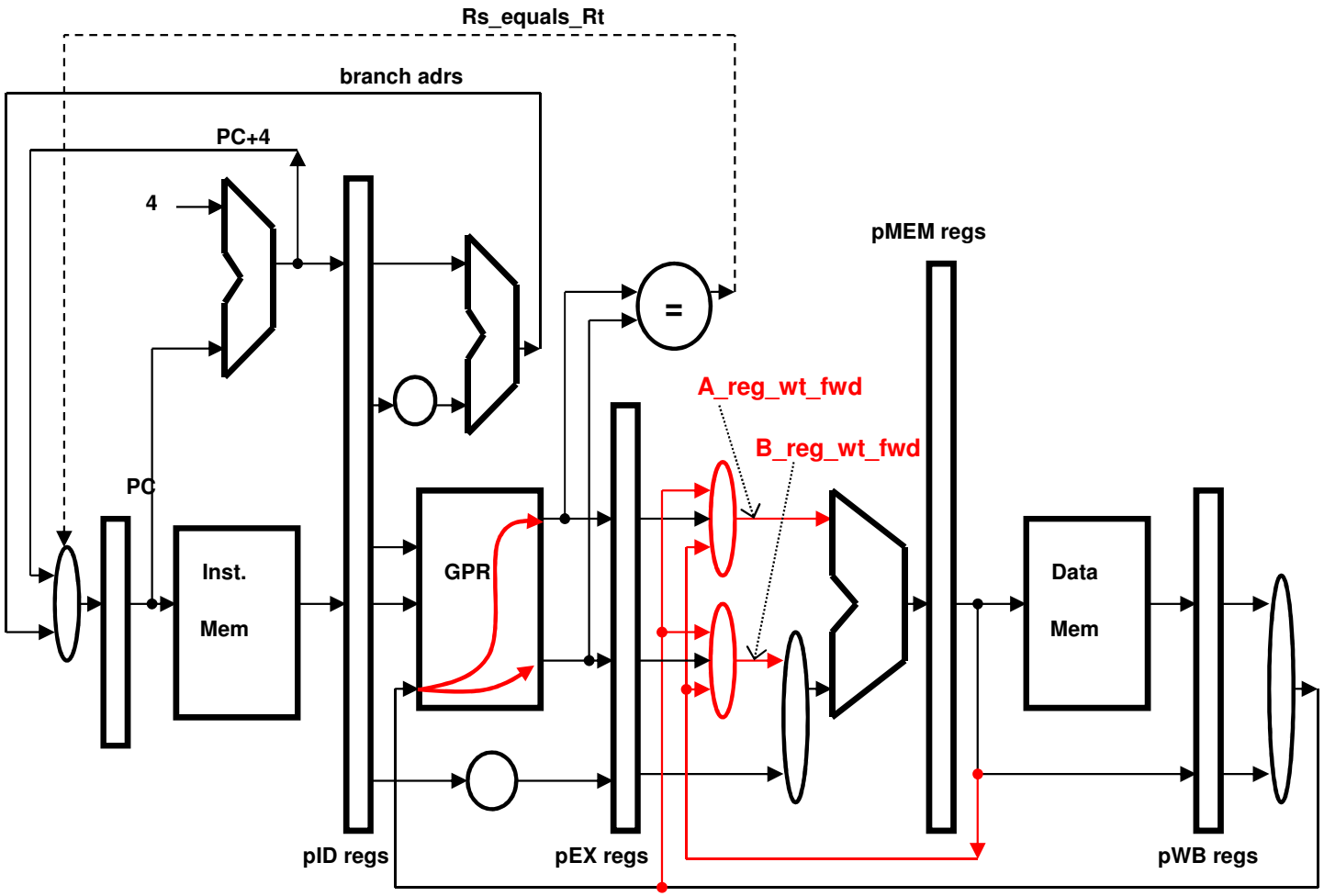


Fig. 5 – MIPS Data Path with Data Forwarding

You need to add the red signals

You need to handle also the B_reg_pMEM. To which signal should we connect it to?

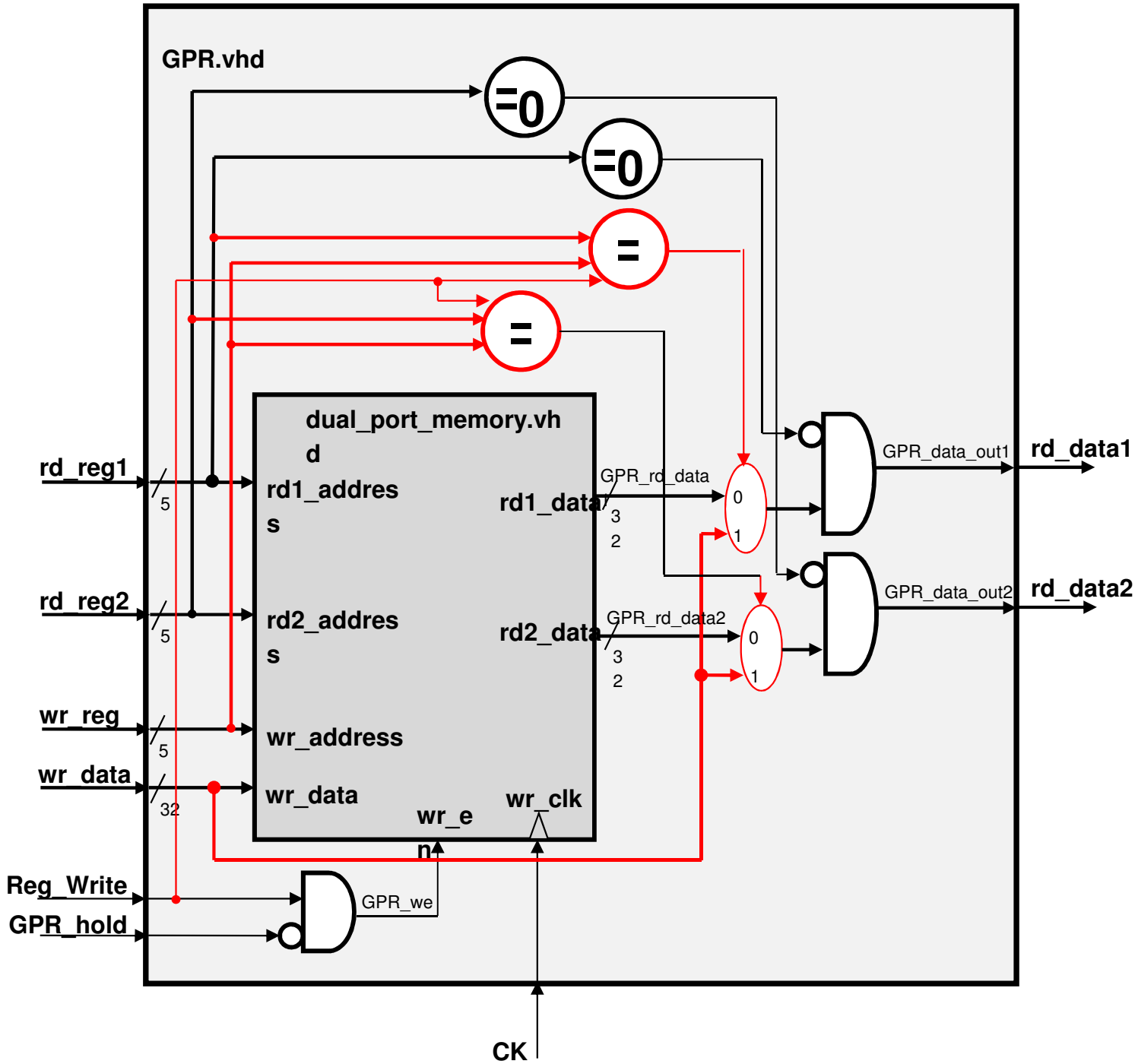


Fig. 6 – MIPS Data Path with Data Forwarding

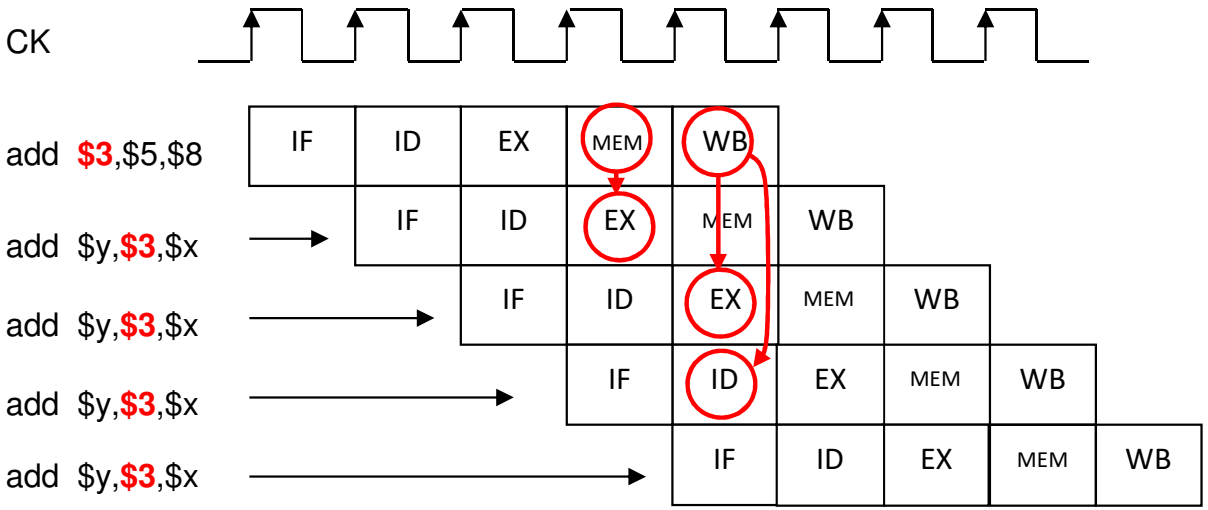


Fig. 3 – Data Forwarding timing diagram

(from the 1st instruction to future instructions)

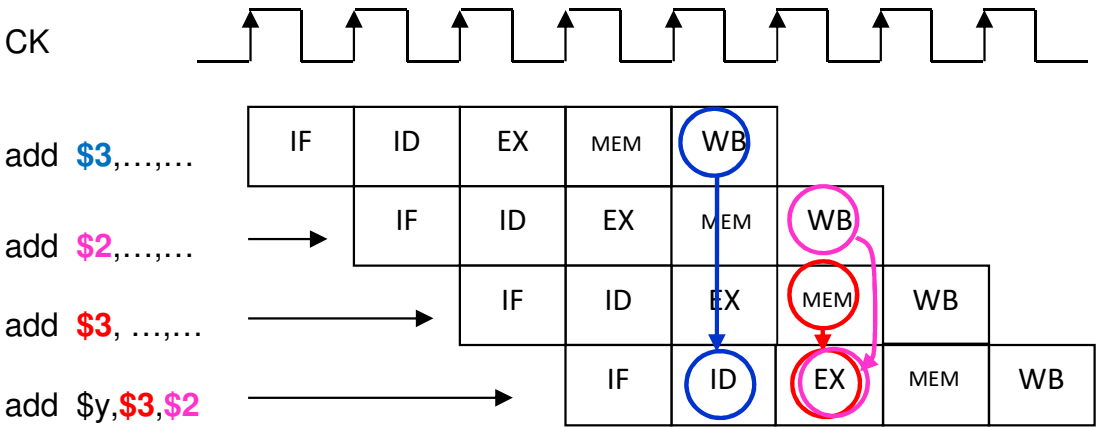


Fig. 3B – The 3 Data Forwarding options to an instruction

(to the 4th instruction from previous instructions)

HW6 – The final MIPS

Part III

Adding Branch Forwarding

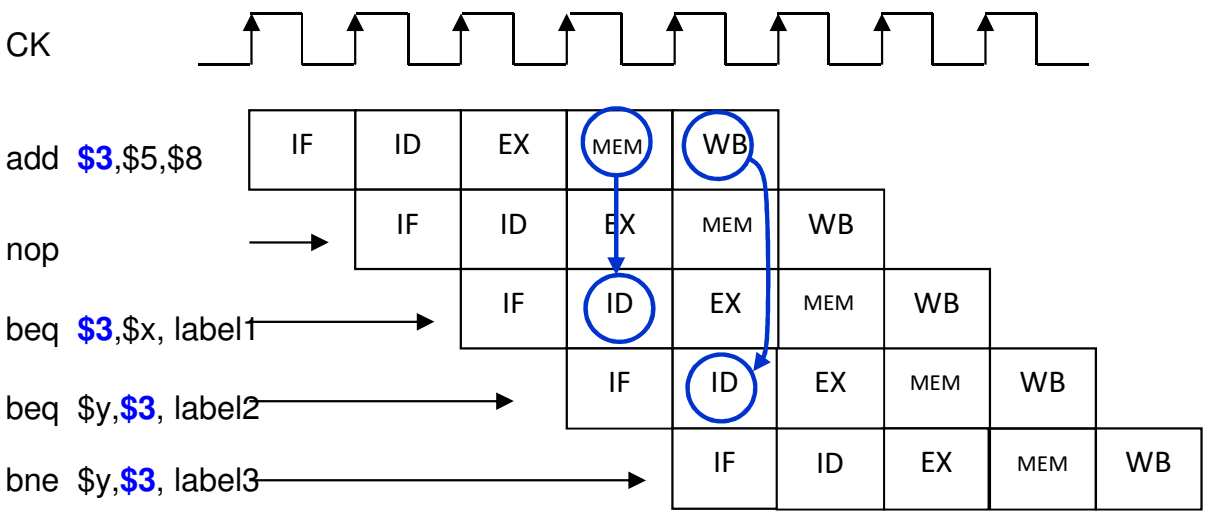


Fig. 7 – Branch Forwarding timing diagram

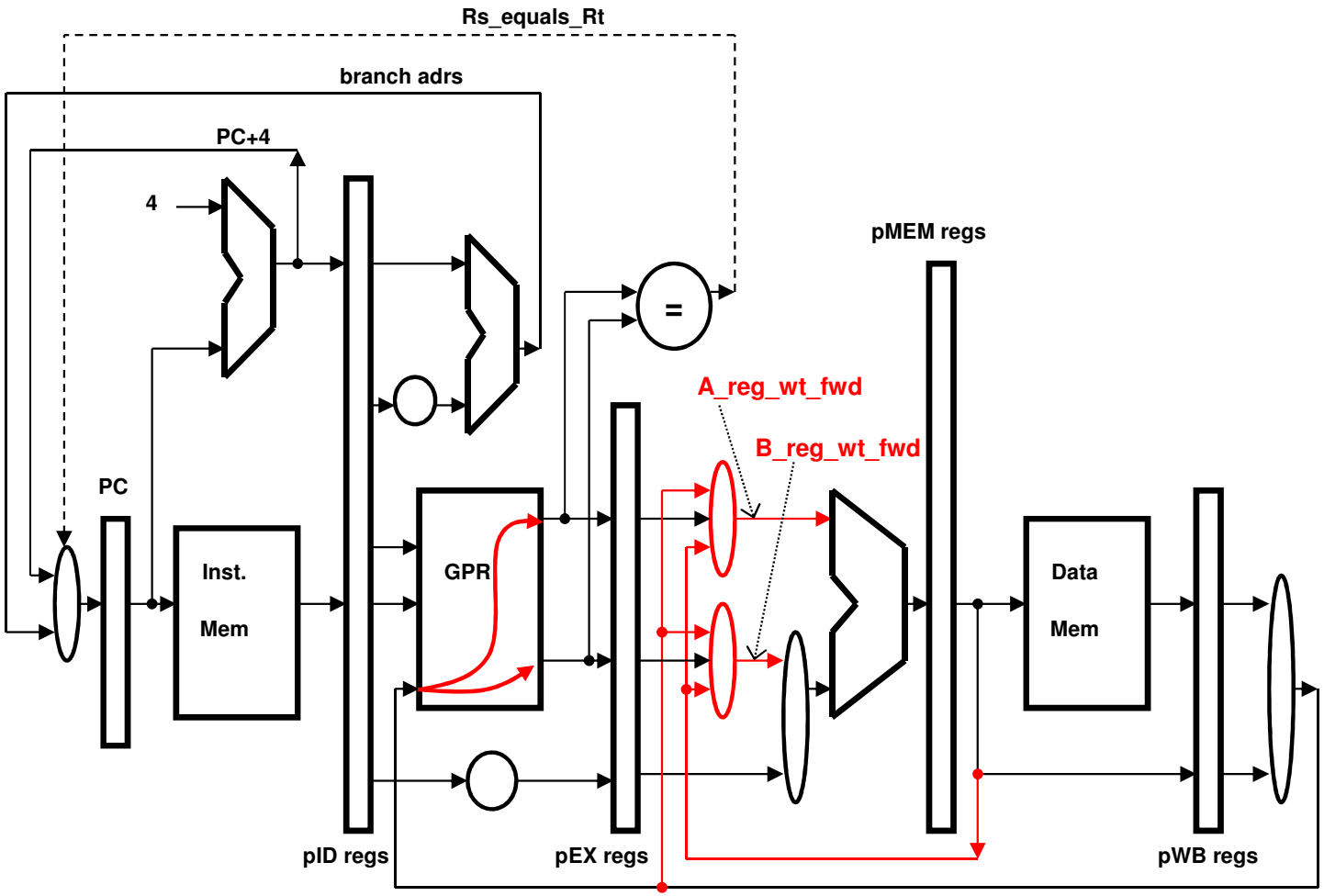


Fig. 5 – MIPS Data Path with Data Forwarding

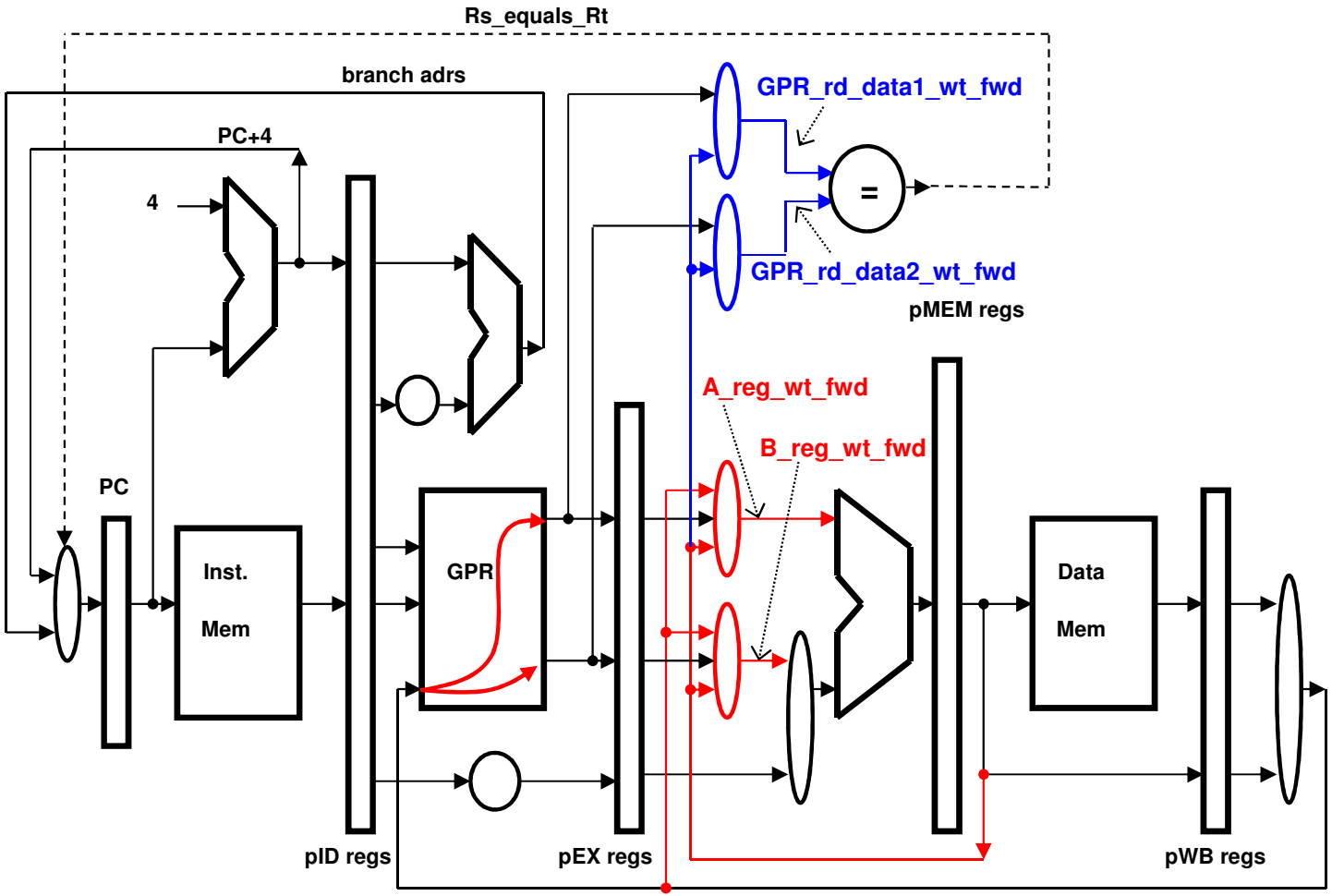


Fig. 8 – MIPS Data Path with Data and Branch Forwarding

You need to add the blue signals

You need to handle also the jr_address. To which signal should we connect it to?

All the new signals are listed in page 11 of the BYOC_HW6 doc.

Also in the "empty" vhd file they are already there and marked with "-- @@@HW6"

Now let’s talk on the Implementation phase

HW6_MIPS - implementation

1. Take the **HW6_top_4sim.vhd** and remove of all TB signals. Then rename it to **HW6_top.vhd**. Another way is to take the **HW6_top.empty** and integrate into it all the contents you made on the **HW6_top_4sim.empty** file in the simulation phase.

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.
- **HW6_top.vhd** – This is your design of HW6
- **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after modifications of HW4 & HW6
- **GPR.vhd** – your GPR File design you prepared in HW3 with the changes of HW6.
- **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 with HW6 modifications
- **BYOC_Clock_driver.vhd** – the CK divider & driver we use for implementation as of HW2.
- **BYOC_Host_Intf.ngc** - The actual infrastructure interfacing the PC.

4. Now run the Xilinx ISE SW, create a **HW6_top.bit** file and test it by running **Pong1_v32.txt** program.

HW6_top – testing the implemented design

We'll run that the **BYOCInterface** SW and load the IMem. Then run the circuit. If we have issues, we will 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 "**Pong1_v32.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 it in ck on mode. This is a simple Pong game and we should be able to control it via the right & left arrows on the keyboard.

For that we should connect a VGA screen to the Nexys2 board, and a PS2 keyboard

- 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 **four** directories/folders. The first is called **Simulation1**, the 2nd is called **Simulation2,** the 3rd is called **Simulation 3**, the 4th 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 HW6 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.

If the game works,

CONGRATULATIONS!!

You actually

built your own computer!!!

Enjoy the assignment!

Thanks for listening!