1. RISC pipeline: Reduced instruction set computer (RISC)
, An ability to use an efficient instruction pipeline
/ ability to execute instructions at the rate of
one per clock cycle.

say a three segment instruction pipeline has following phases:

I: Instruction
A: ALU operation
E: Execute Instruction.

Now, say the operation of following four instructions:

1. LOAD: $R1 \leftarrow M[address\ 1]$
2. LOAD: $R2 \leftarrow M[address\ 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[address\ 3] \leftarrow R3$

Clock	1	2	3	4	5	6
1. LOAD R1	I	A	E			
2. LOAD R2		I	A	E		
3. ADD R_3			I	A	E	
4. STORE				I	A	E

clock	1	2	3	4	5	6	7
1. LOAD R1	I	A	E				
2. LOAD R2		I	A	E			
3. No-operation			I	A	E		
4. ADD R3				I	A	E	
5. STORE					I	A	E

So here actually after LOAD R2 it is wasting a clock cycle. That is why it is called as pipeline timing with delayed load.

- Delayed Branch: say the following five/six instructions:

Load from memory to R1
increment R2
Add R3 to R4
Subtract R5 from R6
Branch to address X.
Next instruction in X.

On this delayed branch system no-operation instructions are being fetched from the memory and they are executed through the pipeline when the branch instruction is executed. It is upto the compiler to find useful instructions to put after the branch instruction. Failing that the compiler can insert no-op instructions.

Clock	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Next instruction in X.								I	A	E

2. LOAD $R_1 \leftarrow M[312]$
Add $R_2 \leftarrow R_2 + M[313]$
Increment $R_3 \leftarrow R_3 + 1$
STORE $M[314] \leftarrow R_3$

	1	2	3	4
	FI	DA	PO	EX.
		FI	DA	PO
			FI	DA
				FI

so seg EX: transfer memory word to R1
F0: Read M[313]
DA: Decode (increment) instruction
FI: Fetch the instruction from memory.

8. RISC → I LOAD R1 ← Memory[313]
A INCREMENT R1 ← R1+1
E

	Stage 1	2	3	4
instr 1.	I	A	E	
instr 2		I	A	E

Data hazards.

Before the completion of loading into R1 it is not possible to increment.

4. 4 floating-point pipeline processor.
each processor uses a cycle time of 40 ns.
total 400 floating-point operations are there.
so 400 operations will be divided into each of four processors
so processing time: $\frac{400}{4} \times 40 = 4000$ ns.

using a single pipeline, # cycle time is given 10 ns.
So processing time $400 \times 10 = 4000$ ns
so no change.

5. 250 billion floating-point operations so 250×10^9
100 megflop ⇒ 100 million floating point operation 100×10^6
so required time = $\frac{250 \times 10^9}{100 \times 10^6}$ sec.
= 2500 sec = 41.67 minutes.