**Computer Science 605.611**

**Problem Set 8**

1. A degree-N superscalar system can execute up to N instructions at the same time. A degree-N superpipelined system allows up to N instructions to occupy each pipeline stage at the same time. A scalar pipeline allows at most one instruction in each stage at a time.

a) (5) Consider a superscalar system that contains 2 integer units, 1 load/store unit and 1 floating point unit, each with its own separate reservation station. If the reservation station for the load/store unit becomes full, how does this impact the system’s ability to issue additional instructions? Explain your answer.

The systems ability to issue additional instructions is impacted in a few ways. The simplest one to discuss would be the load/store instructions. As there is now nowhere to hold them in a reservation station, load/store instructions would need to be stalled until the system makes room. Other non-load/store instructions wouldn’t be affected as they go to the other units (as long as the load/store unit is unneeded). Overall, the filling of a load/store unit happens, the system will most likely need to slow down due to necessary stalls so the instructions can be added to the reservation station.

b) (5) An alternative to using a separate reservation station at the front end of each functional unit is to merge multiple reservation stations into a single collection of reservation stations (sometimes called an instruction window). The instruction window is shared by all functional units and results on the common data bus are broadcast to all entries within the instruction window. The entries in the instruction window are repeatedly scanned to detect instructions that are ready to be issued to the appropriate available functional unit. How is the system’s ability to issue instructions impacted if the instruction window becomes filled with load/store instructions?

The systems ability to issue instructions is impacted much more heavily if the window becomes filles with load/store instructions. Unlike in the last questions, since there is a universal station that is filled, no instructions can be issued until there is space. The system also would not be able to utilize its ILP as well since the corresponding instructions to certain instructions are not available in the window. The system as well would face a lot of broadcast overhead as the results of load/store instructions tend to not be as useful to instructions that are waiting.

2. a) (5) What clock rate is required for our 5-stage scalar pipeline to match the maximum throughput of a degree-4 superscalar version of the system with a 2 GHz clock rate?

Minimum clock rate required for the 5-stage scalar pipeline = \_\_\_8 GHz\_\_\_\_

(This is because the degree 4 superscalar pipeline runs 4 instructions per cycle)

b) (5) What is the minimum number of nano-seconds required to complete 12 instructions on a degree-8 superpipelined version of the system with a 1 GHz clock rate? Express your answer as an integer.

Minimum number of nano-seconds = \_\_2 ns\_\_\_\_

3. (5) A superscalar system employs reservation stations and dynamic instruction scheduling. Based on the instruction sequence shown below, what is the minimum number of functional units (i.e., execution units) required by the system to execute the instruction sequence?

ori $2, $2, 25

xor $3, $3, $3

add $11, $2, $3

add $4, $5, $11

mtc1 $4, $f12

cvt.s.w $f12, $f12

sub.s $f14, $f16, $f18

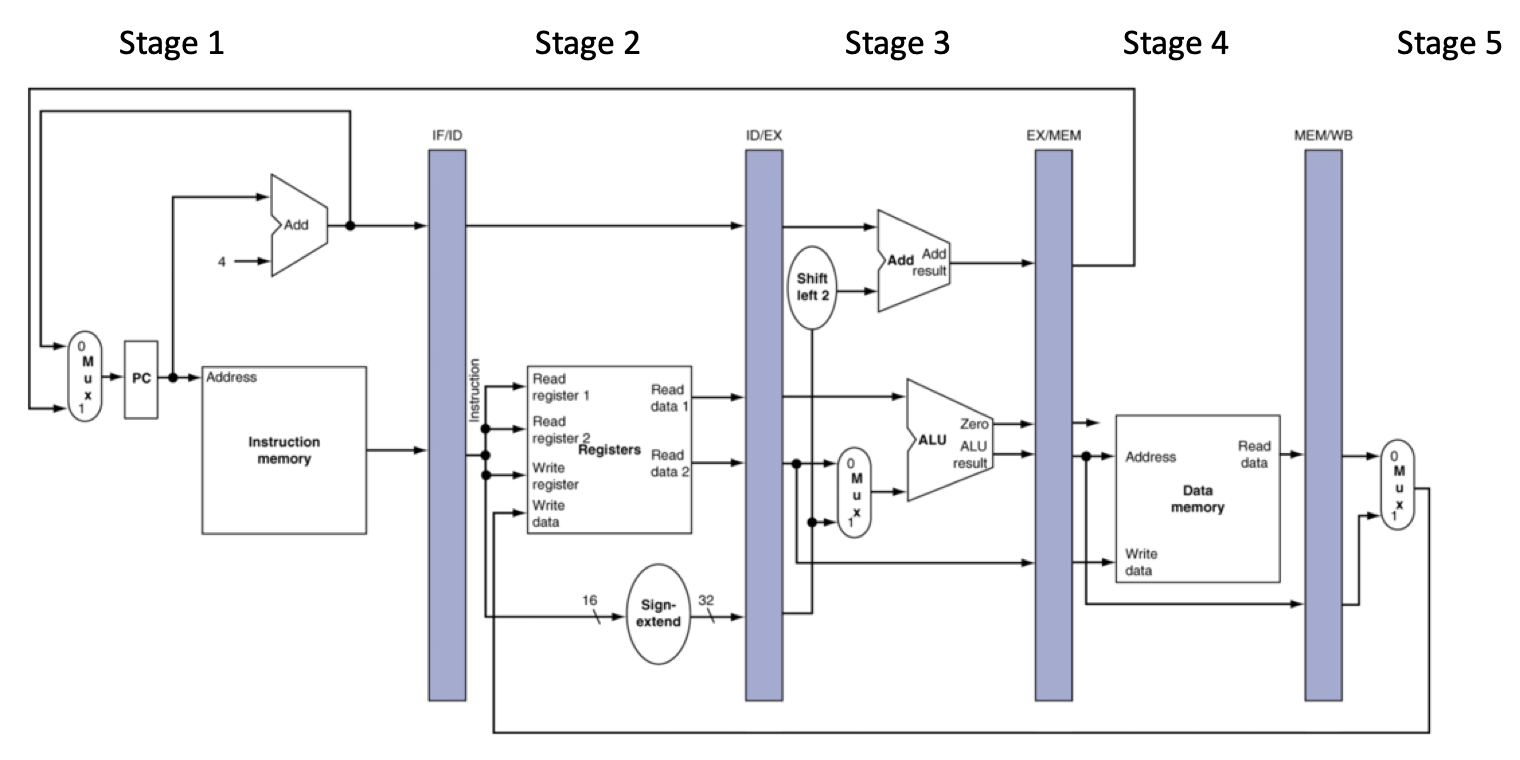
add.s $f8, $f14, $f12

ceil.w,s $f6, $f8

sub.s $f8, $f14, $f8

**3 functional units**. The ALU for the integer operations (ori, xor, add), a specialized unit for data movement and type conversions (mtc1, cvt1.s.w, ceil.w.s), and one for floating point operations (add.s and sub.s).

4. (5) What is the earliest pipeline stage in which a branch mis-prediction can be detected on the MIPS pipelined data path shown below? A branch misprediction is one that turns out to be incorrect.



It would be in **stage 3** as this is where the branch is actually executed, meaning this is the time the computer would realize that it is incorrect or correct.5. (15) Recall that with a true data dependency an instruction writes a register that is required as an input by a following instruction. An anti-dependency occurs if an instruction overwrites a register that is required as an input by an earlier instruction. If two or more instructions write a result to the same register, the register writes must be performed in the correct order; this is called an output dependency. One of the three terms (WAR, WAW or RAW) can be used as an alternate name for each of these three types of dependencies.

Consider the following instruction sequence containing instructions that can be executed out of order on a superscalar system.

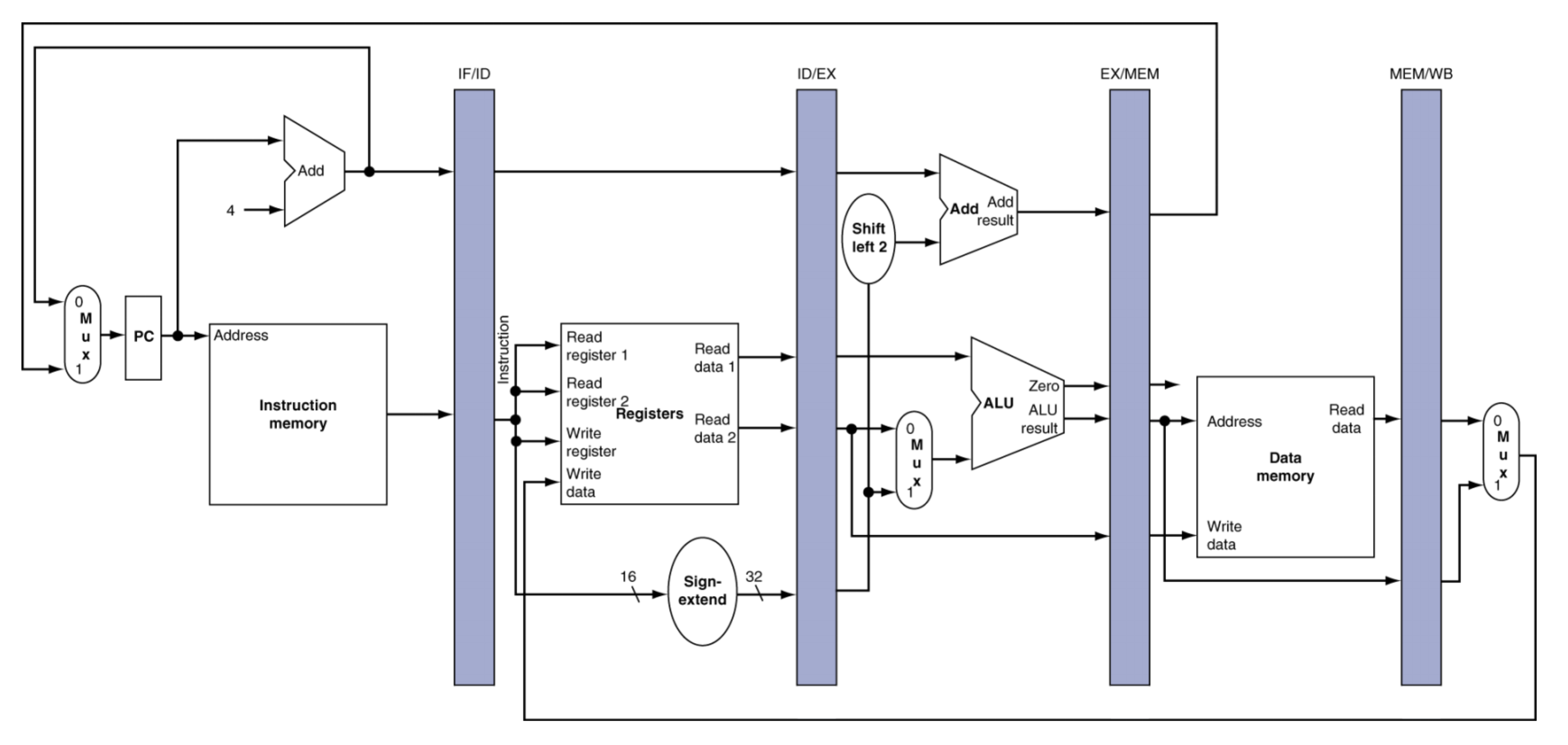
1. lui $9, 0x4000
2. lw $4, 8($9)
3. add $5, $0, $0
4. addiu $4, $5, 4
5. sw $5, 8($9)
6. sll $5, $5, 2
7. addiu $9, $9, 8

Complete the table below by filling in the blank entries to identify the register involved in the dependency, the type of data dependency (WAR, WAW or RAW), the two instructions involved in the dependency and whether register renaming can eliminate the dependency. If register renaming can eliminate the dependency, show the instruction that uses the renamed register with the renamed register substituted.

Use the next higher unused register name for the next register that is renamed. For example: $6, $7, $8, $10, etc. can be used to rename registers where needed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Register | (WAR, WAW or RAW) | First instruction | Second instruction | Reg. renaming works? (yes or no) | Instruction with renamed register |
| $9 | RAW | Lui $9 0x4000 | Lw $4, 8(49) | Yes | Lw $4, $8(10) |
| $9 | WAW | Lui $9 0x4000 | Addiu $9, $9, 8 | yes | Addiu $10, $10, 8 |
| $5 | RAW | Add $5, $0, $0 | Addiu $4, $5, 4 | yes | Addiu $4, $6, 4 |
| $4 | WAW | Lw $4, $8($9) | Addiu $4, $5, 4 | yes | Addiu $6, $5, 4 |
| $5 | WAW | Add $5, $0, $0 | Sll $5, $5, 2 | Yes | Sll $6, $6, 2 |
| $5 | WAR | Addiu $4, $5, 4 | Sw $5. 8($9) | yes | Sw $6. 8($9) |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

6. Base your answers to parts a) and b) below on the following version of the MIPS pipeline:



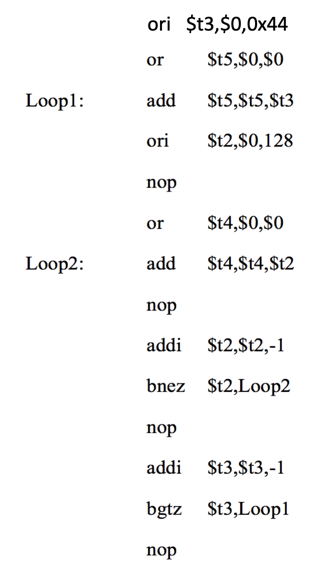
a) (5) Suppose the system employs delayed branching and uses a BHT to predict the behavior of conditional branch instructions. How many instructions must be flushed from the pipeline for each incorrect branch prediction?

A system that employs delayed branching and uses a BHT to predict the behavior of conditional branch instructions can cause either 1 or 3 instructions to be flushed. In a perfect world with a perfect processor, the instruction slots added after the branch instruction would be instructions that are branch independent. This means that when the BHT predicts the branch incorrectly only the branch instruction would need to be flushed as the following instructions are already working in the correct way. However, in some circumstances this might not be the case. The instructions following the branch could be branch dependent, meaning that all three would need to be flushed.

b) (5) Suppose the system does not employ delayed branching and uses a DHT instead of a BHT to predict the behavior of conditional branch instructions. How many instructions must be flushed from the pipeline for each incorrect branch prediction in this case?

In this system there would always need to be three instructions flushed due to an incorrect branch prediction. This is because the compiler simply adds the instructions that will follow what it believes the branch will be. Due to this the instructions following a branch instruction would be unusable to the processor as other instructions would most likely follow the new branch path.

7. Consider the following code containing nested loops:



Loop2 is nested within Loop1.

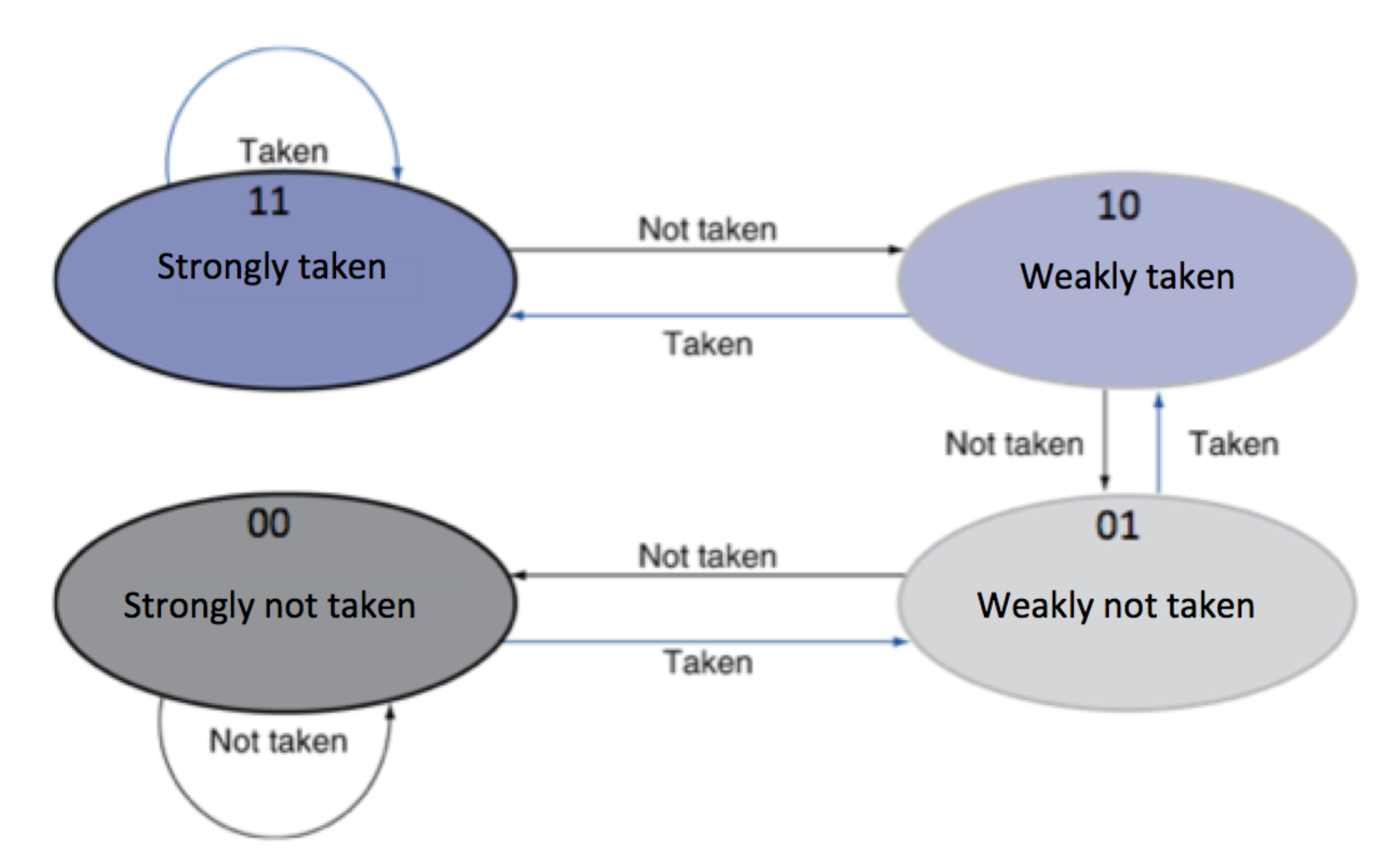
Branch prediction is used for both conditional branch instructions (the bnez and the bgtz) and both instructions employ delayed branching.

a) As described in module 8, a separate single-bit can be used to predict the behavior of each branch instruction (0 means predicted not taken, 1 means predicted taken). The initial value for each prediction bit is 0 (predicted not taken). What is the total number of mis-predictions for each branch instruction when the code shown above is executed on our 5-stage scalar pipeline?

(5) Number of mispredictions for the bnez = \_127\_

(5) Number of mispredictions for the bgtz = \_\_2\_\_

b) Suppose that instead of using a single branch prediction bit, the system uses a separate pair of branch prediction bits for each conditional branch instruction. What is the total number of mispredictions for each of the two conditional branch instructions in this case? The initial value of the branch prediction bits for each branch instruction = 00 (strongly not taken) and the bits are updated based on the diagram below:



(5) Number of mispredictions for the bnez = \_\_4\_

(5) Number of mispredictions for the bgtz = \_\_2\_\_

8. (10) On a certain VLIW system, each long instruction word that is generated is a packet or bundle of six individual machine operations. The hardware system contains three integer units and three floating point units. Therefore, up to 3 integer operations and 3 floating point operations can be performed in parallel in the same clock cycle. However, due to the mix of available instructions and possible dependencies, the system may not be able to make use of all six slots available in every instruction bundle. Unused slots within each six-instruction bundle should be filled with nop instructions.

Each row in the table shown below corresponds to a separate long instruction word.

Complete the table to show the minimum number of long instruction words or bundles that should be generated for the following group of instructions. Include as many additional rows in the table as you need.

add $11, $2, $3

add $4, $5, $11

mtc1 $4, $f12

cvt.s.w $f12, $f12

sub.s $f14, $f16, $f18

add.s $f8, $f14, $f12

ceil.w,s $f6, $f8

sub.s $f8, $f14, $f8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Int unit1 | Int unit2 | Int unit3 | Flt unit1 | Flt unit2 | Flt unit3 |
| add $11, $2, $3 | Mtc1 $4, $f12 | nop | Sub.s $f14, $f16, $f18 | Ceil.w.s $f6, $f8 | Nop |
| Add $4, $5, $11 | nop | Nop | cvt.s.w $f12, $f12 | Nop | nop |
| nop | nop | nop | Add.s $f8, $f14, $f12 | Sub.s $f8, $f14, $f8 | nop |

9. a) (10) The lui instruction in the following instruction sequence is fetched in cycle 1 on our pipeline that includes both a data hazard unit and a forwarding unit. Taking the looping into account, indicate the total number of clock cycles consumed by this code running on the pipeline. The bltz instruction is a delayed branch which branches to the instruction with label “loop” if register $12 contains a value that is less than 0. Explain your answer.

lui $4,0x400C

ori $4, $4, 0x2000

addiu $12, $0, -392

addu $8, $4, $12

nop

loop:

lw $14, 0($8)

lw $16, 400($8)

addu $14, $14, $16

sw $14, 800($8)

addiu $12, $12, 4

bltz $12, loop

addiu $12, $12, 4

xor $12, $12, $12

nop

To start lets consider the first 5 instructions before the loop. Lui will enter the pipeline at cycle one and nop will enter the pipeline at cycle five.

After this we can look at the loop itself. We know $12 contains the value -392 which 4 is added every time the loop runs. So there are 98 loops taken before we exit the loop. Due to this we can determine that each loop should take about 6 cycles since we have the delayed branch (ie the lw instructions are preloaded before the branch hits the evaluation stage.

Finally we can look at the last 3 instructions. Unfortunately due to the delayed branching we have to flush the pipeline and start over from the addiu instruction. This means that the final three instructions will account for 7 cycles for the nop to finish.

In total we have 5 + (98\*6) + 7 = **600 cycles.**

b) (5) What is the total number of clock cycles consumed by this same code running on the pipeline if the bltz instruction is not a delayed branch instruction.?

In good news, we can treat the first 5 instructions just like we treated them in the last problem, so 5 cycles to get to the loop.

The loop is where we get into tricky water, we are still going to execute the loop 98 times. Since we are no longer pushing lw into the pipeline before we compute bltz we have 2 extra cycles to account for each loop.

We will still be able to treat the final instructions the same as well since the pipeline needs to wait until the branch is executed to know what instructions it’ll use. This means that it’ll take 5 cycles

So for this problem we have 5 + 6 (first run through loop) + ((2 + 6) \* 97) + 5 = **792 cycles**.