**BYOC course**

**Assignment #6**

**HW6 MIPS CPU**

**Group N.1**

**204200026**

**201322708**

**300267390**

**Appendix B**

Answer the following questions.

B.1) What are the limitations due to the pipeline latency of the following combinations:

* lw after add where the add Rd is the lw Rs
* lw after add where the add Rd is the lw Rt
* add after lw where the lw Rt is the add 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 answer!

B.1.a - lw after add where the add Rd is the lw Rs [ e.g., lw $4,16($3) ]

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

Nop

Nop

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

WB

EX

ID

IF

Nop

WB

EX

ID

lw Rt offset($3)

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer:  
Without data forwarding, the pipeline has inherent latency. In the case of an LW instruction after an ADD operation where the instruction uses the same register that the previous operation was about to write to, we will use an obsolete value, unless we wait until the add WB phase is over, only then can we access the GPR and obtain the correct Rs value (for thr Rt) since it has been written (the ID phase of the lw).

B.1.b - lw after add where the add Rd is the lw Rt

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

lw $3 offset(Rs)

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

WB

EX

ID

IF

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer: *There is no limitation in the sequence of given instructions*

The infor mation stored in the first add operation wil be overwritten by the information written in the lw instruction. (Since they both write to the same register). The limitation imposed here is not due to pipeline latency but due to the logic of the instructions (assuming this is intentional)

B.1.c - add after lw where the lw Rt is the add Rt

CK

WB

EX

ID

IF

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

MEM

WB

EX

ID

IF

nop

MEM

WB

EX

ID

IF

MEM

nop

WB

EX

ID

IF

MEM

nop

WB

EX

ID

IF

EX

add Rd, Rs, $3

MEM

Lw GPR[Rt] <- DMEM[GPR[$10]+16]

Add GPR[Rd] <- GPR[Rs] + GPR[Rt]

Answer:

The result of the LW is first written to the GPR in the WB phase of the intstruction, to the Rt register. The add operation requires the result in its ID phase(only then is the data actually already written to the GPR), therefore we must wait as described above – 3 ck cycles.

B.1.d - 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

nop

WB

EX

ID

beq $3 offset(rs)

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer:

Lw GPR[Rt] <- DMEM[GPR[$10]+16]

Beq if GPR[Rt]=GPR[Rs] → offset (branch offset)

As in previous questions, we must wait until the information is stored into the Rt register in the GPR (the WB stage of the lw operation) and only then, can we access the information correctly in the ID stage of the beq instruction. (as can be seen above) Therefore, we must wait for 3 ck cycles.

B.2) What are the limitations of all cases of B.1 after you add the Data Forwarding? . Explain your answer!

B.2.a - lw after add where the add Rd is the lw Rs

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

I : lw Rt offset($3)

II : lw Rt offsfet($3)

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

WB

EX

ID

IF

III : lw Rt offset($3)

WB

EX

ID

IV : lw Rt offset($3)

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer:

Add GPR[$3] = GPR[$5]+GPR[$8]

Lw GPR[Rt] = DMEM[GPR[$3]+offset]

Case I : Data forwading enalbes using the result of the add operation from the ALU output into the execute phase of the next instruction. Thus, allowing the lw operation to be peformed without latency of the pipeline. We handle this case by comparing the Rd\_pMEM to the Rs\_pEX, between instructions, if equal we take the result of the previous ALU operation.

Case II : No latency issues either, thanks to the data forwarding we can use the data written through the memtoreg MUX result, since forwarding allows this transfer of information instead of the register value stored in the GPR when the lw reaches the EX phase.

Case III: No latency issues either ! The lw will reach the ID phase, and the information stored in the GPR\_wr\_data will be used when reaching the ID phase (instead of the regular a\_reg source from the GPR).

 IV: Obviously after the WB phase there is no latency issue

B.2.b - lw after add where the add Rd is the lw Rt

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

lw $3 offset(Rs)

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

WB

EX

ID

IF

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer:

There is no limitation here, just like the previous question (without forwarding). Since we will overwrite the $3 register either way, regardless of the timing of the instruction. (so we have no problem with the data loaded into the $3 in the add WB and the data written in the WB phase of the LW (or ID phase).

B.2.c - add after lw where the lw Rt is the add Rt

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

nop

add Rd Rs $3

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

WB

EX

ID

IF

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer

In order to achieve the correct result, we must wait 1 Ck cycle and wait for the WB phase to use the value of $3 in the add instruction (This is supported by the data forwarding circuit in the EX phase of the add).

For usage after the above drawing, we can execute the add instruction at will any ck cycle afterwards thanks to the support of forwarding information from the transparent GPR in the ID phase.   
\* in the last add – 3 clock cycles afterwards – we will already have the new information written after the WB phase.

B.2.d - 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(rs)

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

Answer

Data forwarding enables the result of the lw instruction (written into the Rt register) to be available to the beq instruction only after waiting 2 Ck cycles and only then running the beq instruction.

We cannot perform the instruction earlier as the information must be written in to the GPR from the memory in the lw WB.

This is thanks to transparent GPR, in ID phase of the beq the information from the WB is already available. (see above drawing).

\*Of course no limitation exists if we execute the beq instruction 4 Ck cycles afterwards

B.3) How many times do we perform the instruction following a jal instruction? Explain in detail. What are the implications? If this is a problem, what do you suggest in order to solve it?

We perform the instruction following the JAL instruction twice – this is because until we jump we still fetch,decode and run the following instruction in the pipeline. When returning to the address written into the $31, it stores the PC+4, which is exactly the instruction following the JAL instruction. In order to avoid performing the same instruction twice, we can place a nop instruction immediately after the JAL instruction.

B.4) How soon after jal instruction can we issue a jr $31 instruction in order to return to the right location in the code? Give the answer before data forwarding is added and then after the data forwarding is added. . Explain your answer!

No data forwarding:

With no data forwarding, we must wait until the register $31 is written with the contents of the PC\_plus\_4 data. This is due to the fact that we must wait until the WB phase of the JAL routine to store the value propagated by the JAL instruction.

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

jal routine1

WB

EX

ID

IF

WB

EX

ID

Jr $31

IF

CK

MEM

MEM

MEM

MEM

MEM

EX

With data forwarding:

When performing a Jr instruction, we must have the $31 register already available to us in the ID phase of the Jr instruction. This imposes a limitation as we must wait 2 ck cycle after the JAL instruction for the information to be available. During the WB phase of the JAL routine – we will be able to access the output of the data that will be written to the GPR through the transparent GPR and therefore – will be able to execute the Jr $31 instruction.

WB

EX

ID

IF

WB

EX

ID

IF

WB

EX

ID

IF

nop

nop

jal routine1

WB

EX

ID

IF

Jr $31

WB

EX

ID

IF

CK

MEM

MEM

MEM

MEM

MEM

EX