AS. Label	Time Tag	Mnemonic	
A1	1	lui	r3,0x8002
…			
A3	3	lw	r8,-29(r28)
A4	4	addiu	r5,r8,16
A5	5	lw	r3,-26(r28)
…			
A7	7	addiu	r5,r1,32
A8	8	xor	r8,r3,r5
A9	9	sw	r8,-32(r28)

(a) Code fragment

Cycle	Execute	Forward	Snarf
-1		Ax(r28),Ay(r1)	A3(r28),A5(r28),A7(r1),A9(r28)
+0	A1,A3,A5,A7		
+1		A1(r3),A7(r5)	A8(r3,r5)
+2	A8	A3(r8),A5(r3)	A4(r8),A8(r3),A9(r8)
+3	A4,A8,A9		
+4		A4(r5),A8(r8)	A9(r8)
+5	A9		

(b) Execution schedule

[Before I present this example, I need to somehow introduce of the notion of an active station or issue slot that holds each instruction and handles the execution, forwarding and snooping]

Figure 2 shows an example of how time tags are used to enforce correct execution of instructions by enforcing data dependencies. Part (a) shows a fragment of a MIPS code. For simplicity, in this example we assume that there are no branches between instructions and at the end all of them will commit. Each instruction is assigned to an active station with a unique time tag.

Table 2.(a) illustrates the steps required to execute this code fragment. For simplicity, we assume that each instruction can be executed independent of the other one and there is no limit on the transactions that can be forwarded at each cycle.

In this example, we also assume that the committed value of r1 and r28 registers has already been forwarded. Assuming that load and store takes 2 cycles

As is shown in the first row of the table, in clock 0 instructions at A1, A3, A5 and A7 execute in parallel. The execution is a result of the snarfing of r1 and r28 registers in the previous cycle. Assuming two cycles for the execution of load and store instructions and one cycle for the rest of the instructions in this example, instructions at A1 and A7 will have their results ready in the next clock. The new value for register r3 and r5 is forwarded in clock 1 and is snarfed by A8. In the next clock, instruction at A8 will execute using its newly snarfed register value. Normally A8 will forward the new result, but since A5 is sending out a new value for r3, the A8 unit snarfes the new value for r3. This results in the re-execution of instruction at A8 which happens at clock +3. In the next cycle, A4 sends a new value for register r5 with a time tag of 4. But since the last value of r5 received by A8 had a time tag of 7 which is greater than 4, the new value is ignored by A8 and do not result in another re-execution.