**BYOC course**

**Assignment #6**

**HW6 MIPS CPU**

**Group N.1**

**204200026**

**201322708**

**300267390**

**Appendices**

**Appendix C**

Answer the following questions.

C.1) What are the limitations due to the pipeline latency of the following combinations (assume Data Forwarding already exists):

* beq after add where the add Rd is the beq Rt
* beq after lw where the lw Rt is the beq Rs

Use a similar figure to Fig.2 and Fig. 3 to demonstrate your answers. Explain your answers!

C.1.a - beq after add where the add Rd is the beq Rt

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

nop

nop

add **$3**,$5,$8

WB

EX

ID

IF

Beq Rs offset($3)

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Since DF exists, we must wait 2 ck cycles after the add instruction in order to have the result of the add operation in the mem phase available during the ID phase of the beq instruction. The data will be forwarded from the transparent GPR, during the ID phase of the beq instruction.

C.1.b - beq after lw where the lw Rt is the beq Rs

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

nop

nop

lw **$3**,16($10)

WB

EX

ID

IF

Beq rs $3 offset

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

The answer from the previous question remains the same as it applies to both instructions, the same considerations apply:  
When the lw runs it stores the information from the DMEM into the GPR $3 address, this information will then be written during the WB phase of the lw instruction.

When running the beq instruction, to correctly access the newly retrieved information from the DMEM, we must wait 2 ck cycles after the lw and use the transparent GPR feature to access the information during the ID phase of the beq instruction.

C.2) What are the limitations of all cases of C.1 after you add the Branch Forwarding? . Explain your answers!

C.2.a - beq after add where the add Rd is the beq Rt

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

Beq Rs offset($3)

add **$3**,$5,$8

WB

EX

ID

IF

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

nop

Adding the branch forwarding allows to shorten the ck cycle delay by 1 instruction thanks to the modifications we’ve made – once the beq instruction reaches the ID phase where it loads the values for Rt and Rs the add reaches the MEM phase and the branch forwarding allows to extract the value from ALUout\_reg into the beq Rt value.

C.2.b - beq after lw where the lw Rt is the beq Rs

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

Nop

Nop

lw **$3**,16($10)

WB

EX

ID

IF

Beq $3 offset(Rt)

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Since the beq instruction will run after the lw, we must wait until the data is retrieved from the DMEM, only then will we be able to utilize it in the beq instruction. This mens that we will have to wait 2 ck cycles after the lw instruction in order to use the information as the beq Rs value (using data forwarding) –therefore the limitation from before has not been improved thanks to branch forwarding. This is possible thanks to the transparent GPR.

C.3) Why can’t we check the result of the previous instruction (time slot n-1) by a beq instruction following it (time slot n)?

Even with data forwarding and branch forwarding, the previous instruction before the beq instruction will reach the EX stage and will not finish the calculation in the ALU unit to utilize in the beq instruction (since we will want some information that is to be stored in the previous instruction – e.g. Rd in an add operation will be come Rs or Rt in the beq). Therefore we will have to wait an additional ck cycle.

C.4) List all of the limitations for Assembly programmer you can think of that still exist after adding the Data & Branch Forwarding circuits. . Explain your answer!

1. As we’ve seen we still must always wait in all Rtype instructions or I-type instructions for the ALU to finish at the minimum to use the result of the calculations in following instructions (wait 1 ck cycle).

2. (after solving the next question) – The shortest loop is always 2 instructions, because of the design of the MIPS pipeline.

3. In order to fully load 32 bits we must use 2 instructions LUI and ORI in combination for example.

4. You may not use add after an lw operation utilizing the same register (immediately after) – as seen above.

C5) What is the shortest loop code possible (not an infinite loop)? Any limitations? Explain in detail

The shortest instruction require – always – 2 stages at the least (2 ck phases) –

The IF and the ID. Instructions that create loops are J to the PC register itself – we will fetch it and after the ID phase we will jump to the same address we started from.

0x400000 : J 0x400000

0x400004 : nop (this instruction is required since until we jump we will already fetch and decode the nop)

This loop will cause the program to jump back to 0x400000 and run infinitely. 2 instructions are mandatory even though we will not do anything with second instructions, because of the design of the pipeline.

The shortest possible loop, as explained, consists of 2 instructions. If we modify or create a new jump instruction which injects into the IR a nop instruction during the ID phase – we will shorten the shortest loop command to a single instruction.