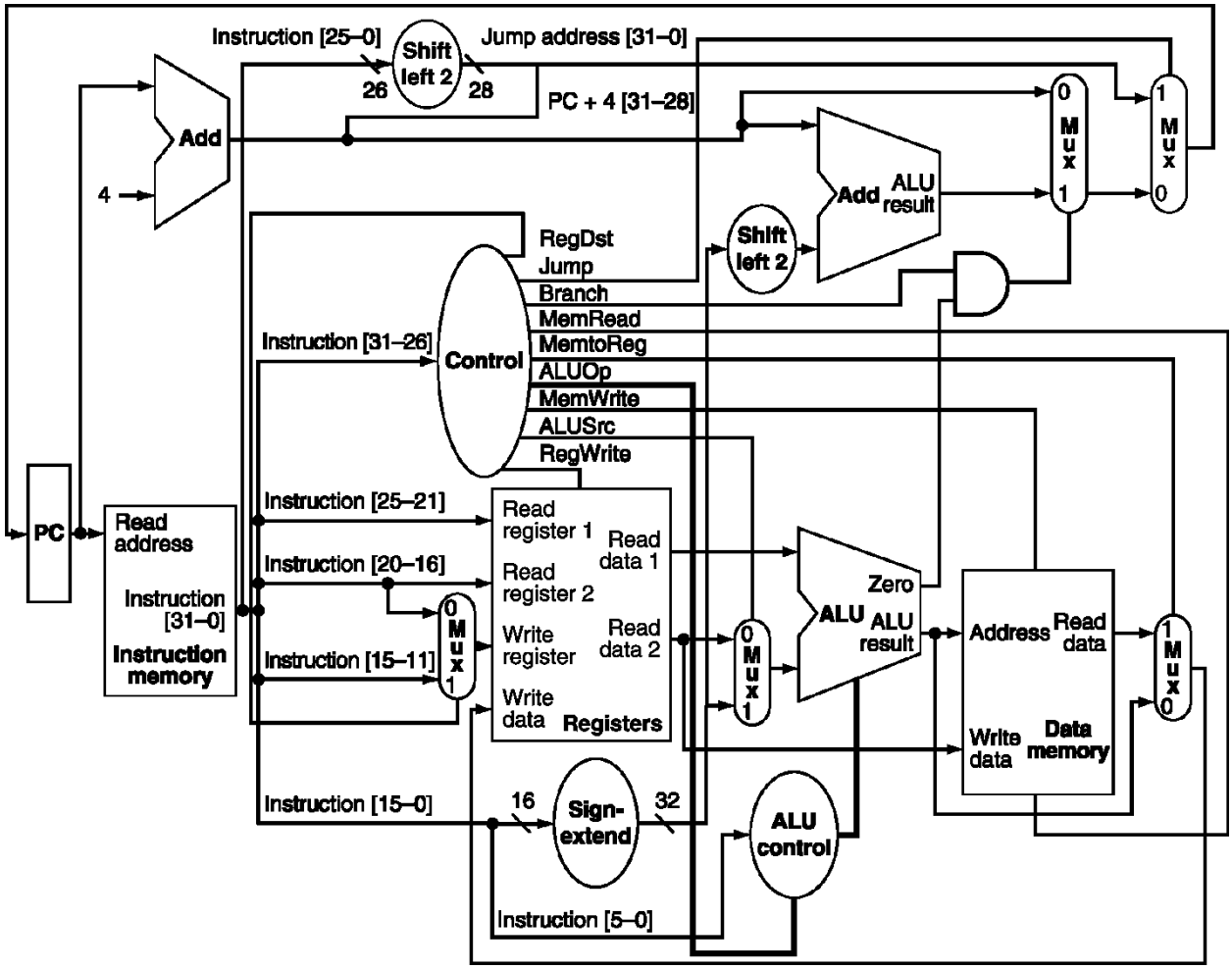
HW#10 (CSC390-sp2018)
Due: 04/27/2018 by 4:00PM
[Ref pages 262- 271 of your text book]

Q1. Consider the R-type instruction, **add $s0, $a1, $t7**, and its corresponding machine code:

000000 00101 01111 10000 00000 100000

The machine code consists of mainly Opcode, Source registers, destination registers, and Function code etc.

Now suppose the instruction is executed in the MIPS processor as shown in the following figure. Highlight the different **registers, control signals** and **multiplexer inputs** that will be activated during the execution process of the above **add** instruction. Also **indicate the buses** where different portions of the machine code and the result of $a1+$t7 can be found.

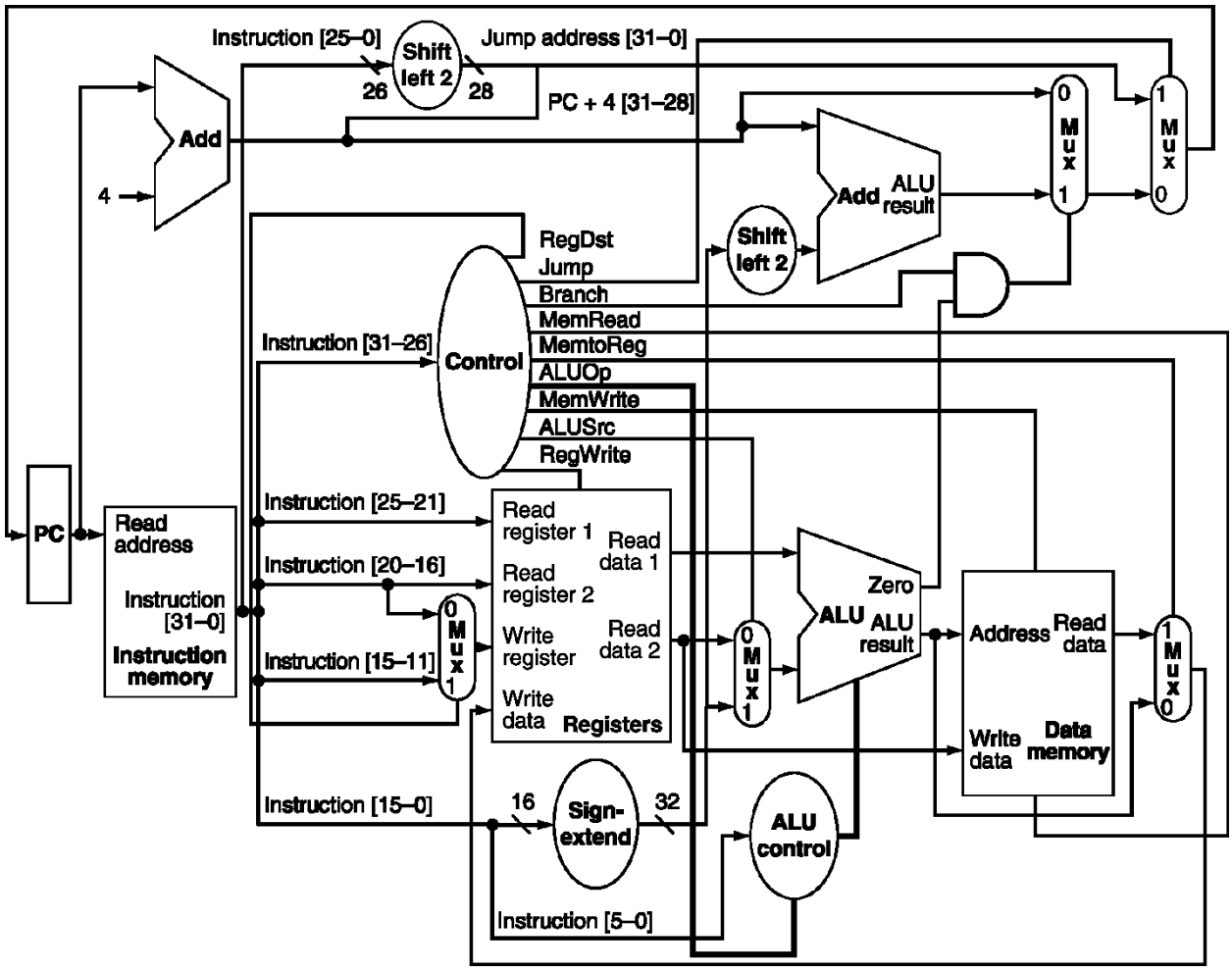


Q2. Consider the load-type instruction, **lw $s0, 32($t2)**, and its corresponding machine code:

100011 01010 10000 0000000000100000

This machine code consists of Opcode, Source registers (rs), destination registers (rt), and 16 bits offset.

Now suppose the instruction is executed in the MIPS processor as shown in the following figure. Highlight the different **registers, control signals and multiplexer inputs** that will be activated during the execution process of the above **lw** instruction. Also **indicate the buses** where different portions of the machine code and the content of **32($t2)** can be found.

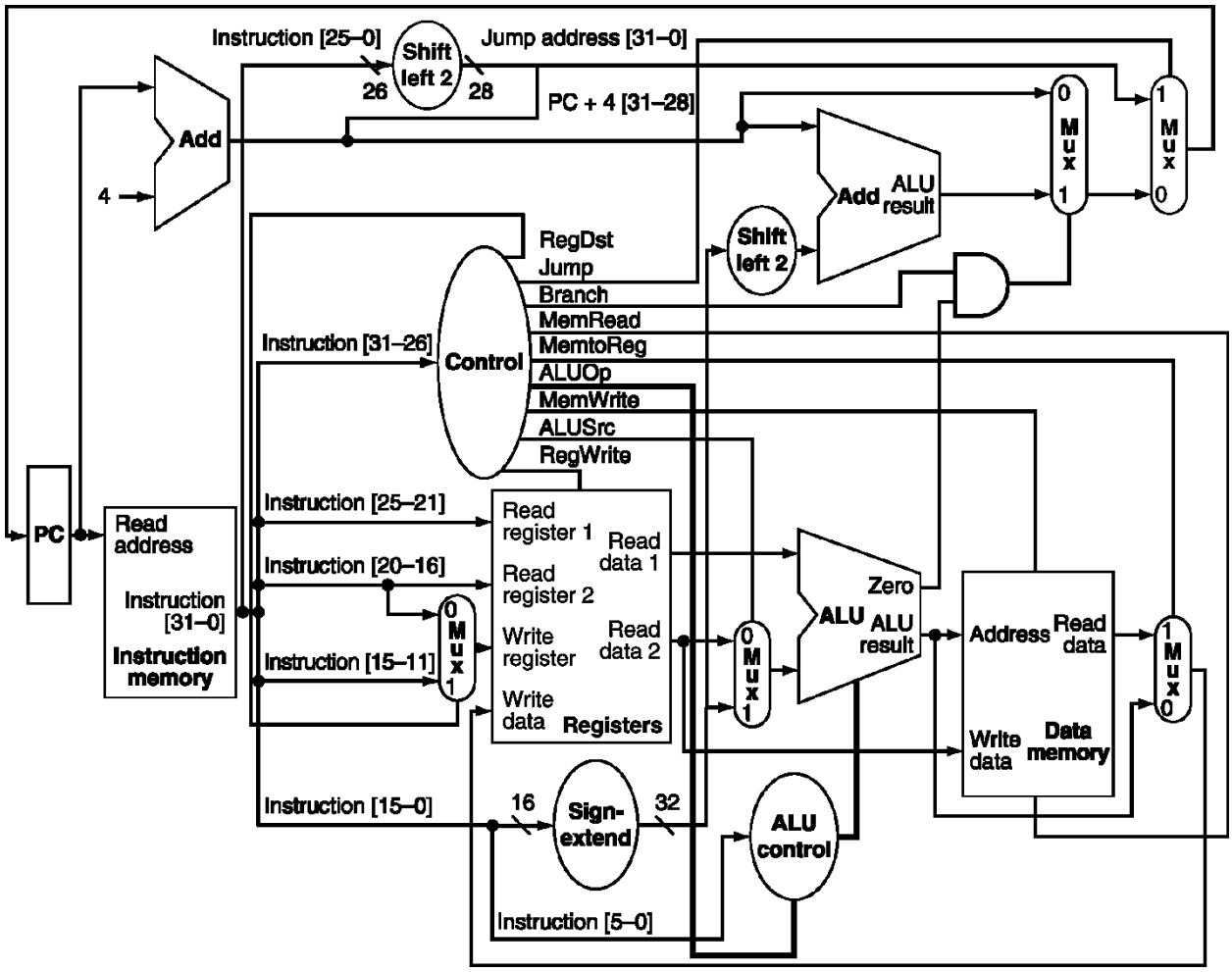


Q3. Consider the branch type instruction, **beq $s3, $s4, Level1**, and its corresponding machine code:

000100 10011 10100 0000000000000010

The machine code consists of mainly Opcode, source registers (rs and rt), and 16-bit offset. Here the offset, Level1, is arbitrarily chosen as 0000000000000010.

Now suppose the instruction is executed in the MIPS processor as shown in the following figure. Highlight the **different, registers, control signals and multiplexer inputs** that will be activated during the execution process of the above **beq** instruction. Also indicate the buses where different portions of the machine code and the result of the physical address of Level1 can be found. Let's assume the content of $s3 and $s4 are equal.



Q4. Pipelining is an implementation technique in which multiple instructions are overlapped in execution to improve the performance by increasing the instruction throughput of the system. However, there are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. There are three different types of hazards, such as structural, data, and control hazards. Consider the following lines of codes and determine where you could have possible hazards. Also indicate the types of Hazard.

and $t2, $s1, $s2

sub $s4, $t2, $s3

or $s7, $s6, $t2

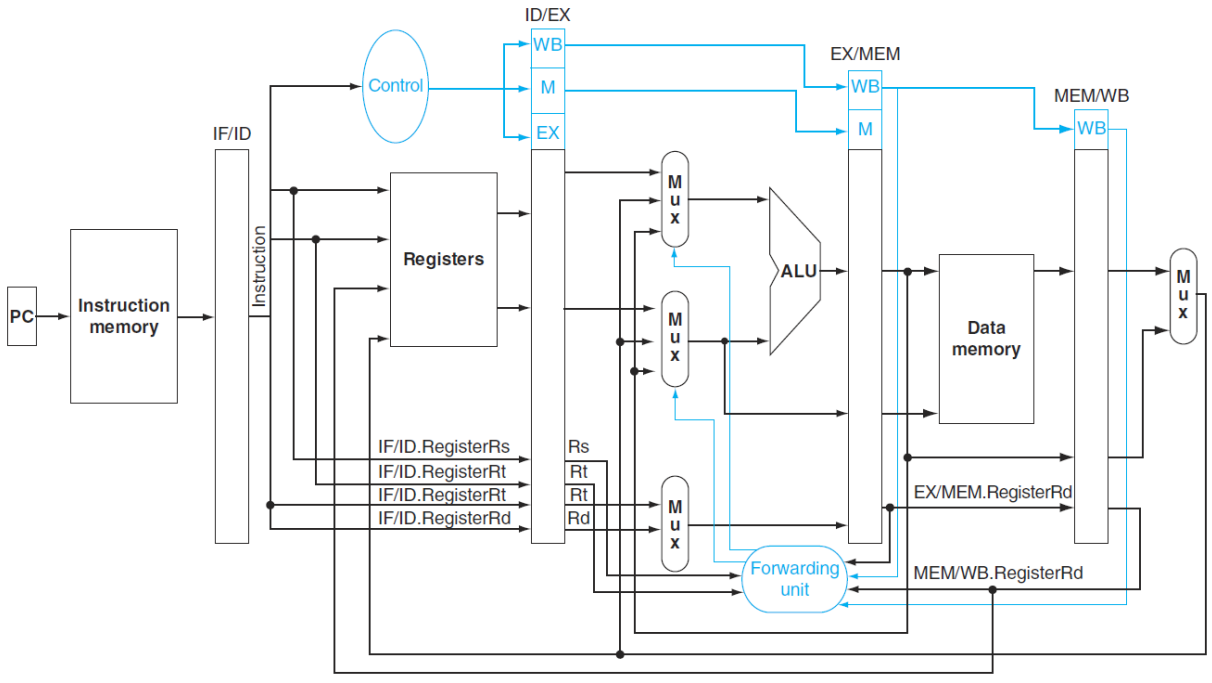
add $t7, $t2, $t5

beq $t2, $s8, exit

sw $s9, 100($t2)

To support the pipelining operation, MIPS processors use **pipeline-register,** in between the pipeline stages. There are four pipeline registers (IF/ID, ID/EX, EX/MEM and MEM/WB registers) in the MIPS architecture (as shown in the following figure). Pipeline registers temporarily hold the control signals and register numbers to facilitate the data forwarding or bypassing, an efficient way of overcoming the data hazards.

Write a short notes on how you can detect the above data hazards by comparing the pipeline registers (such as EX/MEM.RegisterRd, ID/EX.RegisterRs and ID/EX.RegisterRt registers) and using the pipeline-register control signals (such as EX/MEM.RegWrite, MEM/WB.RegWrite).



Q5. One way to improve branch performance is to reduce the cost of the taken branch. Thus far, we have assumed the next PC for a branch is selected in the MEM stage, but if we move the branch execution earlier in the pipeline, then fewer instructions need be flushed and thereby pipelining would become more efficient. The designers observed that many branches rely only on simple tests (equality or sign, for example) and that such tests do not require a full ALU operation but can be done with at most a few gates. Figure 4 shows the datapath and control signals of such a MIPS processor. **Explain,** how the **beq** instruction can be tested using simple gates in the ID stage0.

Show what happens when the branch is taken in the following instruction sequence, assuming the pipeline is optimized for branches that are not taken and the branch execution is moved to the ID stage:

```
10  add $10, $4, $6

14 beq $1, $7, 8

18 or $13, $2, $6

22 and $12, $2, $5

24 add $14, $4, $2
……..
50 sw  $4, 50($7)
```

Use **figure 4** (a & b) to show what happen when a branch is taken. Specifically, **indicate the input/output values of the PC, adders, pipeline register data, and control signals.**

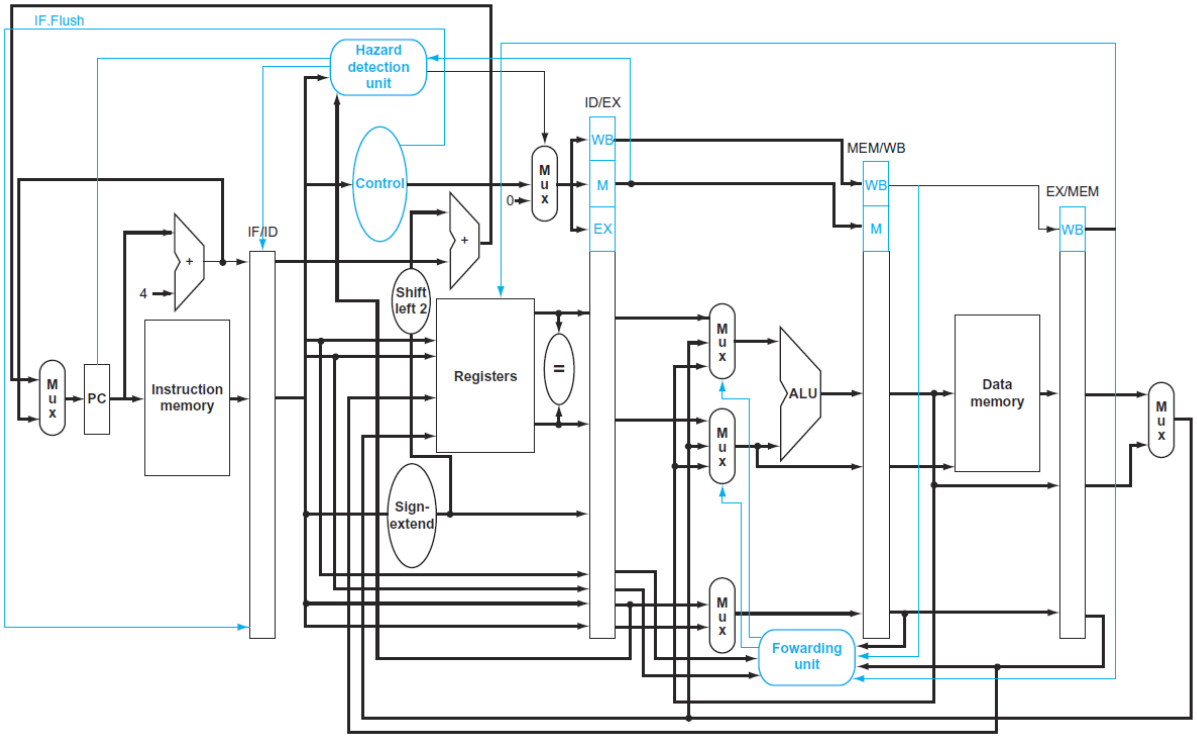


Figure 4 (a)

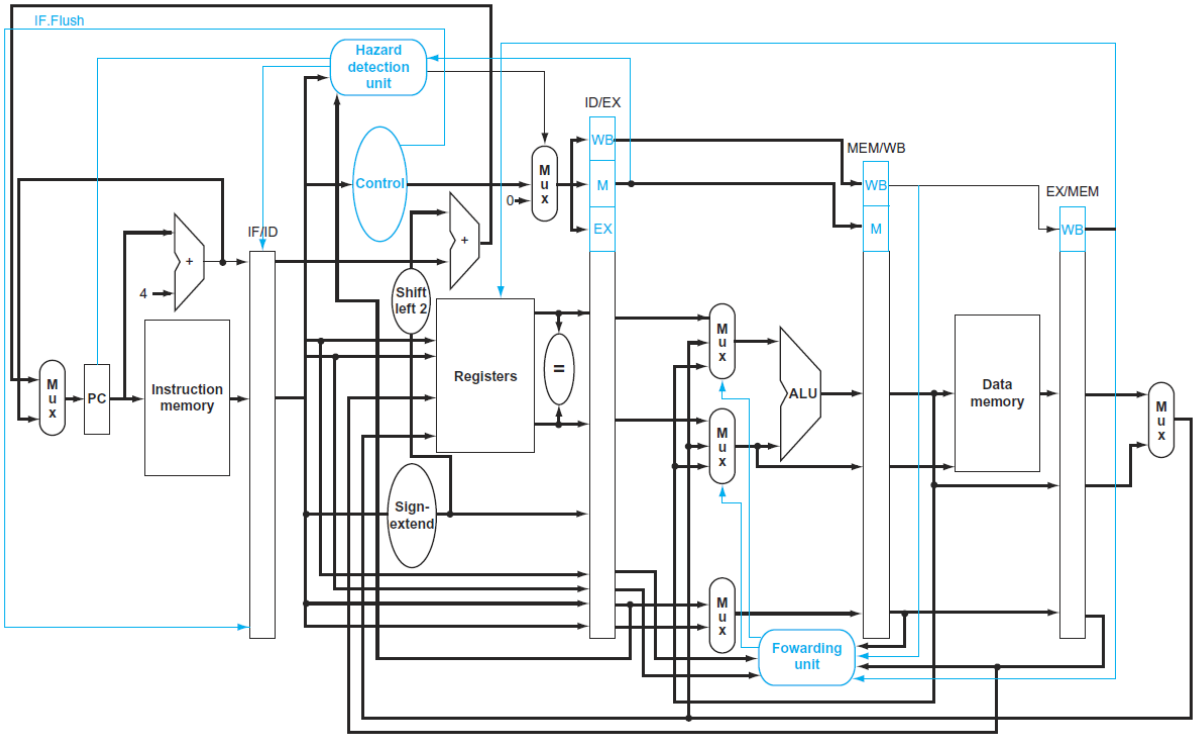


Figure 4(b)