**BYOC course**

**Assignment #6**

**HW6 MIPS CPU**

**Group N.1**

**204200026**

**201322708**

**300267390**

**Appendices**

**Appendix A**

A.1) Fill up the following table describing what happens in each CK cycle in all instructions. You should specify the specific operations that are required for the execution of the instruction.

We filled in the Rtype and j instructions – as examples. We also gave the list of required registers & signals to be mentioned in the table, in the ori instruction line.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| phase | **IF** | **ID** | **EX** | **MEM** | **WB** |
| Instruction |
| Rtype | IR=IMem[PC]  PC= PC+4 | A=GPR[Rs]  B=GPR[Rt]  Active signals:  RegDst=’1’  RegWrite=’1’  ALUOP=”10”  MemToReg=’0’ | ALUOUT = A op B  Rd is chosen:  Rd\_pMEM=Rd\_pEX | ALUOUT\_pWB=  ALUOUT  (ALUOUT is delayed 1ck) | GPR[Rd\_pWB]  = ALUOUT\_pWB |
| addi | IR=IMem[PC]  PC= PC+4 | A=GPR[Rs]  sext\_imm  Active signals:  RegWrite <= '1';  RegDst <= '0';  ALUsrcB <= '1';  MemToReg <= '0';  MemWrite <= '0';  JAL <= '0';  ALUOP=”00” (add) | ALUOUT = A + sext\_imm\_reg  Sext\_imm\_reg = sext\_imm  Rd is chosen:  Rd\_pMEM=Rd\_pEX | ALUout\_reg = ALU\_output  (delay of 1 ck) | GPR\_wr\_data = ALUout\_reg\_pWB;  Rd\_pWB = Rd\_pMEM=Rt  GPR[Rt] = ALUout\_reg\_pWB |
| ori | Need to tell what is loaded to IR & PC – the relevant regs. | Again, all regs that are relevant (A, B, sext\_imm, PC in j & branch)  Also – all **active** signals created at the ID phase | All regs that are relevant (ALUOUT, B\_reg\_pMEM, Rd\_pMEM, sext\_imm) | All regs that are relevant (ALUOUT\_reg\_bWB, Rd\_pWB, MDR)  MDR= DMem[adrs ] or  DMem[adrs]=B\_reg\_pMEM | GPR[Rd\_pWB]  = ALUOUT\_pWB |
| lui | IR=IMem[PC]  PC= PC+4  Rs = “00000”(set when recognizing the opcode of lui) | A\_reg = 0x00000000  (set to zero)  sext\_imm <= imm(15 downto 0) & x"0000";  Active Signals:  RegWrite <= '1';  RegDst <= '0';  ALUsrcB <= '1';  MemToReg <= '0';  MemWrite <= '0';  JAL <= '0';  ALUOP=”00” (add) | sext\_imm\_reg = sext\_imm  ALUOUT = A + sext\_imm\_reg  Rd is chosen:  Rd\_pMEM=Rt\_pEX | ALUout\_reg = ALU\_output | GPR[Rd\_pWB] = aluout\_reg\_pWB  ALUout\_reg\_pWB = ALUout\_reg;  Rd\_pWB = Rd\_pMEM |
| beq | PC = PC +4  IR = IMem[PC] | Active Signals  ALUOP=”01”  If Rs\_equals\_rt =’1’ PC\_Source = “01”  Else  Pc\_source = “00”  Pc\_source =“00”  Then PC\_Plus\_4  PC\_source = “01”  Then branch\_adrs | - | - | - |
| bne | PC = PC +4  IR = IMem[PC] | ALUOP=”01”  The inverse of the above logic.  Pc\_source =“00”  Then PC\_Plus\_4  PC\_source = “01”  Then branch\_adrs | - | - | - |
| lw | PC = PC +4  IR = IMem[PC] | A= Rs  Sext\_imm = imm  Active Signals:  Aluop = “00”  Regwrite = ‘1’  Regdest =’0’  Alusrcb = ‘1’  Memwrite = ’0’  Memtoreg = ‘1’  JAL = ‘0’ | sext\_imm\_reg = sext\_imm  ALUout = A + sext\_imm\_reg  Rd is chosen:  Rd\_pMEM=Rt\_pEX | MDR= DMem[ALUout] | GPR\_wr\_data = MDR\_Reg (data from memory)  Rd\_pWB = Rd\_pMEM; |
| Sw | PC = PC +4  IR = IMem[PC] | A=Rs  B=Rt  sext\_imm = imm  Active Signals:  Aluop = “00”  Regwrite = ‘0’  Regdest =’0’  Alusrcb = ‘1’ Memtoreg = ‘0’  MemWrite = ‘1’  JAL = ‘0’ | sext\_imm\_reg = sext\_imm  ALUout = Rs (=A) + sext\_imm\_reg  B\_reg\_pMEM = Rt (=B) | DMEM[ALUout] = GPR[Rt] |  |
| j | IR=IMem[PC]  PC=PC+4 | PC= jump adrs | nothing | nothing | nothing |
| jal | IR=IMem[PC]  PC= PC+4 | PC = jump\_adrs  B = Rt = $31  Active Signals  ALUop = “00”  RegWrite = ‘1’  RegDst =’0’  ALUsrcB = ‘0’  MemToReg =”0”  MemWrite =”0”  JAL =’1’ | PC\_plus\_4\_pEx = PC\_plus\_4\_pID  Rd is chosen:  Rd\_pMEM=Rt\_pEX  JAL\_pEX = JAL | JAL\_pMEM = JAL\_pEX  PC\_plus\_4\_pMEM = PC\_plus\_4\_pEx | JAL\_pWB = JAL\_pMEM  Rd\_pWB = Rd\_pMEM  If JAL\_pWB = ‘1’ then  GPR\_wr\_data = PC\_plus\_4\_pWB  →  GPR[Rd\_pWB]= PC\_plus\_4\_pWB |
| jr | IR=IMem[PC]  PC= PC+4 | PC= jr\_adrs (=Rs)  Rd = $0 = 0 | - | - | - |

Answer the following questions.

A.2) Describe the changes done in order to support the ORI instruction.   
In order to prevent sign extension, we’ve modified the sign extension circuit in the fetch unit.

And added another case to address the ORI opcode -   
when b"001101" => sext\_imm <= x"0000" & imm(15 downto 0); -- ORI   
(this is Fuchsia! as you can clearly see).

Additionally, we’ve modified the ALU command control circuit and added the following case to support

the ORI instruction.  
when b"11" => ALU\_cmd <= b"001"; -- added OR when the command is ORI

Of course we’ve also modified the active control signals in the Decode phase to make sure the correct data paths are taken and the instruction is peformed correctly:

when b"001101" => -- ori

ALUOP <= b"11";

RegWrite <= '1';

RegDst <= '0';

ALUsrcB <= '1';

MemToReg <= '0';

MemWrite <= '0';

JAL <= '0';

A.3) Describe the changes done in order to support the LUI instruction.  
When executing the LUI instruction we write to the GPR file, therefore we’ve added the required changes to the active control signals selection in the decode phase. As follows:

when b"001111" => -- lui

ALUOP <= b"00";

RegWrite <= '1';

RegDst <= '0';

ALUsrcB <= '1';

MemToReg <= '0';

MemWrite <= '0';

JAL <= '0';

In addition, once the LUI instruction is executed, we automatically set the Rs value to the zero register – since it always contains 32 zero bits, in the Ex phase:

process(CK,RESET,Opcode)

begin

if RESET='1' then

A\_reg <= x"00000000";

elsif CK'event and CK = '1' and HOLD='0' then

if Opcode = b"001111" then -- case of LUI

-- Setting A\_reg to be the 0

A\_reg <= x"00000000";

else

A\_reg <= GPR\_rd\_data1;

end if;

end if;

end process;

We chose to set A\_reg to 0 to make sure that the value taken from the GPR contains 32 bit of zeros (logical zero).

Also, to finalize the implementation of the LUI instruction, we’ve modified the sign extension circuit to support the shifting of 16 bits when detecting the opcode of the LUI instruction. As can be seen below (you will definitely enjoy this ! )

When (=Opcode) b"001111" => sext\_imm <= imm(15 downto 0) & x"0000"; --LUI

A.4) Describe the changes done in order to support the JR instruction.  
Supporting this command was pretty easy, as a wise man once said (in the course bible).

We’ve changed the constant value previously assigned to the “jr\_adrs” to to be the “jr\_adrs\_in” coming from the “Top”, in the fetch unit:

jr\_adrs <= jr\_adrs\_in;

In addition, we’ve connected the Jr\_address in the “Top” (connecting to the jr\_adrs\_in in the fetch unit) to receive the GPR\_rd\_data1\_wt\_fwd – in order to supply the fetch unit with the relevant data once the Jr instruction has been executed.

jr\_address <= GPR\_rd\_data1\_wt\_fwd ; -- HW6 change to support fwding for Jr

In the fetch unit, we’ve modified the PC Source decoder, to detect the Jr instruction, and select the proper PC source, as can be seen below. (Selecting “10” will select the jr\_adress given from the “Top”). To achieve this we’ve addressed the “funct” bits of the command, as can be seen below:

when b"000000" =>

if funct = b"001000" then

PC\_source <= b"10"; -- jr

The above change will ensure we will take the correct data from the Jr\_address\_in and put it into the PC Source.

A.5) Describe the changes done in order to support the JAL instruction.

Supporting this command, phew, was very involving and … we felt immersed in the command.

BUT! In the end we did it.

First, we’ve modified the ID active control signal circuit to enable support of the JAL instruction. This means adding the JAL signal.

when b"000011" => -- JAL

ALUOP <= "00";

RegWrite <= '1'; -- a must to write to the GPR of return address

RegDst <= '0';

ALUsrcB <= '0';

MemToReg <= '0';

MemWrite <= '0';

JAL <= '1';

Inside the fetch unit, we’ve modified the PC Source decoder to select the “11” value which will ensure that we will use the jump address as the input to the PC source. (the address is given from the “Top”).

when b"000011" => PC\_source <= b"11"; --jal

As part of the JAL instruction we make sure that we write the PC\_Plus\_4 to register $31 in the GPR file.

To do so, we set the Rt value to “11111” (31 in binary!).

process(Opcode,IR\_reg)

begin

if Opcode = b"000011" then -- jal

Rt <= b"11111"; -- JAL support = to enable writing PC+4 to register 31

else

Rt <= IR\_reg(20 downto 16);

end if;

end process;

This is done by propagating both the address of the PC+4 to the GPR and the JAL instruction control signal until we reach the WB phase.

To propagate, as we’ve done in previous exercises we use registers that move the JAL instruction control signal value each ck cycle (each phase). Upon reaching the shores of the WB phase, we insert through the MemToReg mux the PC\_Plus\_4\_pWB value in to the GPR\_wr\_data. This will allow the inserting the value of the PC+4 into the GPR (via the GPR\_wr\_data variable).

--MemToReg mux --@@@HW6 requires changes to support JAL instruction

process(MemToReg\_pWB,MDR\_reg,ALUOut\_reg\_pWB,JAL\_pWB,PC\_Plus\_4\_pWB)

begin

if JAL\_pWB = '1' then

GPR\_wr\_data <= PC\_Plus\_4\_pWB;

Rt is forced to be $31 since we want to ensure that RegDst mux will choose the Rd\_pMEM to be Rt\_pEX (setting Rt\_pEX to Rd\_pMEM).