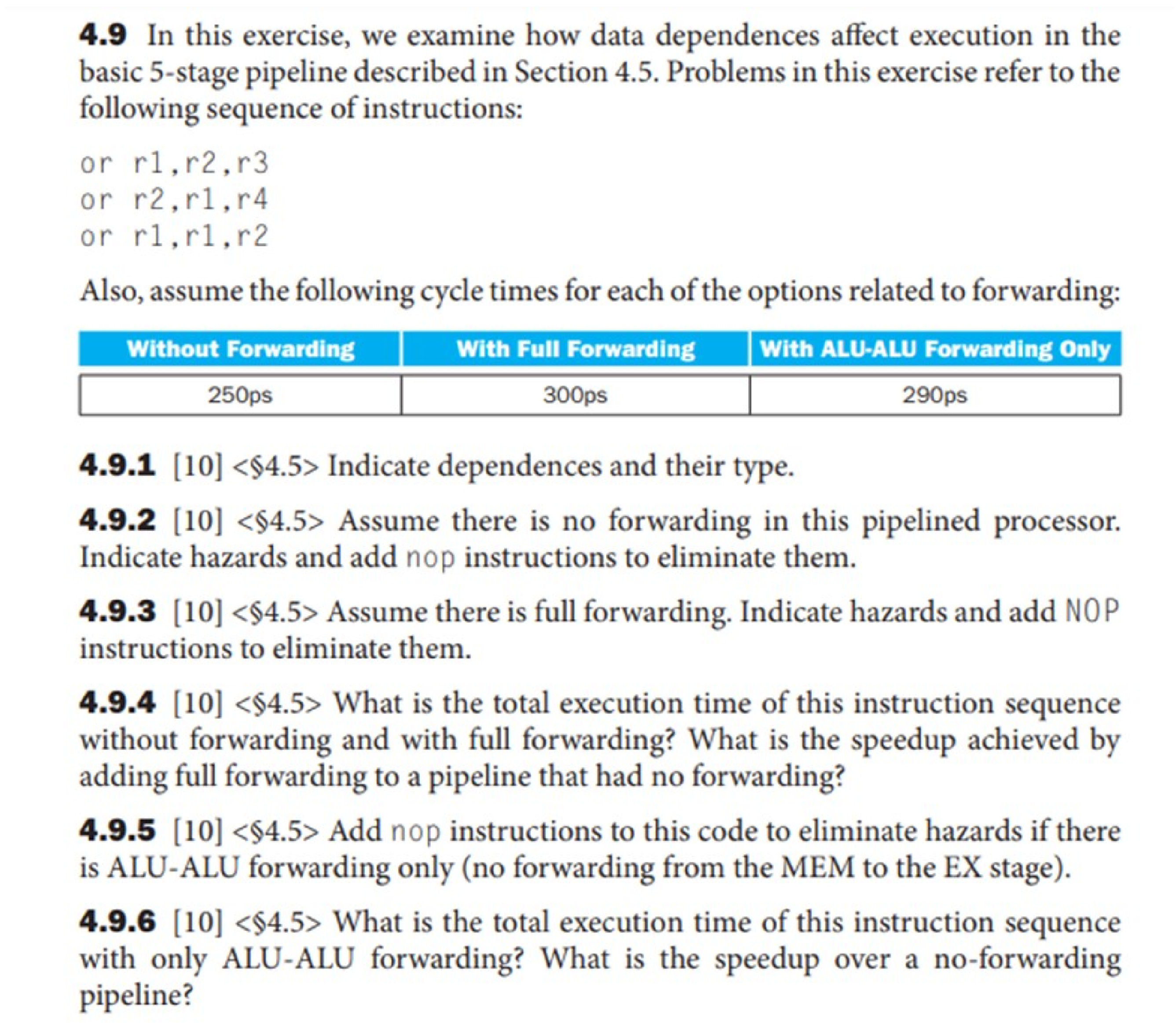
ЛУ 5

задача 1: Да се реши



4.9 In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

4.9.1 [10] <§4.5> Indicate dependences and their type.

4.9.2 [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

4.9.3 [10] <§4.5> Assume there is full forwarding. Indicate hazards and add `NOP` instructions to eliminate them.

4.9.4 [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.9.5 [10] <§4.5> Add `nop` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

4.9.6 [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

==

Задача 2: Да се реши

4.15 The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-Type	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

4.15.1 [10] <§4.8> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

4.15.2 [10] <§4.8> Repeat 4.15.1 for the "always-not-taken" predictor.

4.15.3 [10] <§4.8> Repeat 4.15.1 for for the 2-bit predictor.

4.15.5 [10] <§4.8> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.15.6 [10] <§4.8> Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?

==

Зад 3: Да се реши

4.16 This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: `T, NT, T, T, NT`

4.16.1 [5] <§4.8> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

4.16.2 [5] <§4.8> What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken)?

4.16.3 [10] <§4.8> What is the accuracy of the two-bit predictor if this pattern is repeated forever?

4.16.4 [30] <§4.8> Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever. You predictor should be a sequential circuit with one output that provides a prediction (1 for taken, 0 for not taken) and no inputs other than the clock and the control signal that indicates that the instruction is a conditional branch.

4.16.5 [10] <§4.8> What is the accuracy of your predictor from 4.16.4 if it is given a repeating pattern that is the exact opposite of this one?

4.16.6 [20] <§4.8> Repeat 4.16.4, but now your predictor should be able to eventually (after a warm-up period during which it can make wrong predictions) start perfectly predicting both this pattern and its opposite. Your predictor should have an input that tells it what the real outcome was. Hint: this input lets your predictor determine which of the two repeating patterns it is given.

==

Теория

Branch Predictor - Това е цифрова електроника целяща да предвиди дали един условен преход (скок) ще бъде взет или не. Съществуват различни видове Branch predictor-и, като едни от най-простите предвиждат че скок винаги ще бъде взет (**Always taken**), или обратното, никога няма да бъде взет (**Always not taken**). Друг метод е да се пази история на предишни скокове. Примерно в 1 бит да се пази информация предишния преход дали е бил взет (1) или не (0). Съществуват и много по-сложни и изтънчени решения.

В на настоящото упражнение ще използваме т.н **2-bit Predictor**, който може да се представи като finite-state машина. Като в зависимост от състоянието в която тя се намира и от новопостъпилият сигнал (**brach taken** or **not taken**), се преминава в последващо състояние. Като Predict **strongly taken** (11) & Predict **weakly taken** (10) Предвиждат че преходът е взет, а **strongly not taken** (00) & **weakly not taken** (01) предвиждат че преходът не е взет.

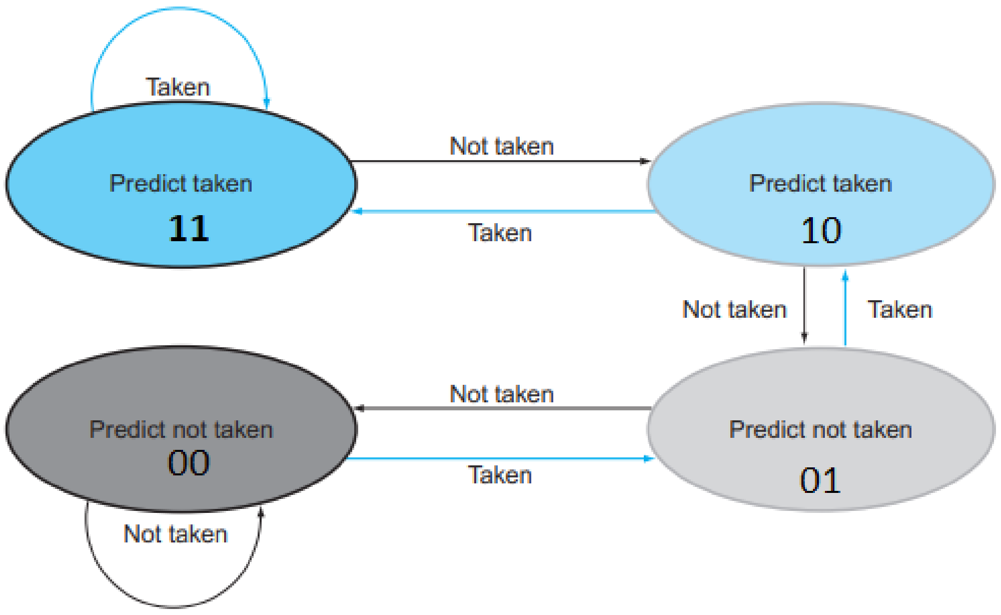
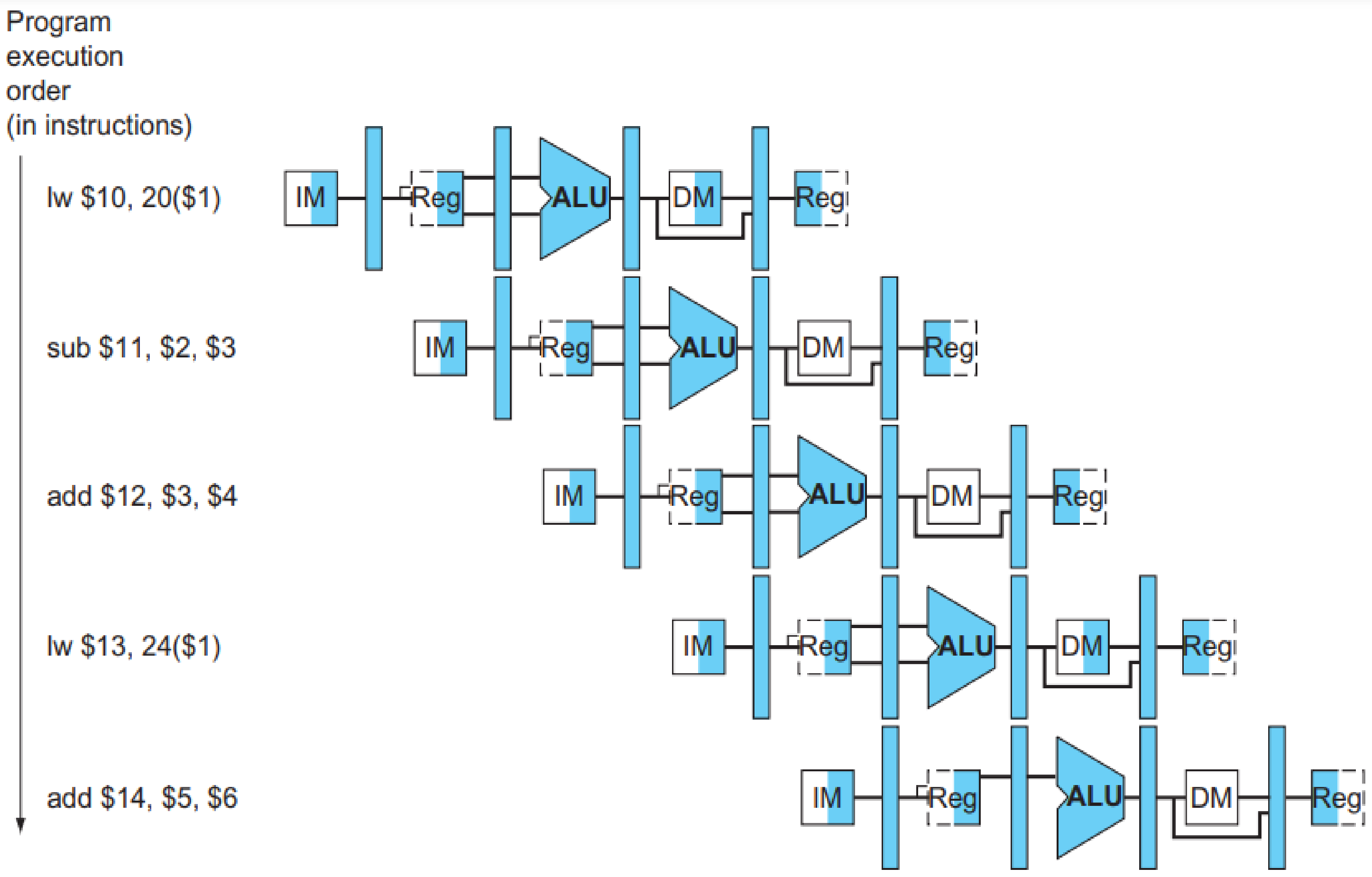


FIGURE 4.63 The states in a 2-bit prediction scheme. By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted only once. The 2 bits are used to encode the four states in the system. The 2-bit scheme is a general instance of a counter-based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the mid-point of its range as the division between taken and not taken.

Branch predictor-а е важен защото вместо да чакаме условието да бъде изчислено за да разберем от кой адрес ще извличаме последващи инструкции (което отнема множество цикли при **pipeline**), можем веднага да започнем да изпълняваме инструкции от адрес от който сме предвидили че ще е правилен. Ако този адрес се окаже неправилен. Трябва да изхвърлим лошите инструкции от нашия pipeline и да продължим изпълнението от правилния адрес.



Program execution order (in instructions)
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
IM
Reg
ALU
DM
Reg