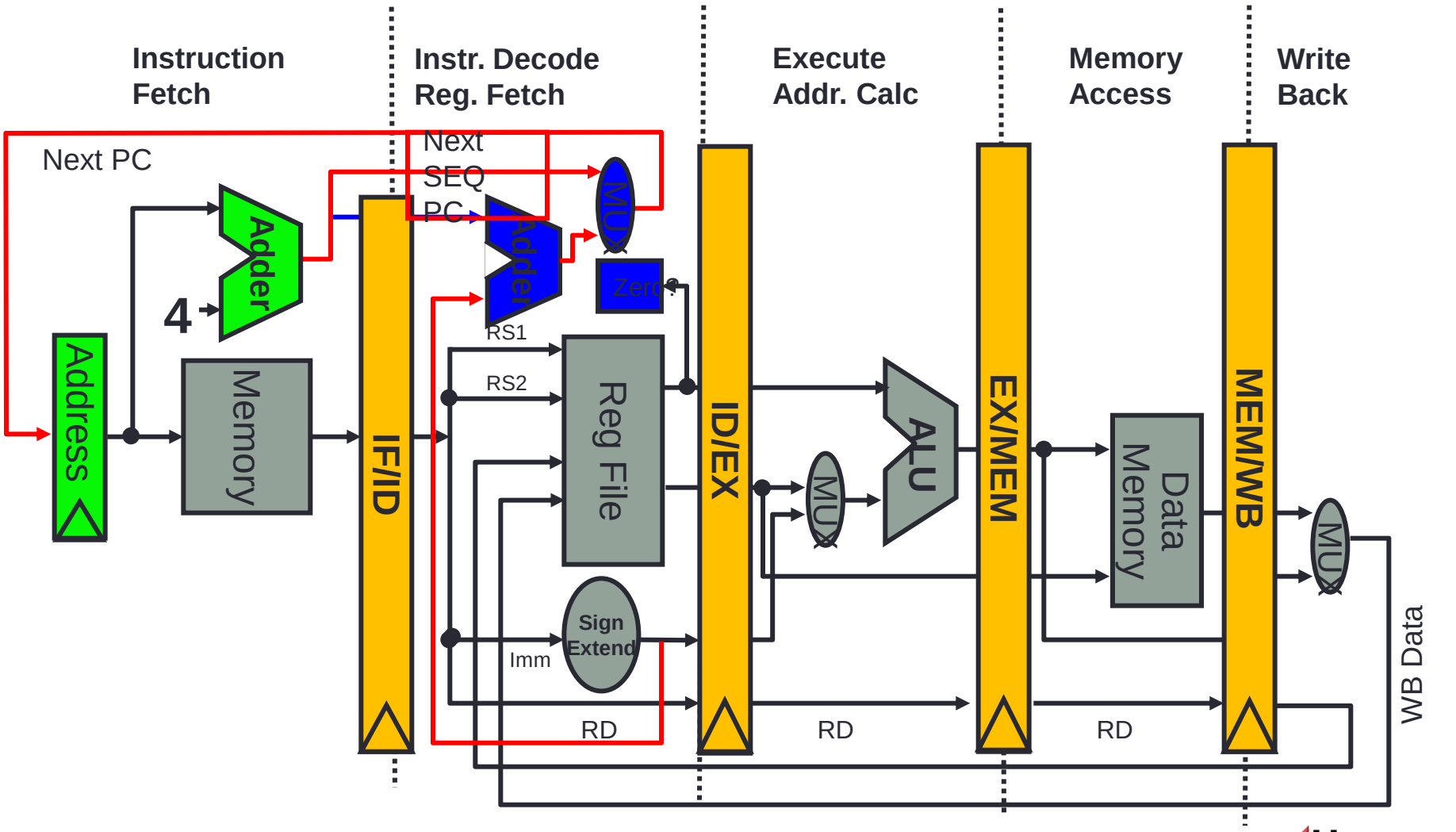
ADVANCED ILP

Dr Noor Mahammad Sk

RISC 5 Stage Processing (ILP)



ALU

- Integer Adder
- Integer Multiplier
- Floating Point Adder
- Floating Point Multiplier
- Logic Unit
- Shit Unit

ALU

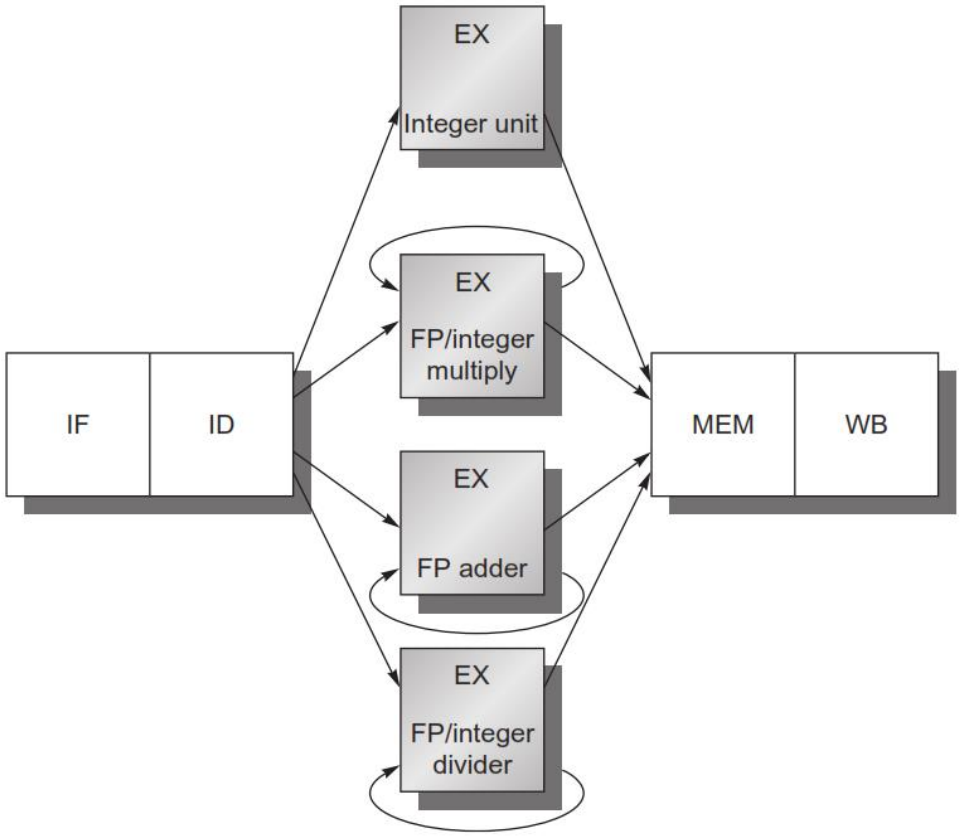
- Integer Adder – CLA – 4 CC
- Integer Multiplier – WTM – 8 CC
- Floating Point Adder – 20 CC
- Floating Point Multiplier – 40 CC
- Logic Unit – 1 CC
- Shit Unit – 4 CC

RISC V FP Operation

- It is impractical to require that all RISC V FP operations complete in 1 clock cycle, or even in 2.
- Doing so would mean accepting a slow clock or using enormous amounts of logic in the FP units, or both.
- Instead, the FP pipeline will allow for a longer latency for operations.
- This is easier to grasp if we imagine the FP instructions as having the same pipeline as the integer instructions, with two important changes.
- First, the EX cycle may be repeated as many times as needed to complete the operation—the number of repetitions can vary for different operations.
- Second, there may be multiple FP functional units.
- A stall will occur if the instruction to be issued will cause either a structural hazard for the functional unit it uses or a data hazard.

- Let's assume that there are four separate functional units in our RISC V implementation:
- 1. The main integer unit that handles loads and stores, integer ALU operations, and branches
- 2. FP and integer multiplier
- 3. FP adder that handles FP add, subtract, and conversion
- 4. FP and integer divider

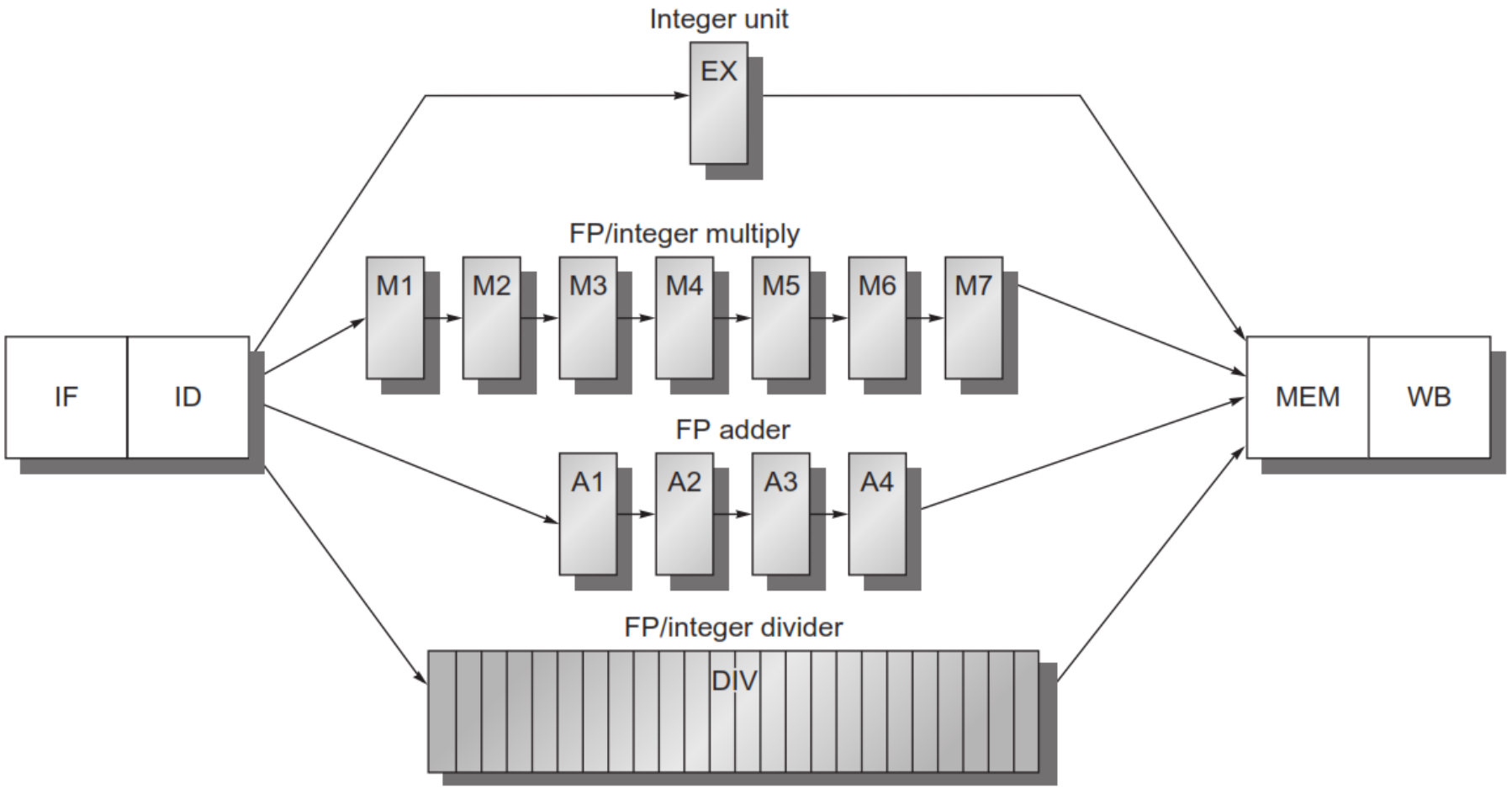
- The RISC V pipeline with three additional unpipelined, floating-point, functional units.
- Because only one instruction issues on every clock cycle, all instructions go through the standard pipeline for integer operations.
- The FP operations simply loop when they reach the EX stage.
- After they have finished the EX stage, they proceed to MEM and WB to complete execution.



Latencies and initiation intervals for functional units

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

A pipeline that supports multiple outstanding FP operations.



- The FP multiplier and adder are fully pipelined and have a depth of seven and four stages, respectively.
- The FP divider is not pipelined, but requires 24 clock cycles to complete.
- The latency in instructions between the issue of an FP operation and the use of the result of that operation without incurring a RAW stall is determined by the number of cycles spent in the execution stages.
- For example, the fourth instruction after an FP add can use the result of the FP add.
- For integer ALU operations, the depth of the execution pipeline is always one and the next instruction can use the results.

The pipeline timing of a set of independent FP operations.

`fmul.d`	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
`fadd.d`		IF	ID	*A1*	A2	A3	**A4**	**MEM**	WB		
`fadd.d`			IF	ID	*EX*	**MEM**	WB				
`fsd`				IF	ID	*EX*	**MEM**	WB			

A typical FP code sequence showing the stalls arising from RAW hazards

	Clock cycle number																
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
`fld f4,0(x2)`	IF	ID	EX	MEM	WB												
`fmul.d f0,f4,f6`		IF	ID	Stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
`fadd.d f2,f0,f8`			IF	Stall	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	MEM	WB
`fsd f2,0(x2)`					IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	EX	Stall	Stall	Stall	MEM

- The longer pipeline substantially raises the frequency of stalls versus the shallower integer pipeline.
- Each instruction in this sequence is dependent on the previous and proceeds as soon as data are available, which assumes the pipeline has full bypassing and forwarding.
- The fsd (store operation) must be stalled an extra cycle so that its MEM does not conflict with the fadd.d.
- Extra hardware could easily handle this case.

	Clock cycle number										
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
`fmul.d f0,f4,f6`	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
`fadd.d f2,f4,f6`				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
`fld f2,0(x2)`							IF	ID	EX	MEM	WB

- Three instructions want to perform a write-back to the FP register file simultaneously, as shown in clock cycle 11.
- This is not the worst case, because an earlier divide in the FP unit could also finish on the same clock.
- Note that although the fmul.d, fadd.d, and fld are in the MEM stage in clock cycle 10, only the fld actually uses the memory, so no structural hazard exists for MEM.

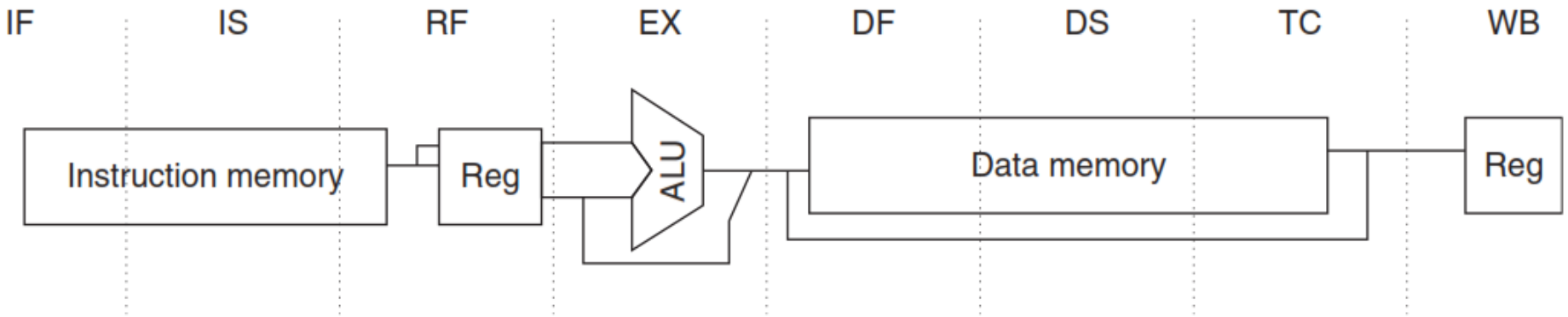
The MIPS R4000 Pipeline

- The R4000 implements MIPS64 but uses a deeper pipeline than that of our five-stage design both for integer and FP programs.
- This deeper pipeline allows it to achieve higher clock rates by decomposing the five-stage integer pipeline into eight stages.

Eight Stages

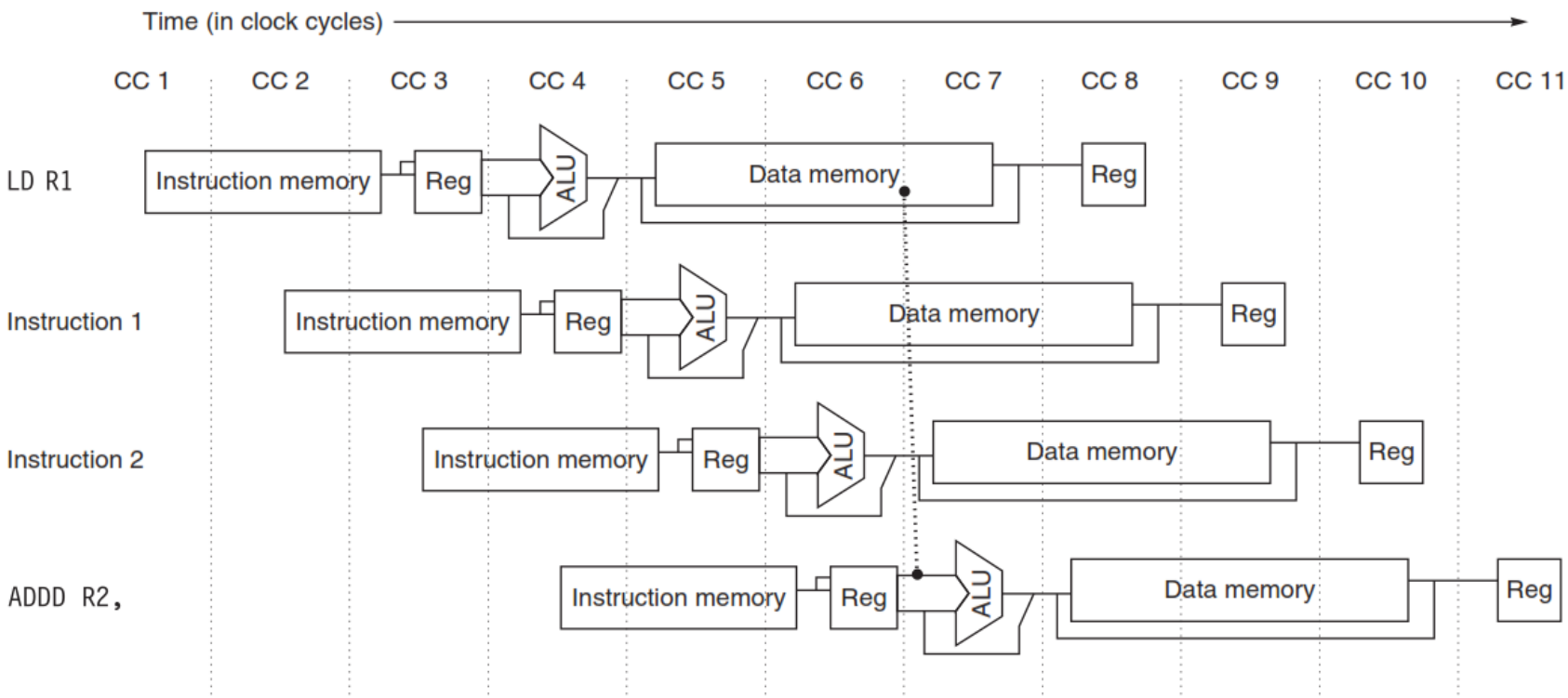
- IF—First half of instruction fetch; PC selection actually happens here, together with initiation of instruction cache access.
- IS—Second half of instruction fetch, complete instruction cache access.
- RF—Instruction decode and register fetch, hazard checking, and instruction cache hit detection.
- EX—Execution, which includes effective address calculation, ALU operation, and branch-target computation and condition evaluation.
- DF—Data fetch, first half of data cache access.
- DS—Second half of data fetch, completion of data cache access.
- TC—Tag check, to determine whether the data cache access hit.
- WB—Write-back for loads and register-register operations.

The eight-stage pipeline structure of the R4000 uses pipelined instruction and data caches.



- The instruction is actually available at the end of IS, but the tag check is done in RF, while the registers are fetched.
- Thus, we show the instruction memory as operating through RF.
- The TC stage is needed for data memory access, because we cannot write the data into the register until we know whether the cache access was a hit or not.

The structure of the R4000 integer pipeline leads to a x 1 load delay



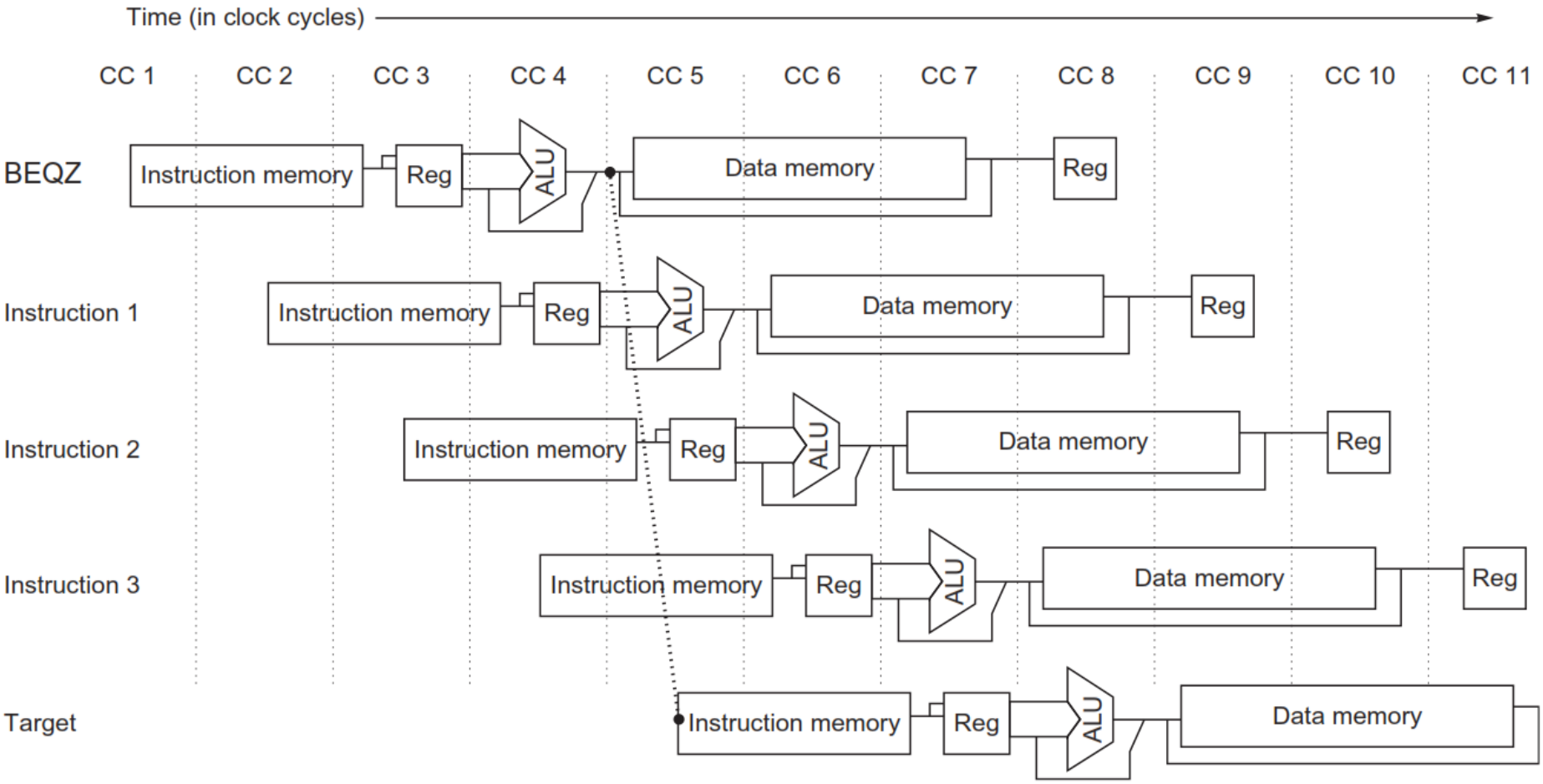
- A x1 delay is possible because the data value is available at the end of DS and can be bypassed.
- If the tag check in TC indicates a miss, the pipeline is backed up a cycle, when the correct data are available.

A load instruction followed by an immediate use results in a x1 stall.

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
`ld x1,...`	IF	IS	RF	EX	DF	DS	TC	WB	
`add x2,x1,...`		IF	IS	RF	Stall	Stall	EX	DF	DS
`sub x3,x1,...`			IF	IS	Stall	Stall	RF	EX	DF
`or x4,x1,...`				IF	Stall	Stall	IS	RF	EX

- Normal forwarding paths can be used after two cycles, so the add and sub get the value by forwarding after the stall.
- The or instruction gets the value from the register file.
- Because the two instructions after the load could be independent and hence not stall, the bypass can be to instructions that are three or four cycles after the load.

The basic branch delay is three cycles, because the condition evaluation is performed during EX.



The Floating-Point Pipeline

- The R4000 floating-point unit consists of three functional units:
- A floating-point divider,
- A floating-point multiplier, and
- A floating-point adder.
- The adder logic is used on the final step of a multiply or divide.

The eight stages used in the R4000 floating-point pipelines

Stage	Functional unit	Description
A	FP adder	Mantissa add stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

FP instruction	Latency	Initiation interval	Pipe stages
Add, subtract	4	3	U, S+A, A+R, R+S
Multiply	8	4	U, E+M, M, M, M, N, N+A, R
Divide	36	35	U, A, R, D^{28}, D+A, D+R, D+A, D+R, A, R
Square root	112	111	U, E, $(A+R)^{108}$, A, R
Negate	2	1	U, S
Absolute value	2	1	U, S
FP compare	3	2	U, A, R

- The latencies and initiation intervals for the FP operations initiation intervals for the FP operations both depend on the FP unit stages that a given operation must use.
- The latency values assume that the destination instruction is an FP operation;
- The latencies are one cycle less when the destination is a store.
- The pipe stages are shown in the order in which they are used for any operation.
- The notation S+A indicates a clock cycle in which both the S and A stages are used.
- The notation D^{28} indicates that the D stage is used 28 times in a row.