

第三章 Instruction-Level Parallelism and Its Dynamic Exploitation

陈文智

chenwz@zju.edu.cn

浙江大学计算机学院

2014年10月

3.3.5 Pipelining **Control** Hazards

- **Taxonomy of Hazards**

- **Structural hazards**

- These are conflicts over hardware resources.

- **Data hazards**

- Instruction depends on result of prior computation which is not ready (computed or stored) yet
- OK, we did these, Double Bump, Forwarding path, software scheduling, otherwise have to stall

- **Control hazards**

- branch condition and the branch PC are not available in time to fetch an instruction on the next clock

The Control hazard

一、Cause

- branch condition and the branch PC are not available in time to fetch an instruction on the next clock
- The next PC takes time to compute
- For conditional branches, the branch direction takes time to compute.
- Control hazards can cause a greater and greater performance loss for MIPS pipeline than do data hazards.

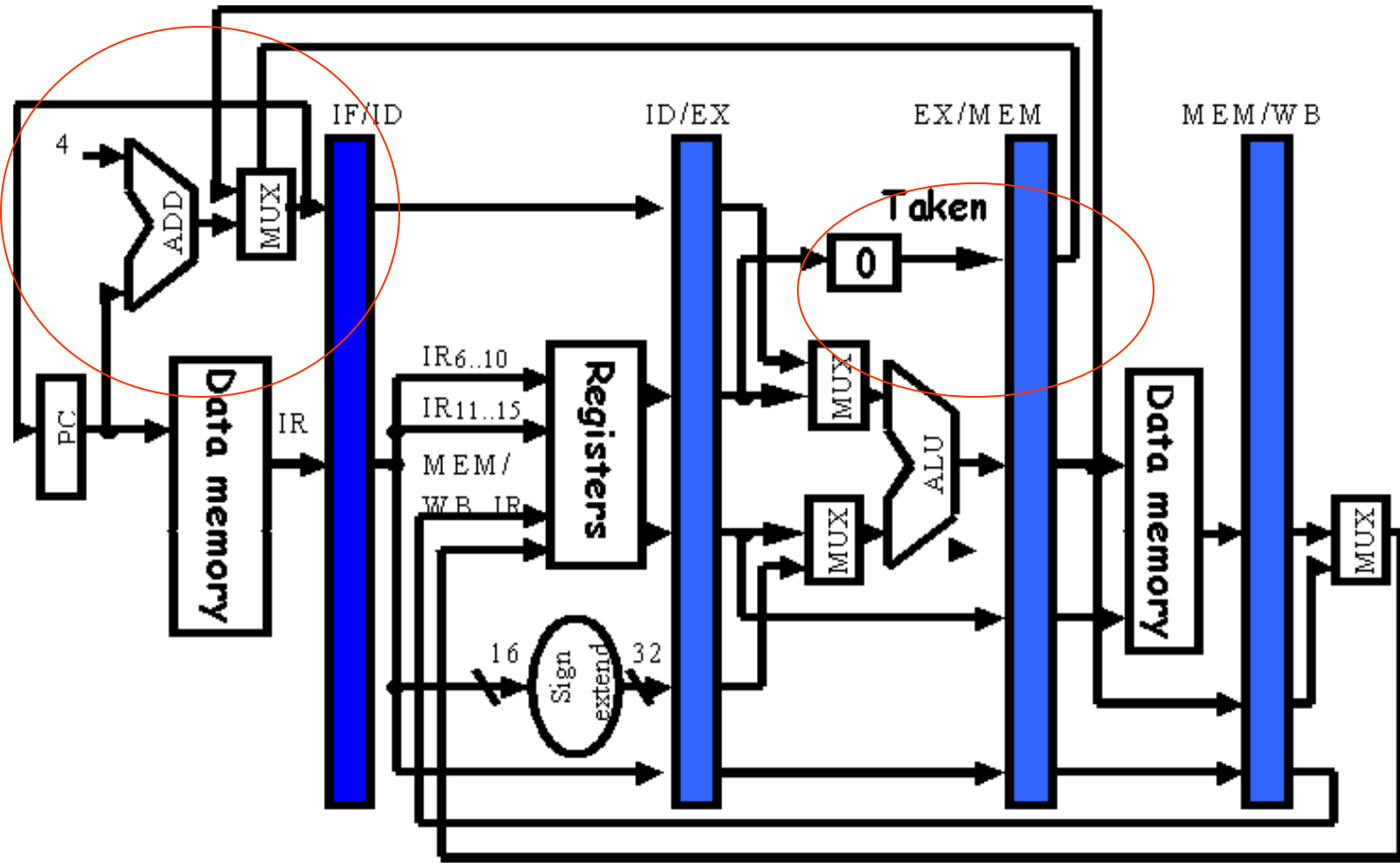
Example: Branches

Address	Instruction	
36	NOP	
40	ADD R30, R30, R30	
44	BEQ R1, R3, 24	<- this branches to address 72
48	AND R12, R2, R5	} We execute all these if R1 != R3
52	OR R13, R6, R2	
56	ADD R14, R2, R2	
60	...	
64	...	
68		} We execute just these if R1 == R3
72	LW R4, 50(R7)	
76	...	

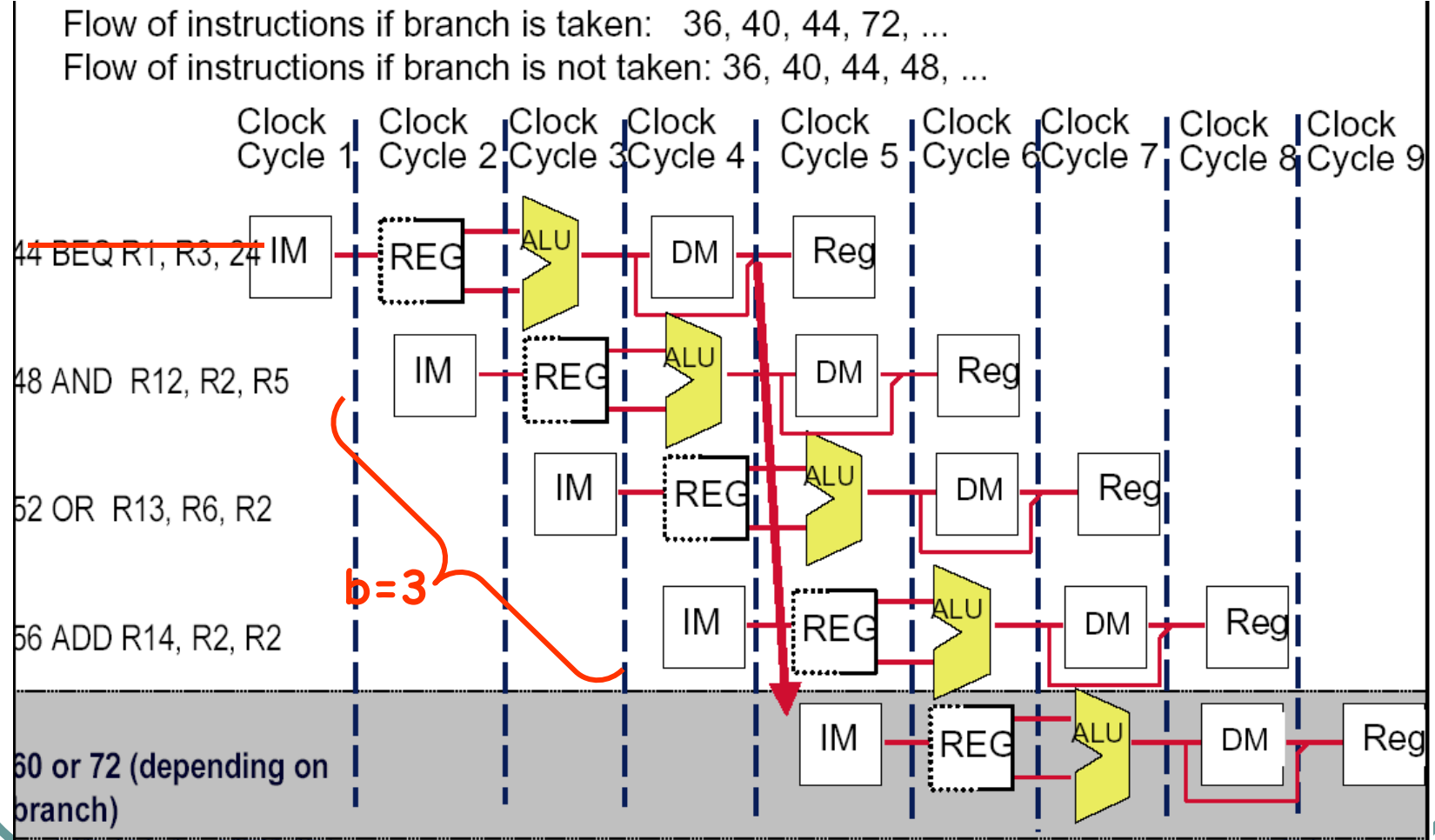
Flow of instructions if branch is taken: 36, 40, 44, 72, ...

Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

Branches of Basic Pipelined Datapath



二、The Penalty of Control hazard

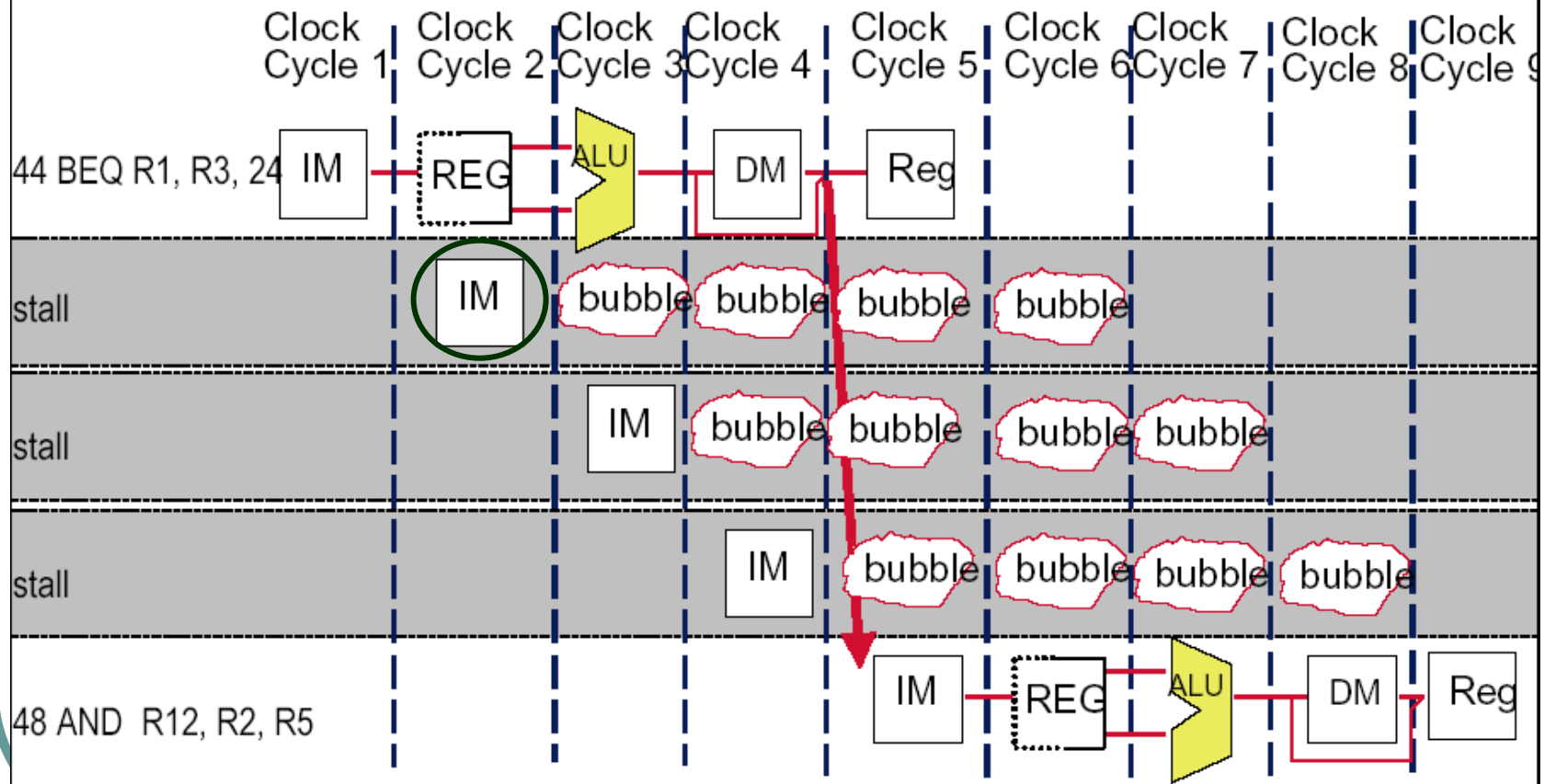


三、 Dealing with the control hazard

- Four simple solutions
 - Freeze or flush the pipeline
 - Penalty is fixed.
 - Can not be reduced by software.
 - Predict-not-taken (Predict-untaken)
 - Treat every branch as not taken
 - Predict-taken
 - Treat every branch as taken
 - Delayed branch
- Note:
 - Fixed hardware
 - Compile time scheme using knowledge of hardware scheme and of branch behavior

(1) Freeze or flush the pipeline

Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

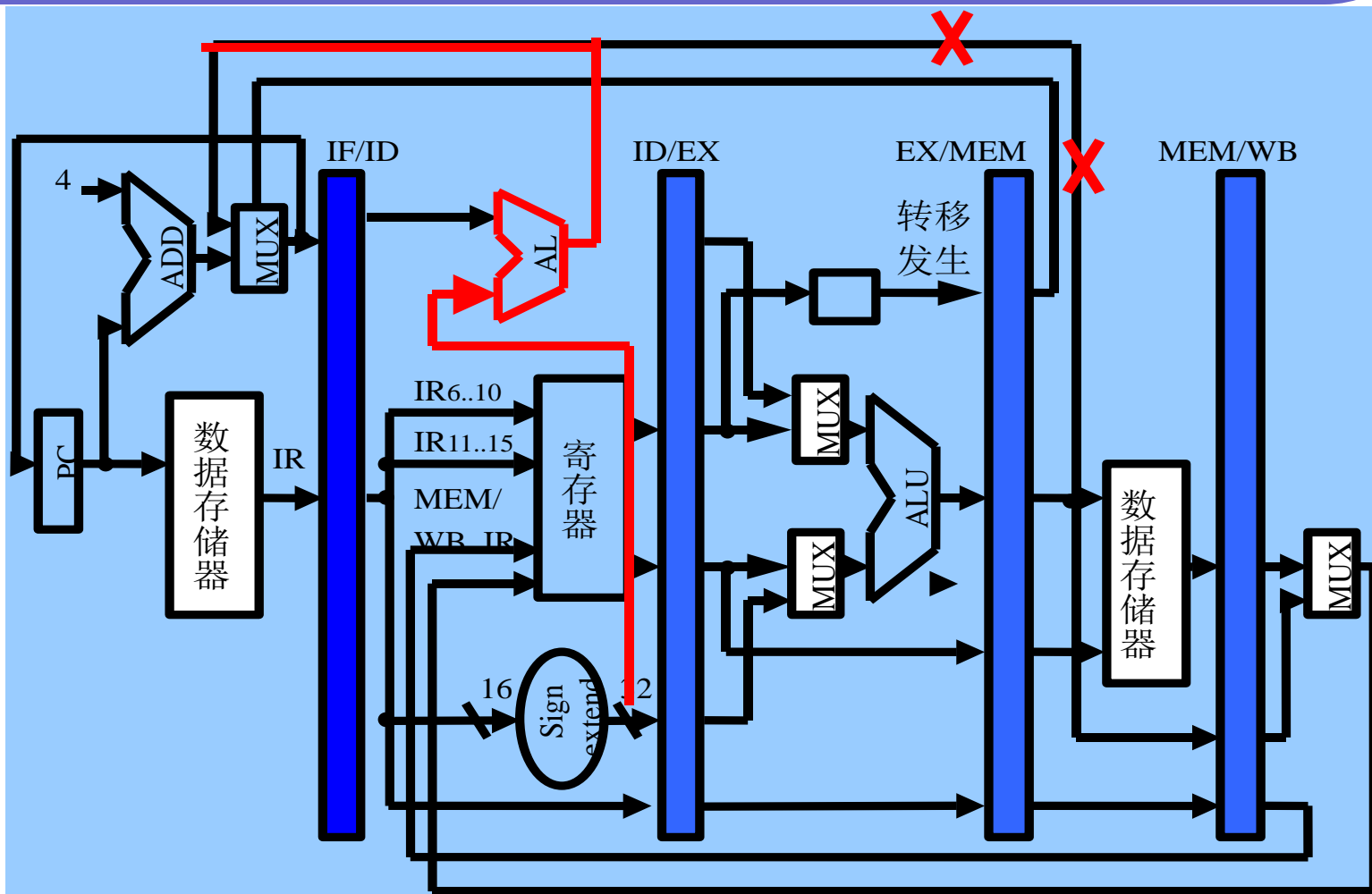


(2) Predict not-taken

- Hardware:
 - Treat every branch as not taken (or as the formal instruction)
 - When branch is not taken, the fetched instruction just continues to flow on. No stall at all.
 - If the branch is taken, then restart the fetch at the branch target, which cause 3 stall.(should turn the fetched instruction into a no-op)
- Compiler:
 - Can improve the performance by coding the most frequent case in the untaken path.

(3)Predict –taken

- Most branches(60%) are taken, so we should make the taken branch more faster. Why not try assuming the branch always taken?
- Hardware
 - Treat every branch as taken (evidence: more than 60% branches are taken)
 - As soon as the branch target address is computed, assume the branch to be taken and begin fetching and executing at the target.
 - Only useful when the target is known before the branch outcome.
 - No advantage at all for MIPS 5-stage pipeline.
- Compiler
 - Can improve the performance by coding the most frequent case in the taken path.



Pipeline status for predict-taken

Branch is **taken**: 1 stall

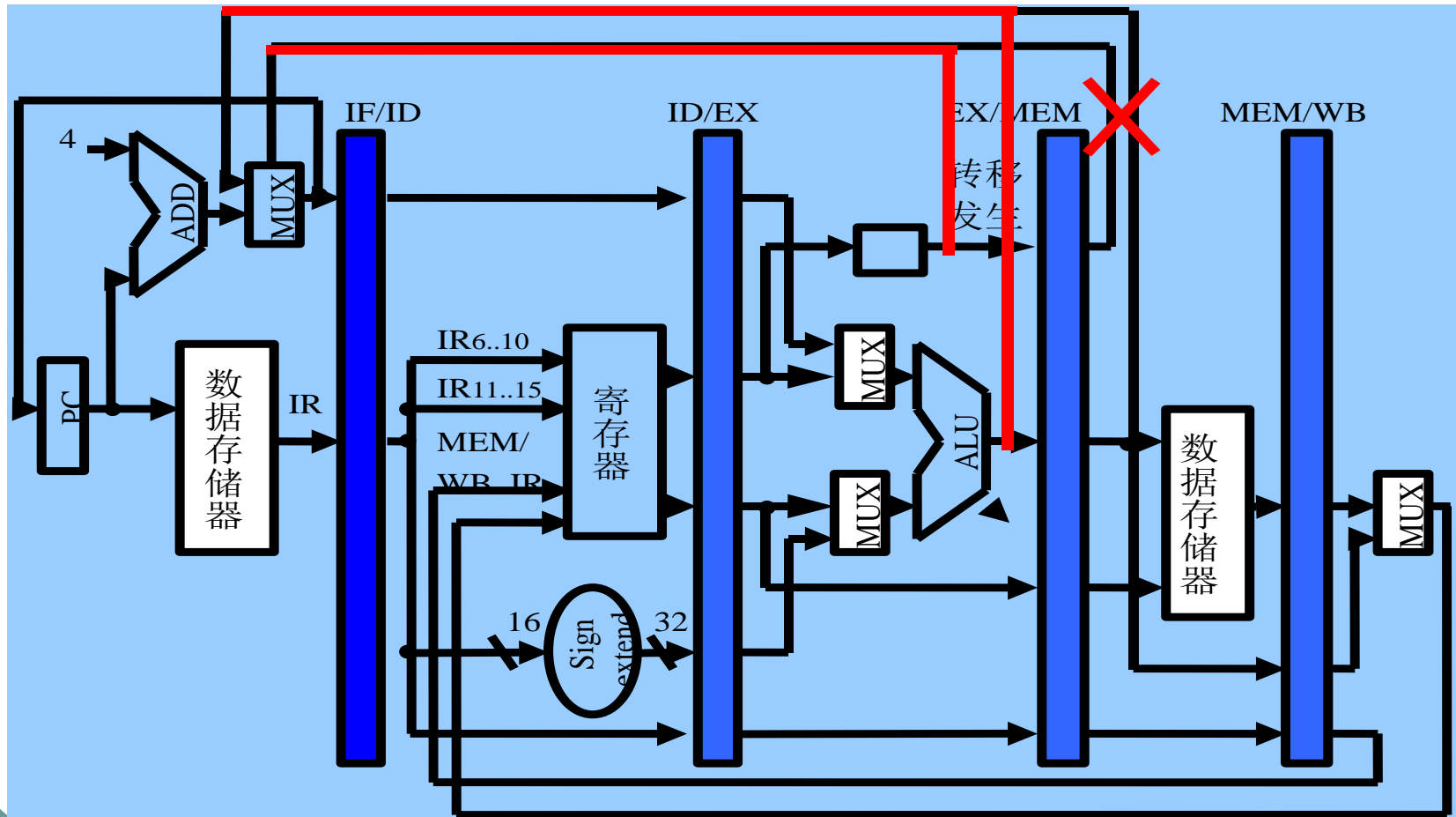
44 BEQ R1, R3, 24	IF	ID	EX	MEM	WB		
48 AND R12, R2, R5		IF	idle	idle	idle	idle	
72 LW R4, 50(R7)			IF	ID	EX	MEM	WB
76				IF	ID	EX	MEM
80					IF	ID	EX

Branch is **not taken**: 3 stall

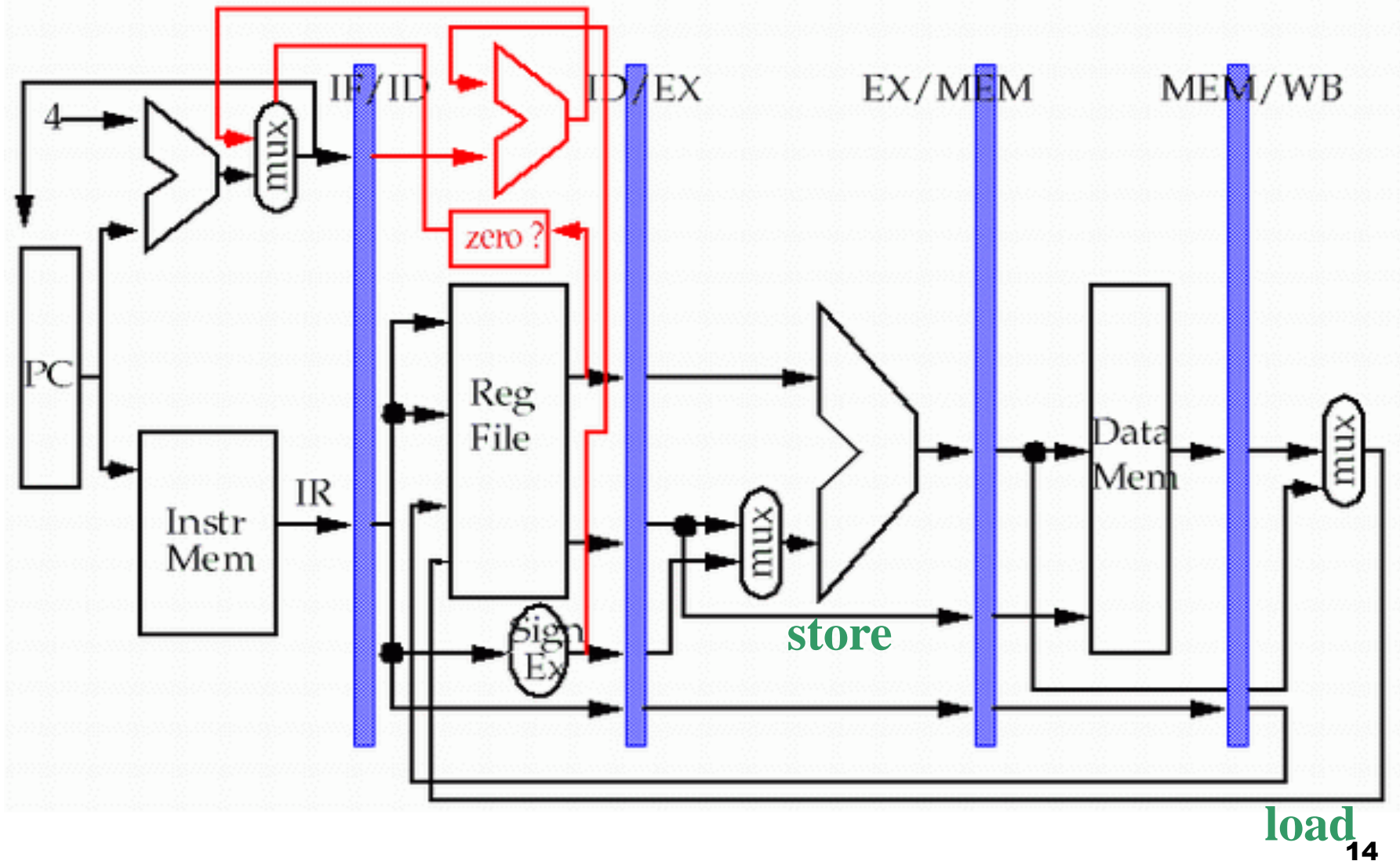
44 BEQ R1, R3, 24	IF	ID	EX	MEM	WB		
48 AND R12, R2, R5		IF	idle	idle	idle	idle	
72 LW R4, 50(R7)			IF	ID	idle	idle	idle
76				IF	idle	idle	idle
48 AND R12, R2, R5					IF	ID	EX

Modify MIPS Datapath

Move the Branch Computation Forward

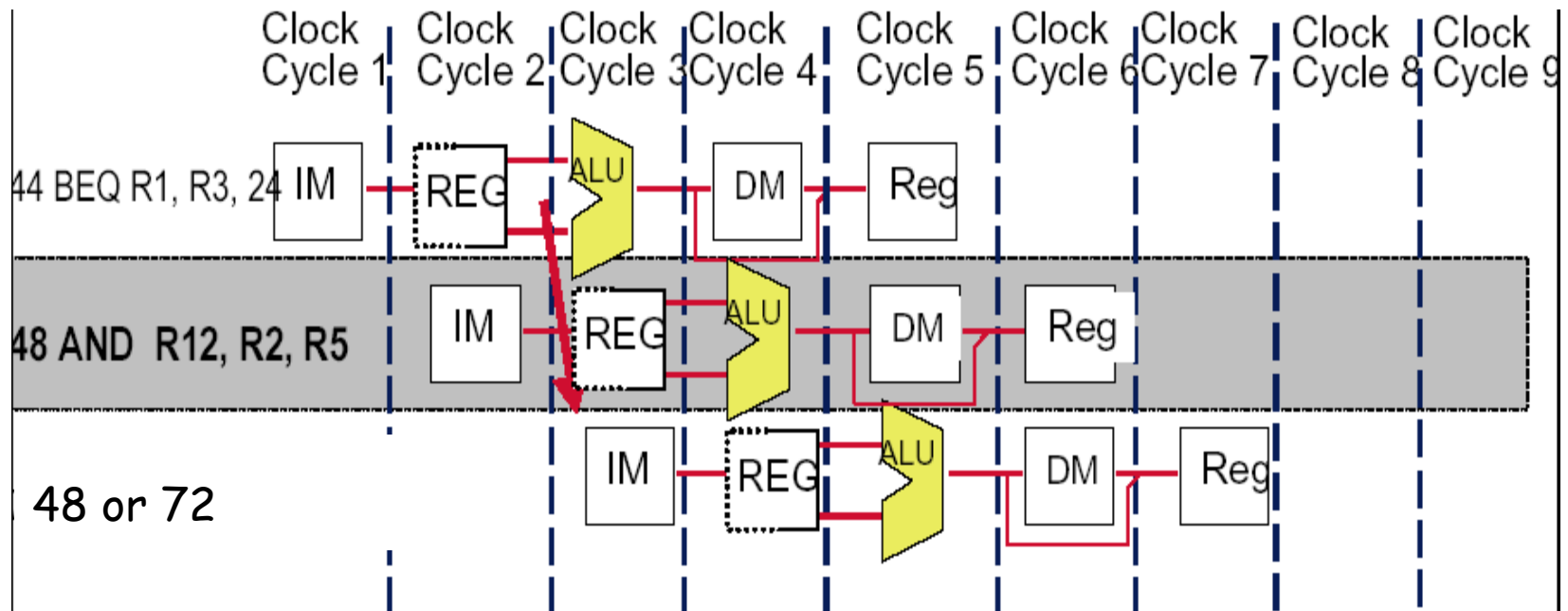


Move the Branch Computation more Forward



Result: New & Improved MIPS Datapath

- Need just **1** extra cycle after the BEQ branch to know right address
- On MIPS, its called - **the branch delay slot**




(4)Delayed branch

- Good information
 - Just 1 cycle to figure out what the right branch address is
 - So, not 2 or 3 cycles of potential NOP or stall
- Strange news
 - OK, it's **always** 1 cycle, and we **always** have to wait
 - And on MIPS, **this instruction always executes, no matter whether the branch taken or not taken. (hardware scheme)**

Branch delay slot

- Hence the name: **branch delay slot**

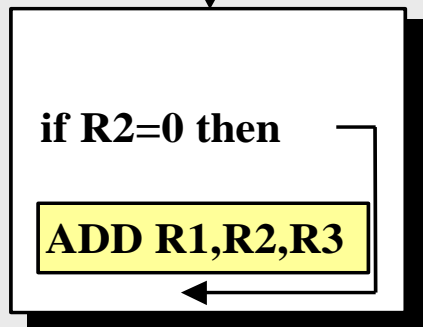
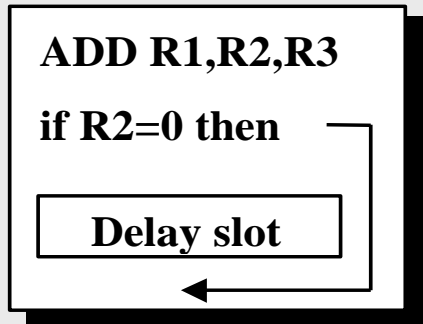
```
branch instruction
sequential successor1
sequential successor2
...
sequential successorn
branch target if taken
```



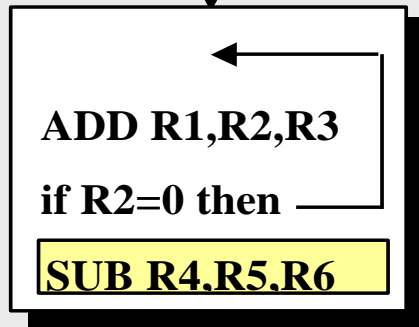
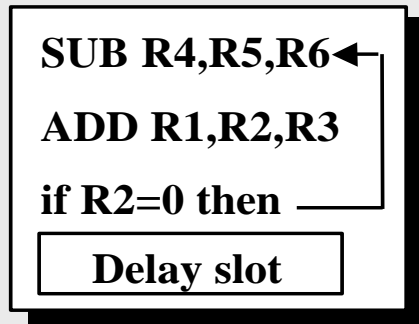
Branch delay slots

- The instruction cycle after the branch is used for address calculation , 1 cycle delay necessary
- SO...we regard this as a **free instruction cycle**, and we just DO IT
- Consequence
 - You (or your compiler) will need to **adjust your code** to put some useful work in that "slot", since just putting in a is wasteful

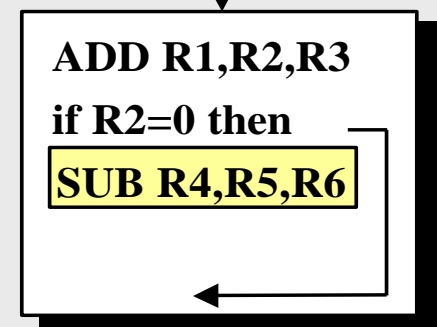
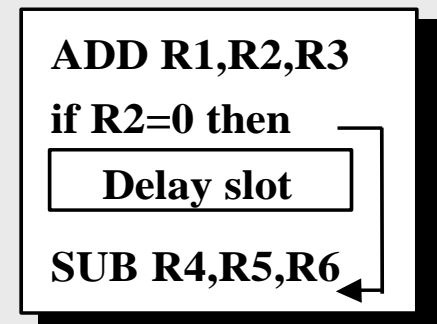
How to adjust the codes?



(a) From before



(b) From target



(c) From fall-through

Example: rewrite the code (a)

Without Branch Delay Slot

Address	Instruction
36	NOP
40	ADD R30, R30, R30
44	BEQ R1, 24
48	AND R12, R2, R5
52	OR R13, R6, R2
56	ADD R14, R2, R2
60	...
64	...
68	...
72	LW R4, 50(R7)
76	...


With Branch Delay Slot

Address	Instruction
36	NOP
40	BEQ R1, R3, 28
44	ADD R30, R30, R30
48	AND R12, R2, R5
52	OR R13, R6, R2
56	ADD R14, R2, R2
60	...
64	...
68	...
72	LW R4, 50(R7)
76	...

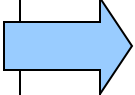
- Flow of instructions if branch is taken: 36, 40, 44, 72, ...
- Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

Example: rewrite the code (b-1)

Loop: LW R2, 0(R1)
ADD R3, R2, R4
SW R3, 0(R1)
.....
SUB R1, R1, #4
BNEZ R1, Loop




LW R2, 0(R1)
Loop: ADD R3, R2, R4
SW R3, 0(R1)
.....
SUB R1, R1, #4
BNEZ R1, Loop
LW R2, 0(R1)

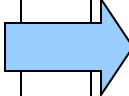


Example: rewrite the code (b-2)

```
Loop: LW   R2, 0(R1)
      ADD  R3, R2, R4
      SW   R3, 0(R1)
      DIV  .....
      .....
      SUB  R1, R1, #4
      BNEZ R1, Loop
```



```
Loop: LW   R2, 0(R1)
      ADD  R3, R2, R4
      DIV  .....
      .....
      SUB  R1, R1, #4
      BNEZ R1, Loop
      SW   R3, +4(R1)
```



Scheduling strategy vs. performance improvement

Scheduling strategy	Requirements	Improves performance when?
a. From before branch	Branch must not depend on the rescheduled instruction	Always
b. From target	Must be OK to execute rescheduled instruction if branch is not taken. May need to duplicate instructions.	When branch is taken. May enlarge program if instructions are duplicated.
c. From fall through	Must be OK to execute instruction if branch is taken.	When branch is not taken.
d. place a no-op		No improvement.

Constraints of the delayed branch

- There are restrictions on the instructions that are scheduled into the delay slots
- The compiler's ability to predict accurately whether or not a branch is taken determines how much useful work is actually done.
- For scheduling scheme b and c,
 - It must be O.K. to execute the SUB instruction if the prediction is wrong.
 - Or the hardware must provide a way of cancelling the instruction.

About delayed branch

- Delayed branch are adopted in most RISC processors.
- In general, the length of branch delay is more than 1 . However, always just one slot is used due to the compiler complexity.

Summary for control hazard

- Control hazards can cause a greater performance loss than do data hazards.
- In general, the deeper the pipeline, the worse the branch penalty in clock cycles.
- A higher CPI processor can afford to have more expensive branches.
- The efficiency of the three schemes greatly depends on the branch prediction.

3.4 Extending the MIPS Pipeline to Handle Multicycle Operations

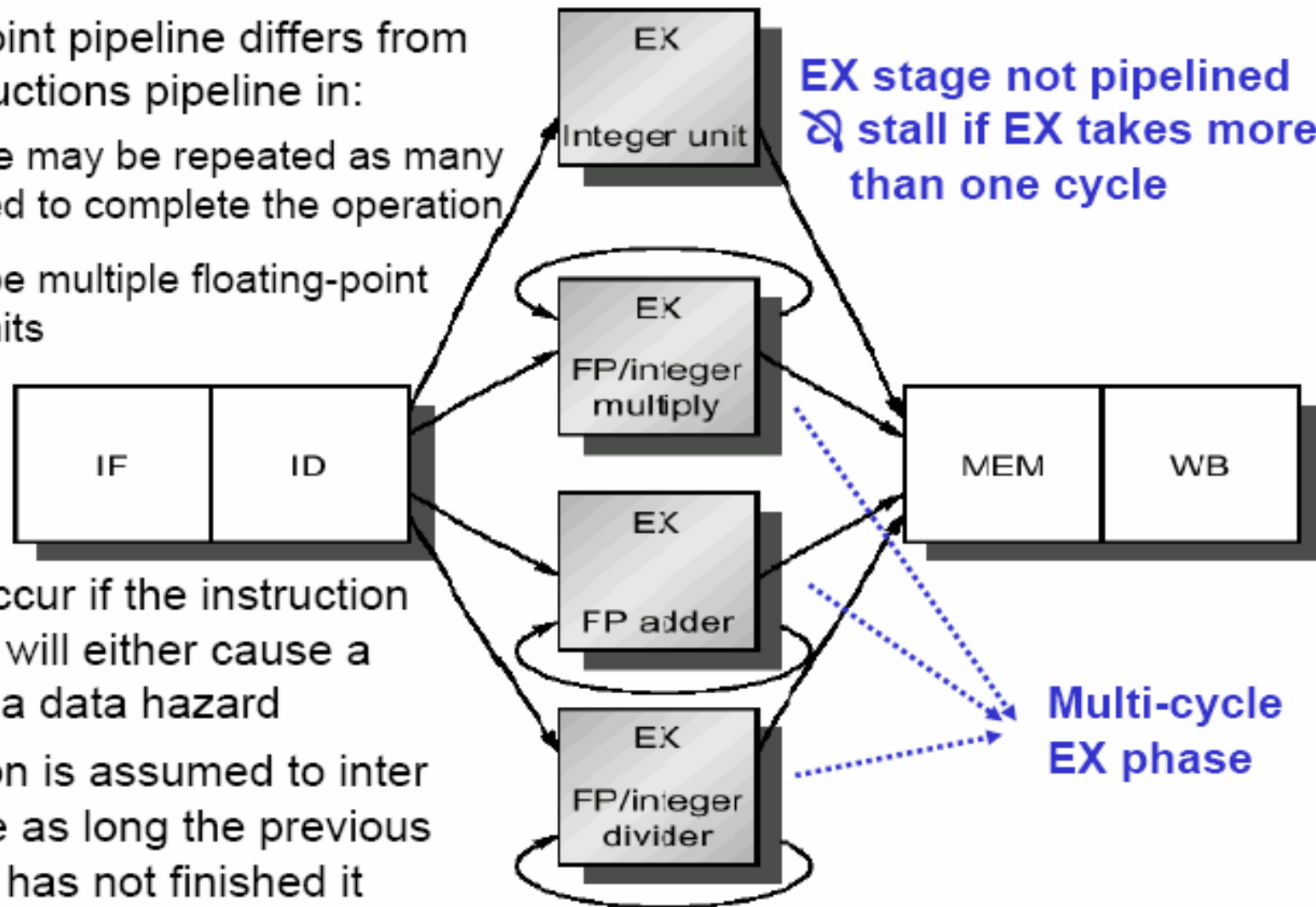
Floating-Point Pipeline

❑ It is impractical to require that all MIPS floating point operations complete in one clock cycle (complex logic and/or very long clock cycle)

❑ A floating-point pipeline differs from integer instructions pipeline in:

- ❶ The EX cycle may be repeated as many times as need to complete the operation
- ❷ There may be multiple floating-point functional units

Integer & FP instructions →

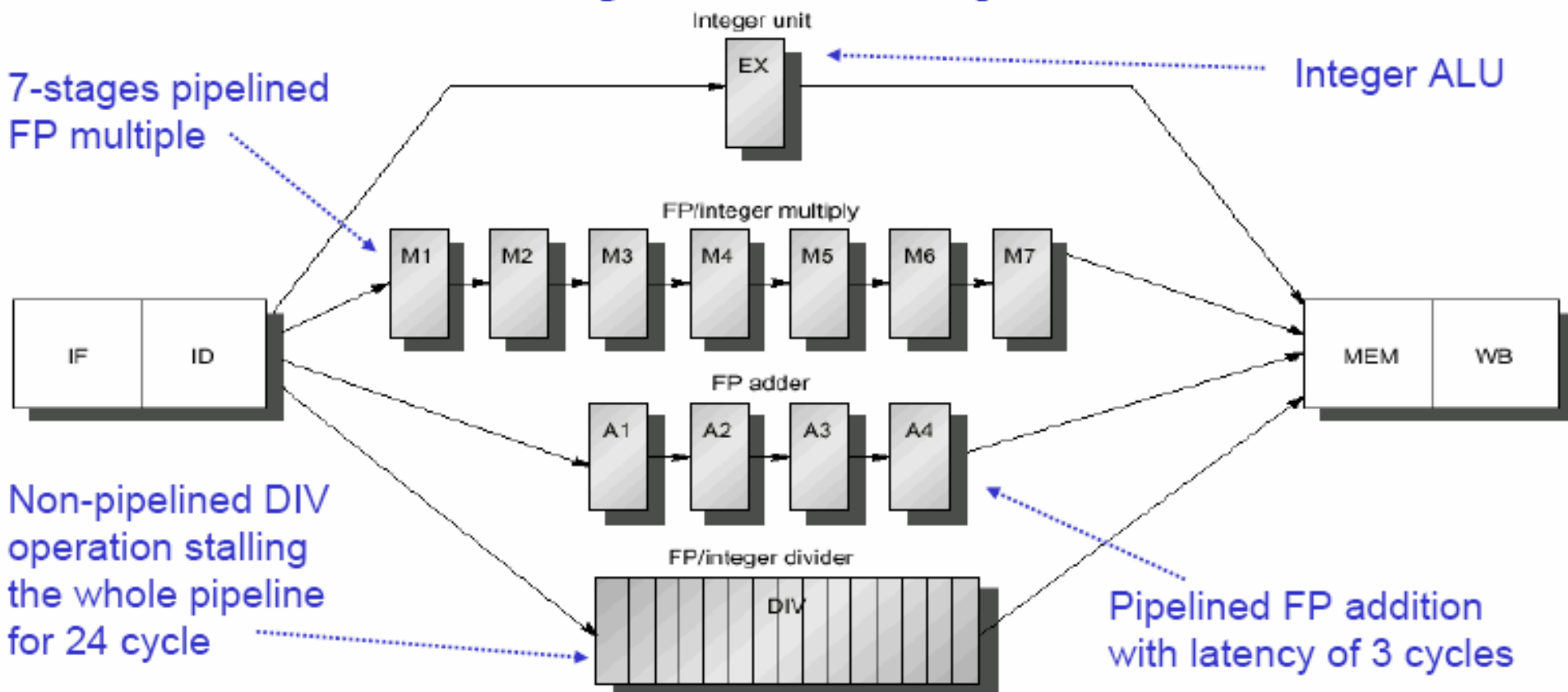


EX stage not pipelined
⌘ stall if EX takes more than one cycle

Multi-cycle EX phase

- ❑ A stall will occur if the instruction to be issued will either cause a structural or a data hazard
- ❑ No instruction is assumed to enter the EX stage as long as the previous instruction has not finished it

Multi-cycle FP Pipeline



MULTD	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
ADDD		IF	ID	A1	A2	A3	A4	MEM	WB		
LD			IF	ID	EX	MEM	WB				
SD				IF	ID	EX	MEM	WB			

Example: *blue* indicate where data is needed and *red* when result is available

Multi-cycle FP Pipeline EX Phase

- ❑ *Example*: assume that floating point add, subtract and multiple can be performed in stages while integer and FP divide cannot
- ❑ **Latency**: the number of intervening cycles between an instruction that produces a result and an instruction that uses
- ❑ Since most operations consume their operands at the beginning of the EX stage, latency is usually number of the stages of the EX an instruction uses
- ❑ Naturally long latency increases the frequency of RAW hazards
- ❑ **Initiation (Repeat) interval**: the number of cycles that must elapse between issuing two operations of a given type

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

Example of typical latency of floating point operations



FP Pipeline Challenges

Issues:

- Because the divide unit is not fully pipelined, structural hazards can occur
- Because the instructions have varying running times, the number of register writes required in a cycle can be larger than 1
- WAW hazards are possible, since instructions no longer reach WB in order
- WAR hazards are NOT possible, since register reads are still taking place during the ID stage
- Instructions can complete in a different order than they were issued, causing problems with exceptions
- Longer latency of operations makes stalls for RAW hazards more frequent


Instruction	Clock cycle number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD F4, 0(R2)	IF	ID	EX	MEM	WB												
MULTD F0, F4, F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADDD F2, F0, F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	
SD 0(R2), F2					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

Example of RAW hazard caused by the long latency



Write-Caused Structural Hazard

Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
MULTD F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADDD F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
LD F2, 0(R2)							IF	ID	EX	MEM	WB

- ❑ At cycle 11, the MULTD, ADDD and LD instructions will try to write back causing structural hazard if there is only one write port
- ❑ Additional write ports are not cost effective since they are rarely used and it is better to detect and resolve the structural hazard
- ❑ Structural hazards can be detected at the ID stage and the instruction will be stalled to avoid a conflict with another at the WB
- ❑ Alternatively, structural hazards can be handled at the MEM or WB by  rescheduling the usage of the write port

WAW Data Hazards

Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
MULTD F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADDD F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
LD F2, 0(R2)						IF	ID	EX	MEM	WB	
....							IF	ID	EX	MEM	WB

❑ WAW hazards can be corrected by either:

- ❶ Stopping the latter instruction (LD in example) at the MEM until it is safe
- ❷ Preventing the first instruction from overwriting the register

❑ Correcting WAW Hazards at cycle 11 is not problematic unless there is an instruction between ADDD and LD that read F2 causing RAW hazard

❑ WAW hazards can be detected at the ID stage and thus the first instruction can be converted to no-op

❑ Since WAW hazards are generally very rare, designers usually go with the simplest solution

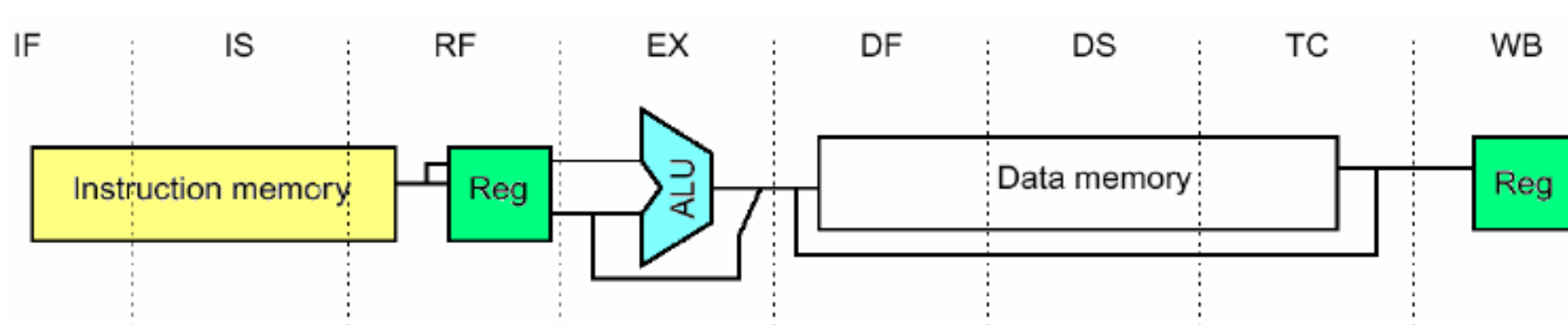
Detecting Hazards

- ❑ Hazards can occur among FP instructions and among FP and integer instructions
- ❑ Using separate register files for integer and FP limits potential hazards to just FP load and store instructions
- ❑ Assuming all checks are to be performed in the ID phase, hazards can be detected through the following steps:
 - ① Check for structural hazards:
 - Wait if the functional unit is busy (Divides in our case)
 - Make sure the register write port is available when needed
 - ② Check for a RAW data hazard
 - Requires knowledge of latency and initiation interval to decide when to forward and when to stall
 - ③ Check for a WAW data hazard
 - Write completion has to be estimated at the ID stage to check with other instructions in the pipeline
- ❑ Data hazard detection and forwarding logic can be derived from values stored at the data stationary between the different stages



MIPS 4000 Pipeline

- ❑ Implements the MIPS-3 64-bit instruction
- ❑ Uses 8 stages pipeline through pipelining instruction and data cache access
- ❑ Deeper pipeline allows for higher clock rate but increases load/branch delays



IF: First half of instr. fetch; PC selection, initiation of instruction cache access

IS: Second half of the instruction fetch completing instruction cache access


RF: Instr. Decode, register fetch, hazard checking and instr. cache hit detection

EX: ALU operations, effective address calculation, and condition evaluation

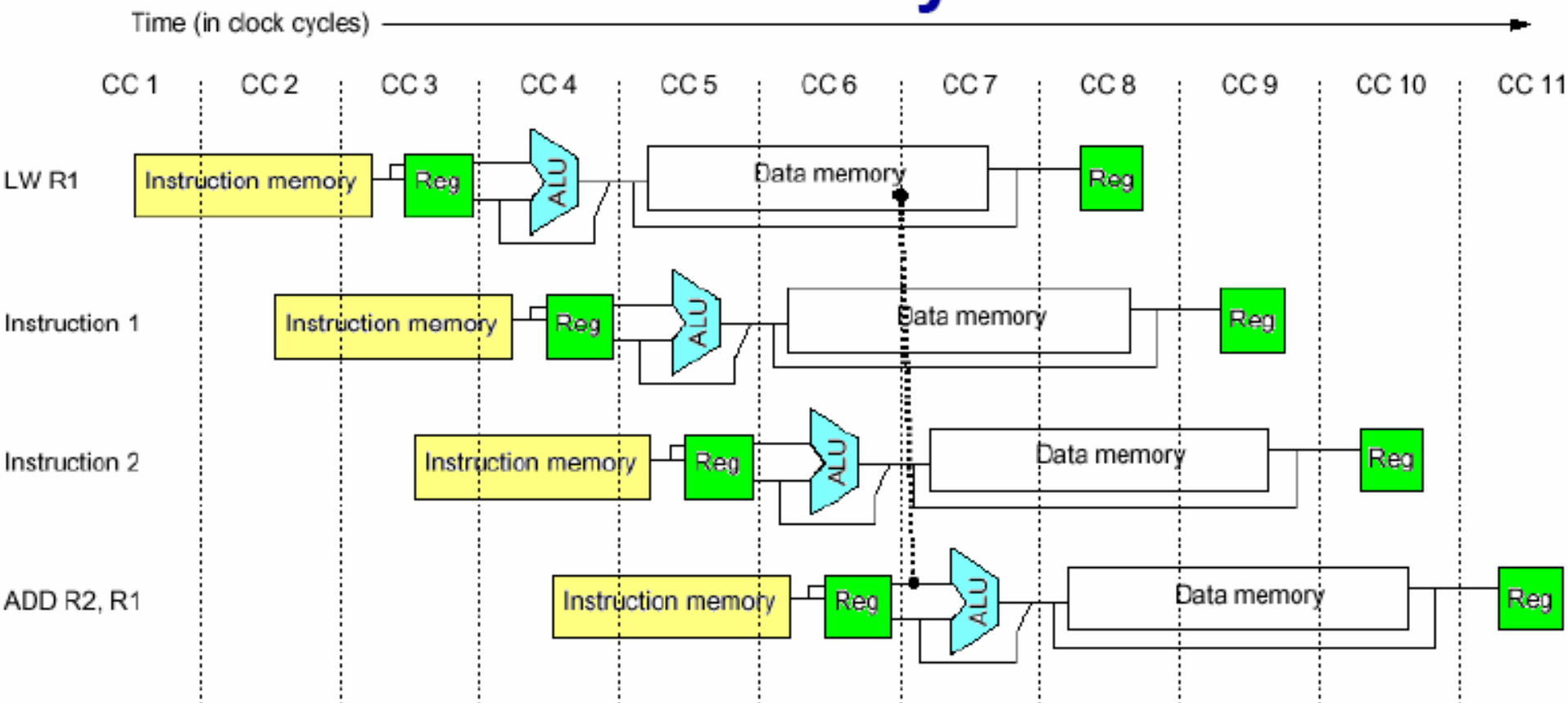
DF: Data fetch, first half of data cache access

DS: Second half of data fetch, completion of data cache access

TC: Tag check, determine whether the data cache access hit

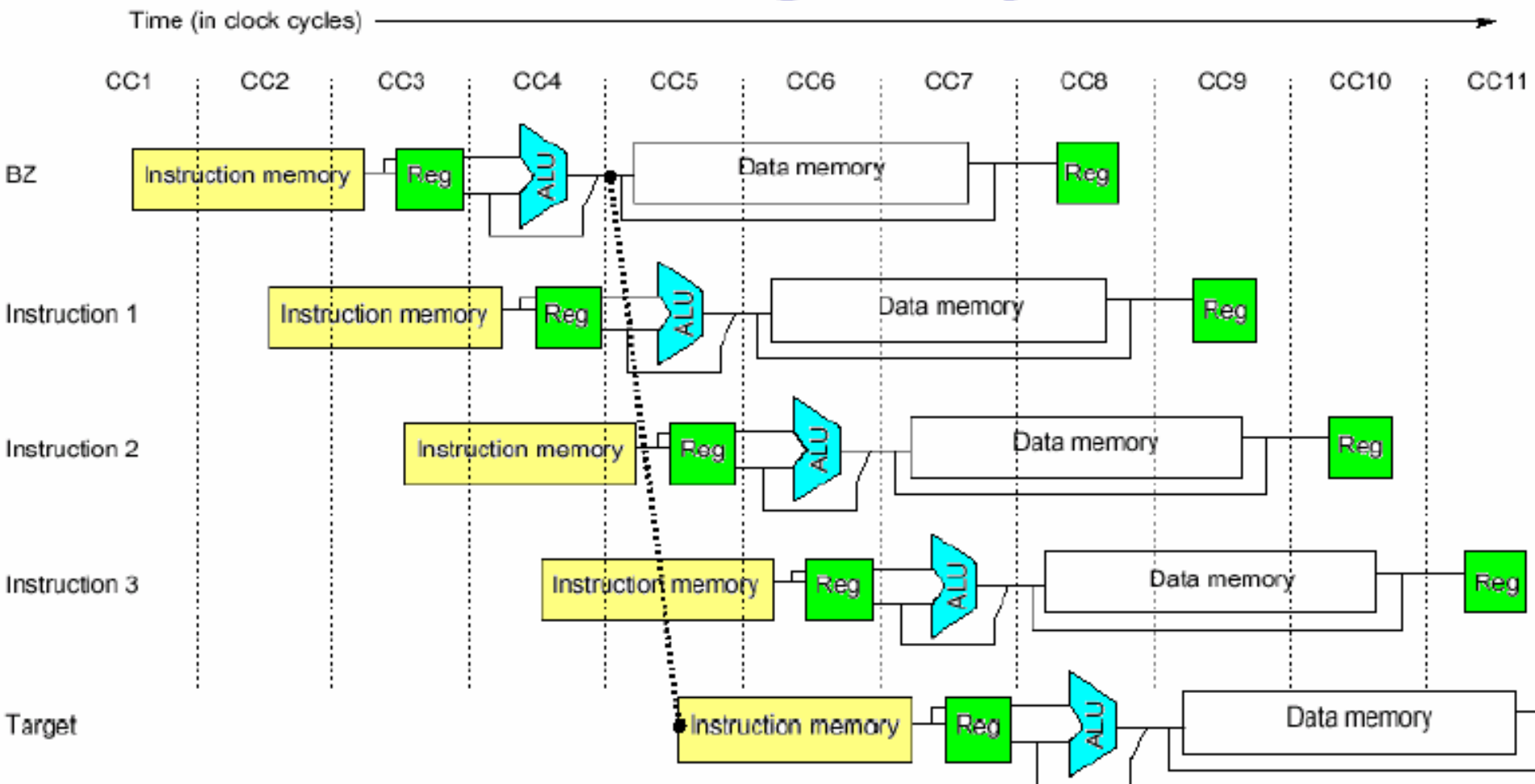
 **WB:** Write back for loads and register-register operations

Load Delays



- ❑ Data value are available at the end of the DS cycle, cause a two cycle delay for load instructions
- ❑ Pipelined cache access increases the need for forwarding and complicates the forwarding logic
- ❑ A cache miss will stall the pipeline additional one (or more) cycles

Branching Delays



- ❑ Branch conditions are computed during EX stage extending the basic branch delay to 3 cycles
- ❑ MIPS allows for a single-cycle delay branching and a predict-not-taken strategy for the remaining two branching delay cycles