Pipelining for DLX 560 Prozessor

Pipelining : implementation-technique

Pipelining makes CPUs fast.

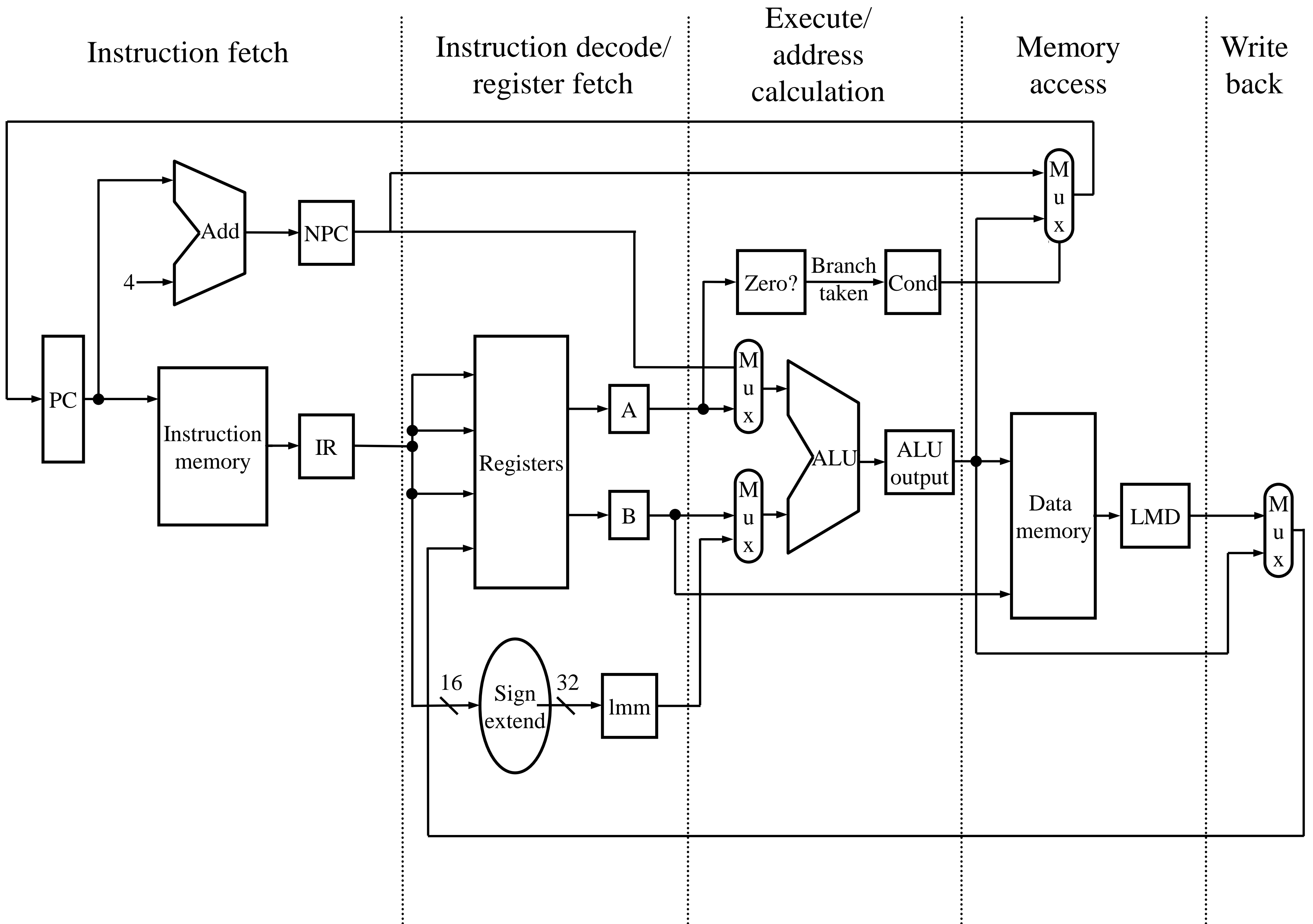
pipe stages

As many instructions as possible in one unit of time

Pipelining can

- **Reduce CPI**
- **Reduce cycle time**

- **Invisible for user**



DLX-Datenpfad mit Taktzyklen (ohne Pipelining)

1. Instruction Fetch Zyklus: (IF):

IR	**<--**	**Mem[PC]**
NPC	**<--**	**PC + 4**

2. Instruction decode / Register fetch Zyklus (ID):

A	**<--**	**Regs[$IR_{6..10}$]**
B	**<--**	**Regs[$IR_{11..15}$]**
IMM	**<--**	$((IR_{16})^{16}\#\#IR_{16..31}$

3. Execution / Effective Address Zyklus (EX):

Instructin with memory access (load/store):

ALUoutput <-- A + IMM

. Register-Register ALU-instruction:

ALUoutput <-- A func B

Register-Immediate ALU-instruction:

ALUoutput <-- A op IMM

Branch-instruction:

ALUoutput <-- NPC + IMM
Cond <-- (A op 0)

4. Memory Access / Branch Completion phase (MEM):

LW-instruction (load):

LMD ← Mem[ALUoutput]
PC ← NPC

SW-instruction (store):

Mem[ALUoutput] ← B
PC ← NPC

Branch-instruction:

if Cond then
PC <-- ALUoutput
else PC <-- NPC

5. Write back phase (WB):

Register-Register ALU instruction:

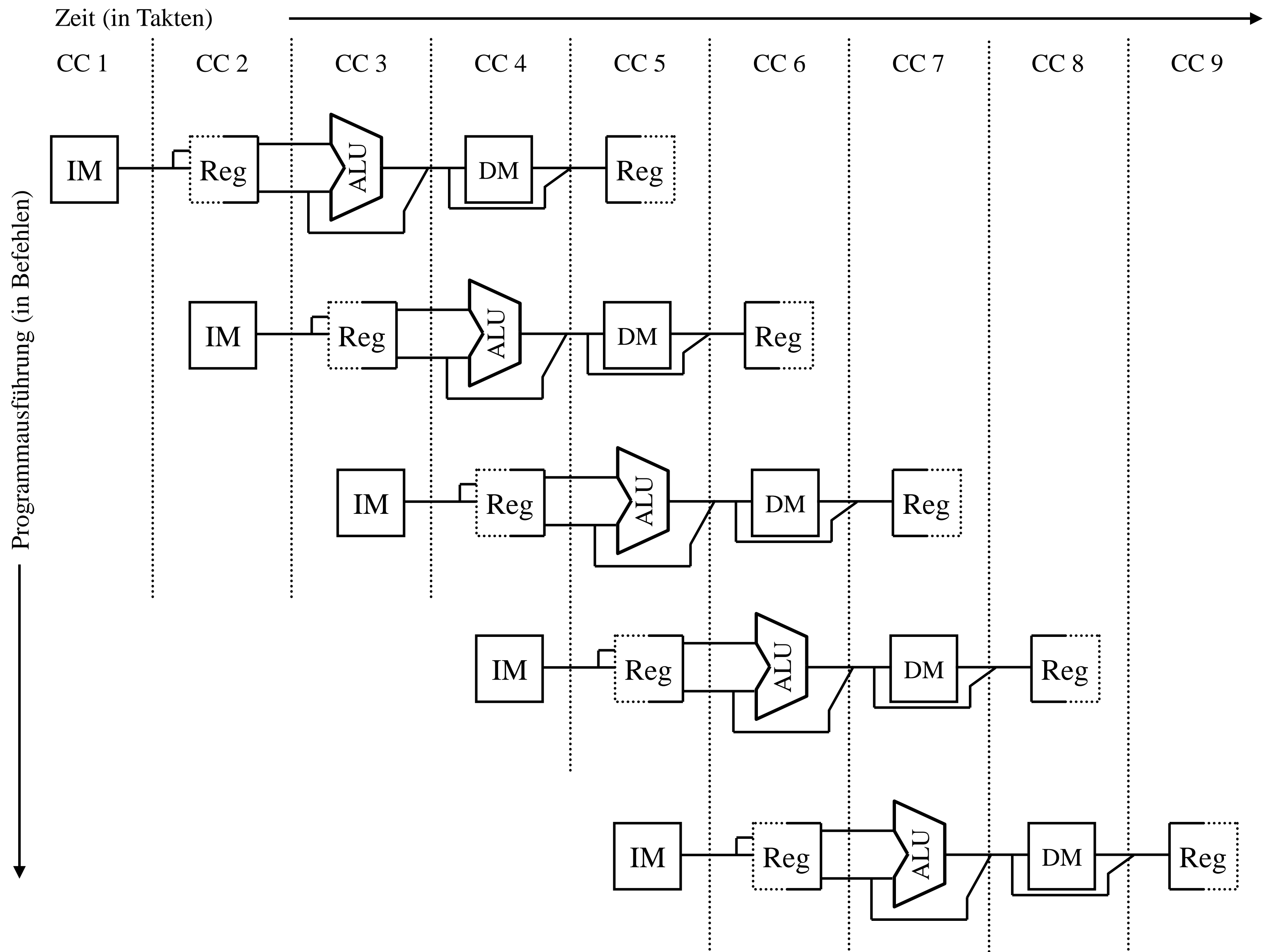
Regs[IR$_{16..20}$] <-- ALUoutput

Register-Immediate ALU instruction:

Regs[IR$_{11..15}$] <-- ALUoutput

Lade Befehl:

Regs[IR$_{11..15}$] <-- LMD



Vereinfachte Darstellung des DLX-Datenpfads

Pipeline Diagramm

Befehl	Takt 1	2	3	4	5	6	7	8	9
Befehl *i*	IF	ID	EX	MEM	WB				
Befehl *i* + *1*		IF	ID	EX	MEM	WB			
Befehl *i* + *2*			IF	ID	EX	MEM	WB		
Befehl *i* + *3*				IF	ID	EX	MEM	WB	
Befehl *i* + *4*					IF	ID	EX	MEM	WB

Stufe	Was wird alles getan		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALU Output} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.$IR_{6..10}$]; ID/EX.B ← Regs[IF/ID.$IR_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← $(IR_{16})^{16}$ # # $IR_{16..31}$;		
	ALU Befehl	**Load oder store Befehl**	**Verzweigungsbefehl**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM. ALUOutput ← ID/EX. A func ID/EX. B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B ← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC+ID.EX.Imm; EX/MEM.cond ← (ID/EX.A *op* 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs [MEM/WB. $IR_{16..20}$] ← MEM/WB.ALUOutput; or Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.LMD;	

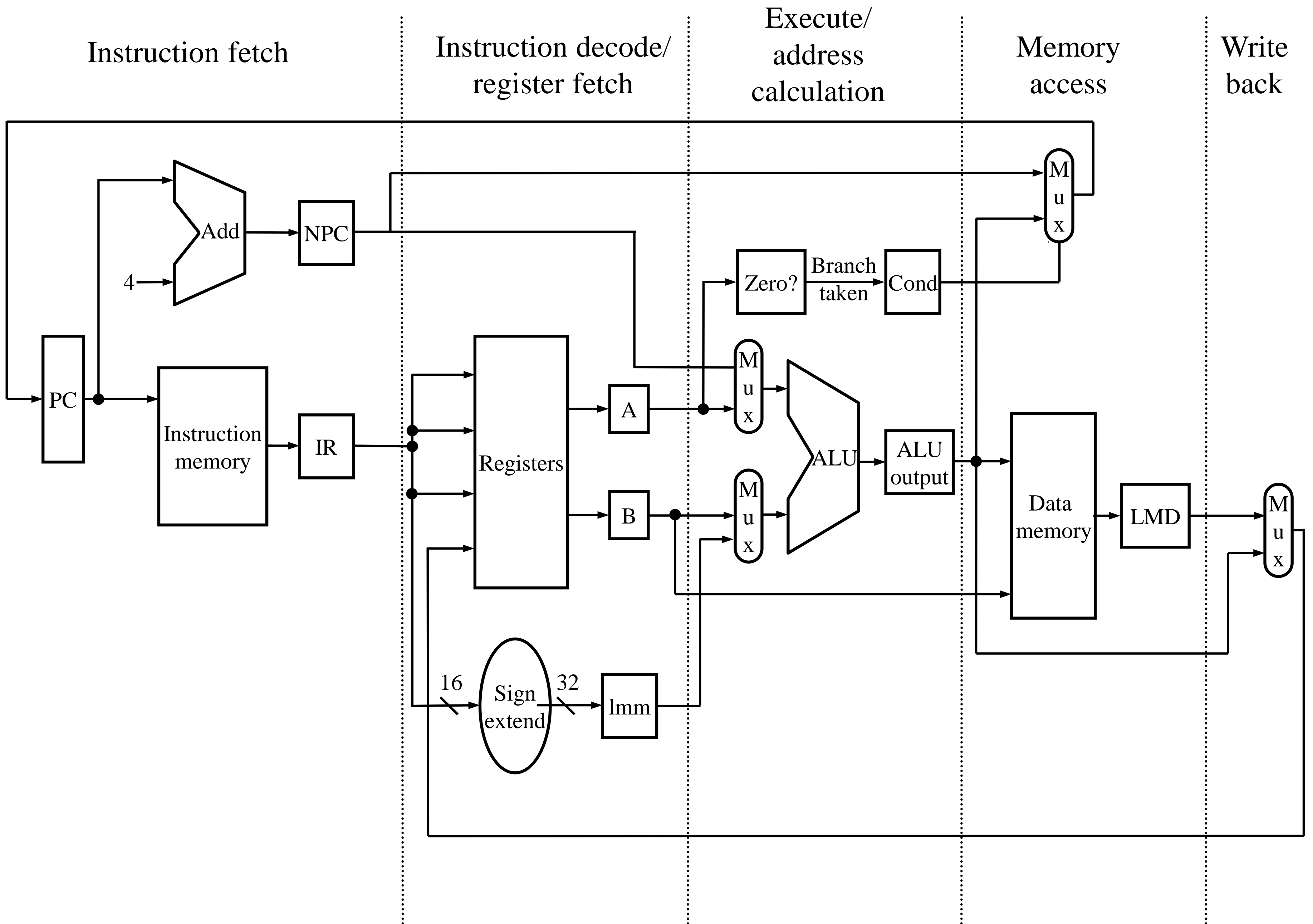
Die Aktivitäten in den ersten zwei Stufen sind nicht befehlsabhängig. Das muß auch so sein, weil wir den Befehl ja erst am Ende der zweiten Stufe interpretieren können.

Durch die festen Bit-Positionen der Operandenregister im IR-Feld ist die Dekodierung und das Register-Lesen in einer Phase möglich.

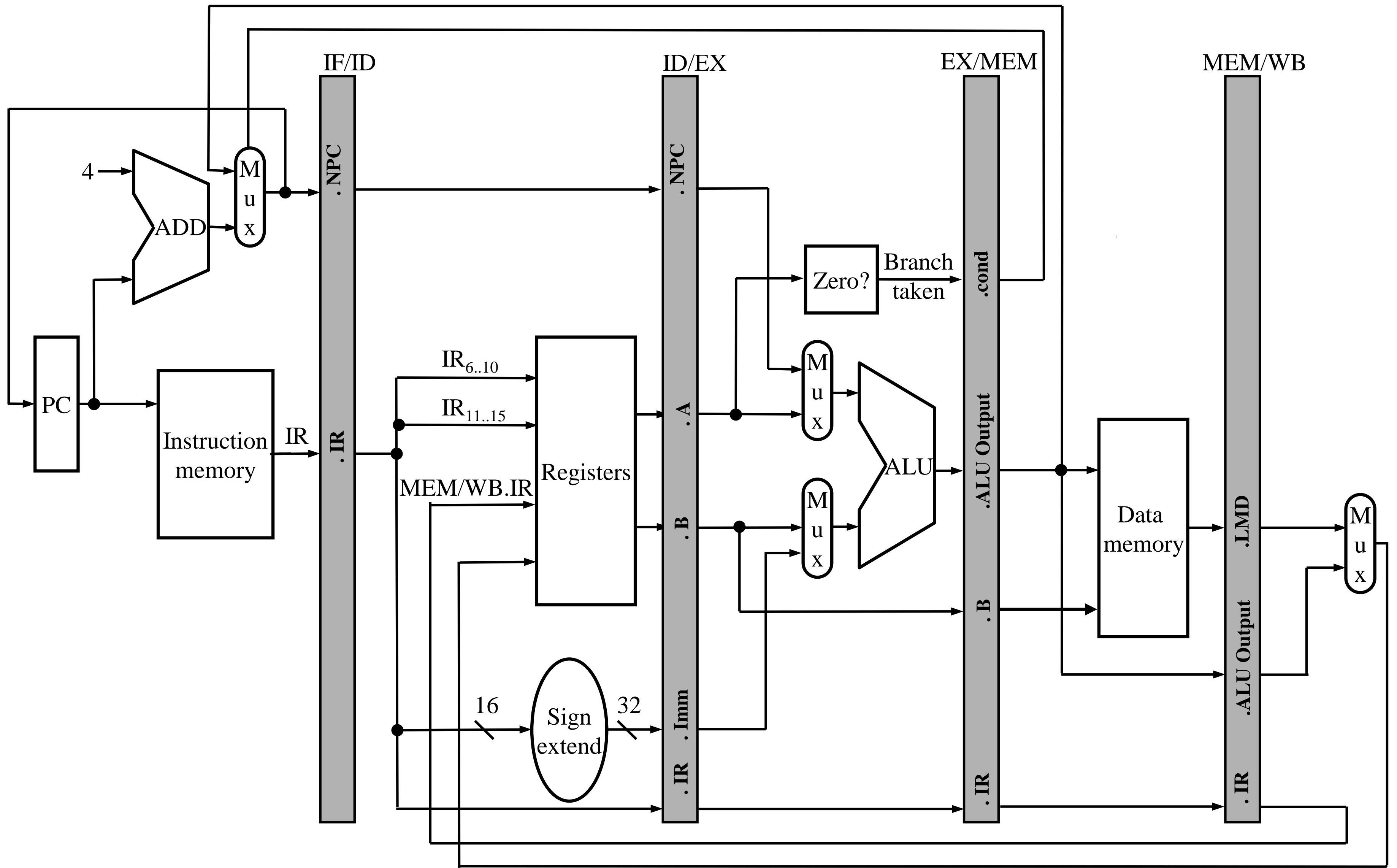
Um den Ablauf in dieser einfachen Pipeline zu steuern, ist die Steuerung der vier Multiplexer in dem Diagramm erforderlich:

oberer ALU-input-Mux:	**Verzweigung oder nicht**
unterer ALU-input-Mux:	**Register-Register-Befehl oder nicht**
IF-Mux:	**EX/MEM.cond**
WB-Mux:	**load oder ALU-Operation**

Es gibt einen fünften (nicht eingezeichneten Mux), der beim WB auswählt, wo im MEM/WB.IR die Adresse des Zielregisters steht, nämlich an Bits 16..20 bei einem Register-Register-ALU-Befehl und an Bits 11..15 bei einem Immediate- oder Load-Befehl.



DLX-Datenpfad mit Taktzyklen (ohne Pipelining)


IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M
u
x
PC
Instruction memory
IR
.NPC
.IR
$IR_{6..10}$
$IR_{11..15}$
MEM/WB.IR
Registers
16
Sign extend
32
.NPC
.A
.B
.Imm
.IR
Zero?
Branch taken
M
u
x
ALU
M
u
x
.cond
.ALU Output
.B
.IR
Data memory
.LMD
.ALU Output
.IR
M
u
x

All temporary registers are now in the pipeline latches.
Pipeline latches hold data and control information (IR, NPC, …)

Within one clock cycle data are read from the pipeline laches, move through the units of the pipeline stage and are stored in the next pipeline latch.

Stufe	**Was wird alles getan**		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALU Output} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.$IR_{6..10}$]; ID/EX.B ← Regs[IF/ID.$IR_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← $(IR_{16})^{16}$ # # $IR_{16..31}$;		
	ALU Befehl	**Load oder store Befehl**	**Verzweigungsbefehl**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM. ALUOutput ← ID/EX. A func ID/EX. B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B ← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC+ID.EX.Imm; EX/MEM.cond ← (ID/EX.A *op* 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs [MEM/WB. $IR_{16..20}$] ← MEM/WB.ALUOutput; or Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.LMD;	

The activities of the first two stages are not instruction dependent. This ist because the instruction needs to be decoded first.

Register fetch is possible without knowledge of the corresponing instruction.

To control the pipeline, there are only the following control bits required:

upper ALU-input-Mux:	**Branch or not**
lower ALU-input-Mux:	**Register-Register-Instruction or not**
IF-Mux:	**EX/MEM.cond**
WB-Mux:	**load or ALU-Operation**

Pipeline Hazards

There are cases where the pipeline cannot be executed one step per clock cycle. These are called pipeline hazards.

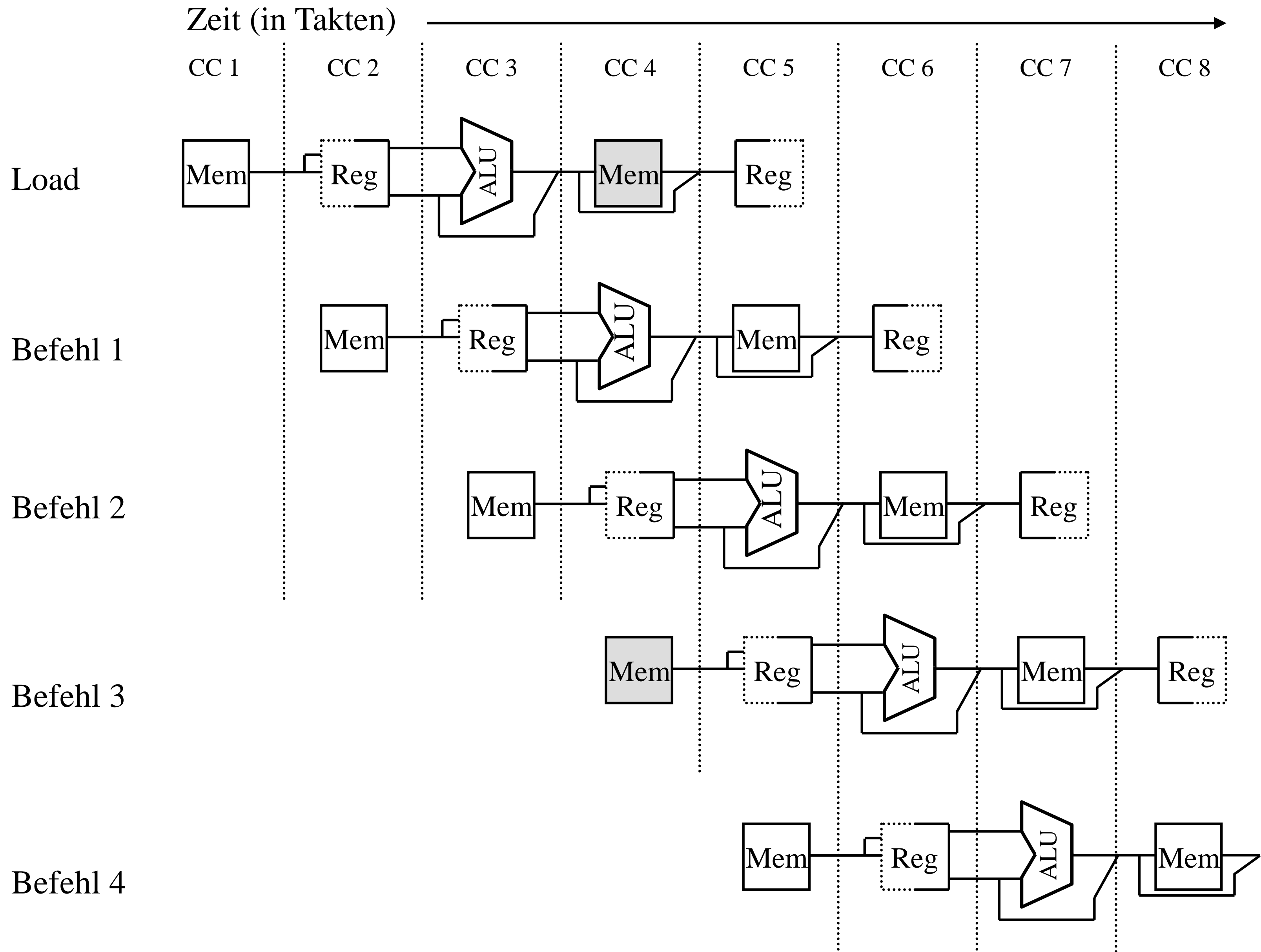
Struktural hazards: Two different stages want access to the same hardware.

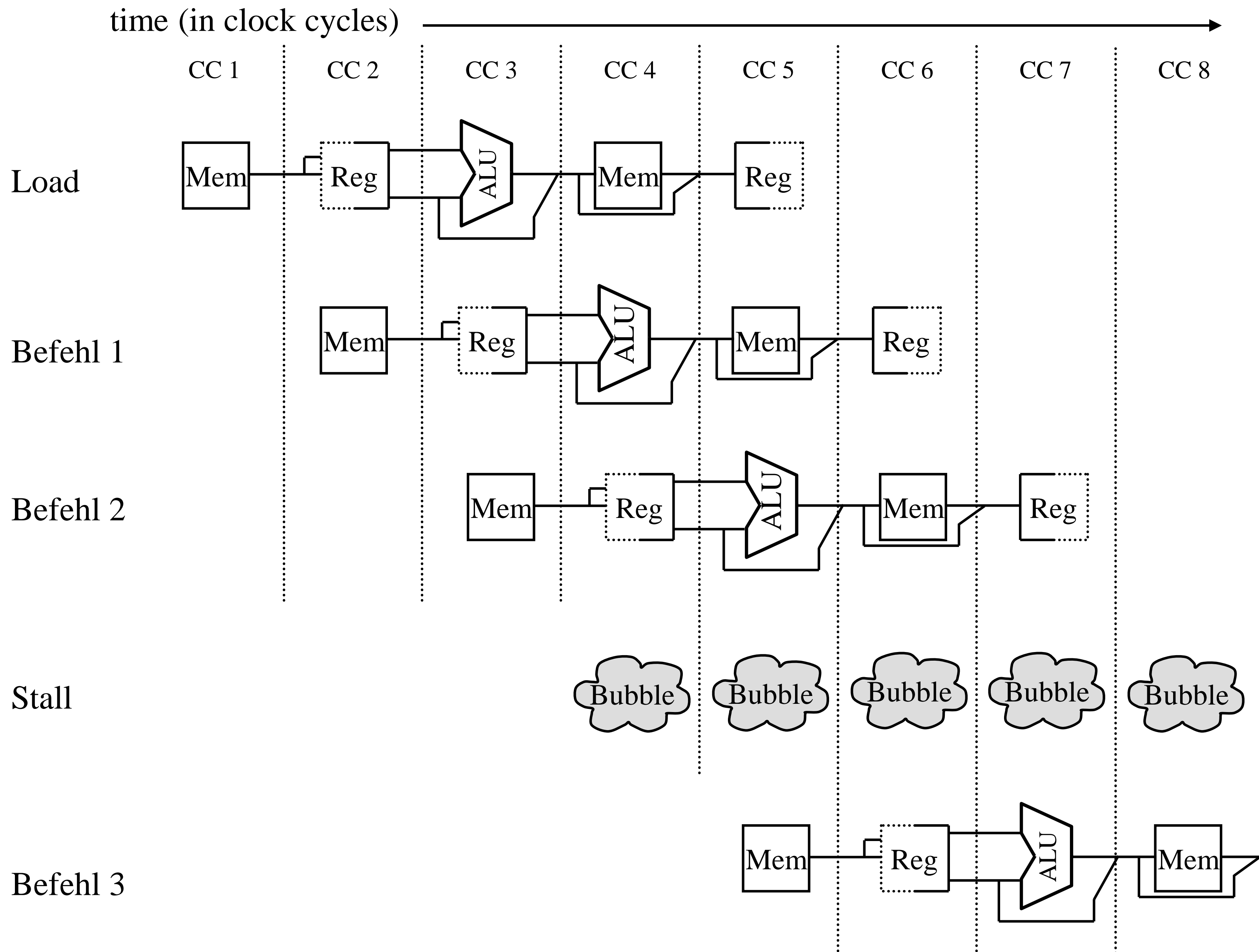
Data hazards: Registers are read before their actual contents is written.

Control hazards: PC after branches and jumps

Hazards can be repaired by stalling the pipeline. Instruction before the stall must be finished, instructions after the stall must wait with their execution.

No new instructions are fetched as long as the pipeline is stalled.


Zeit (in Takten)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
Load
Befehl 1
Befehl 2
Befehl 3
Befehl 4
Mem
Reg
ALU

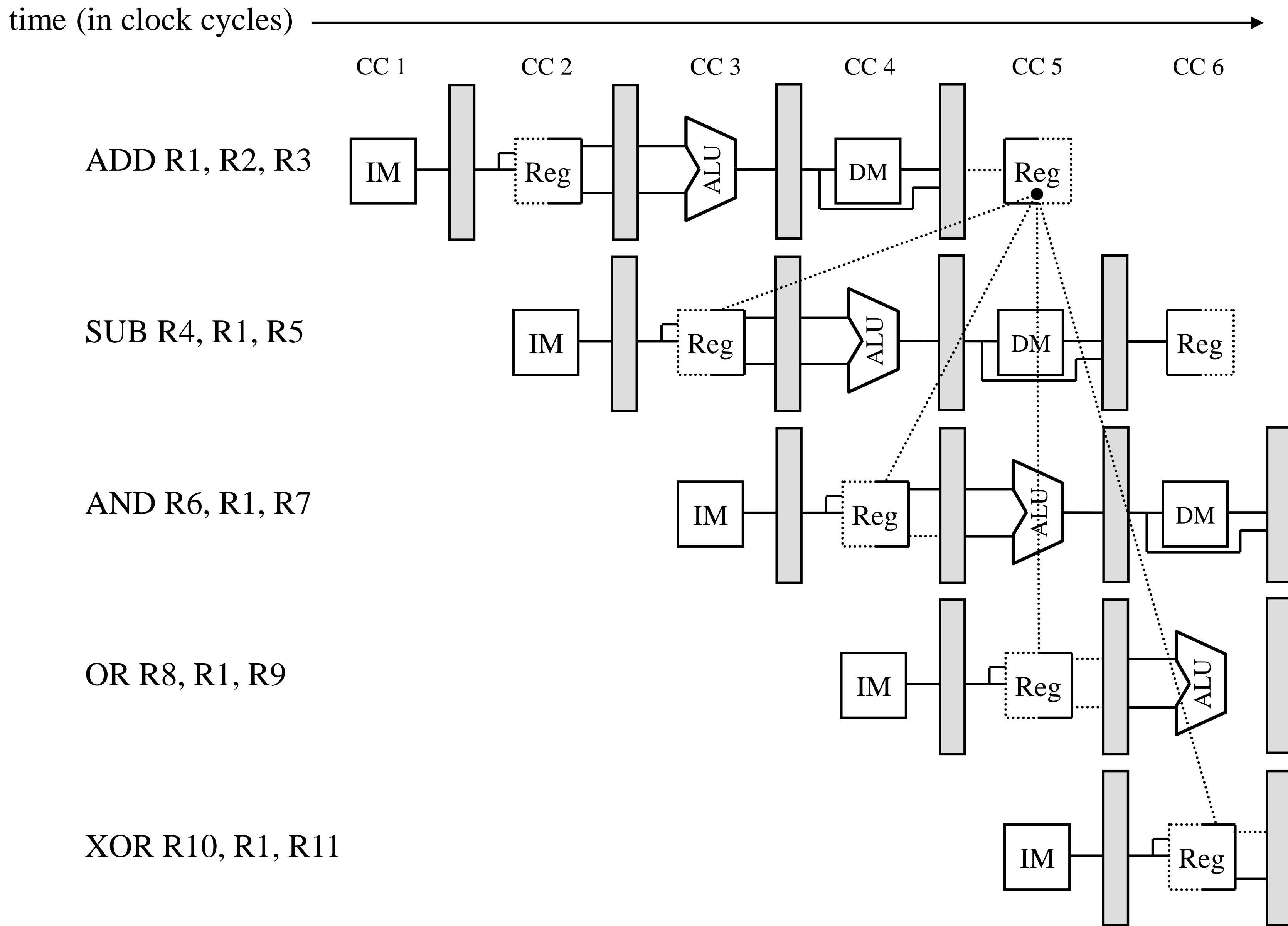

time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
Load
Mem
Reg
ALU
Mem
Reg
Befehl 1
Mem
Reg
ALU
Mem
Reg
Befehl 2
Mem
Reg
ALU
Mem
Reg
Stall
Bubble
Bubble
Bubble
Bubble
Bubble
Befehl 3
Mem
Reg
ALU
Mem

Befehl	**Takt**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load Befehl	IF	ID	EX	MEM	WB					
Befehl $i+1$		IF	ID	EX	MEM	WB				
Befehl $i+2$			IF	ID	EX	MEM	WB			
Befehl $i+3$				stall	IF	ID	EX	MEM	WB	
Befehl $i+4$						IF	ID	EX	MEM	WB
Befehl $i+5$							IF	ID	EX	MEM
Befehl $i+6$								IF	ID	EX

Data hazards

Consider the following piece of Assembler-Program

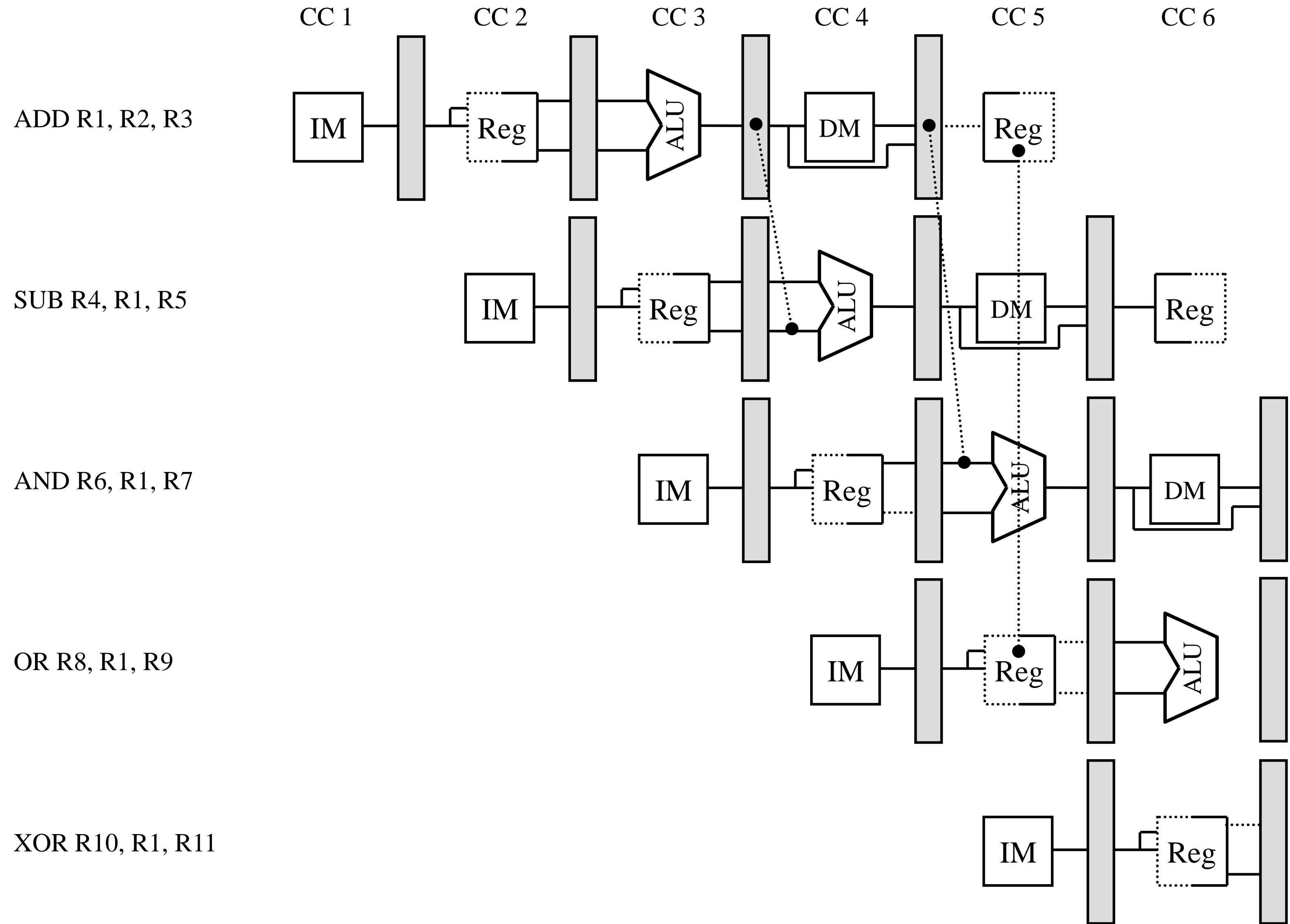
```
ADD     R1, R2, R3
SUB     R4, R5, R1
AND     R6, R1, R7
OR      R8, R1, R9
XOR     R10, R1, R11
```

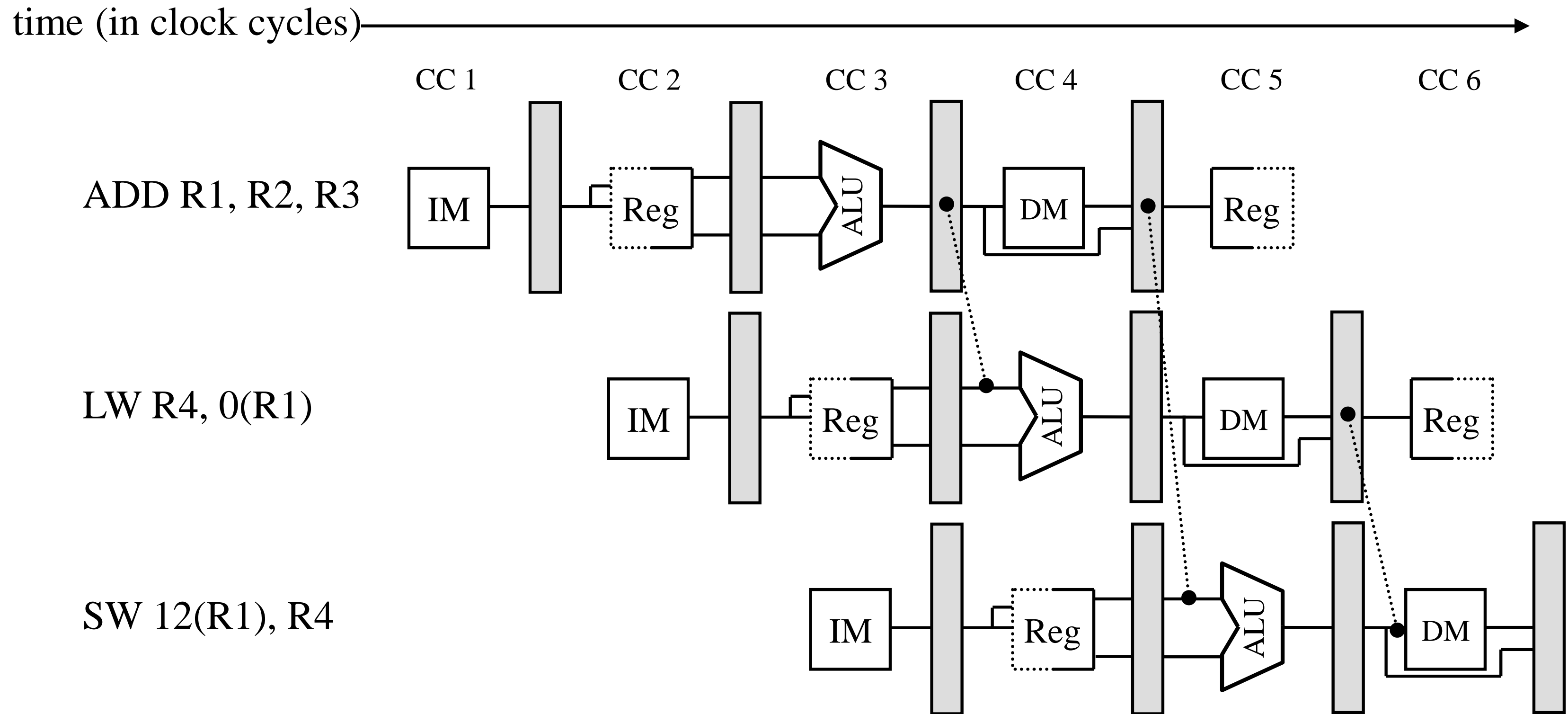

time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
ADD R1, R2, R3
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
IM
Reg
ALU
DM
Reg

Data hazards can be repaired by **forwarding**.

1. The result of the ALU (aus EX/MEM) is also braught back to the ALU-input.

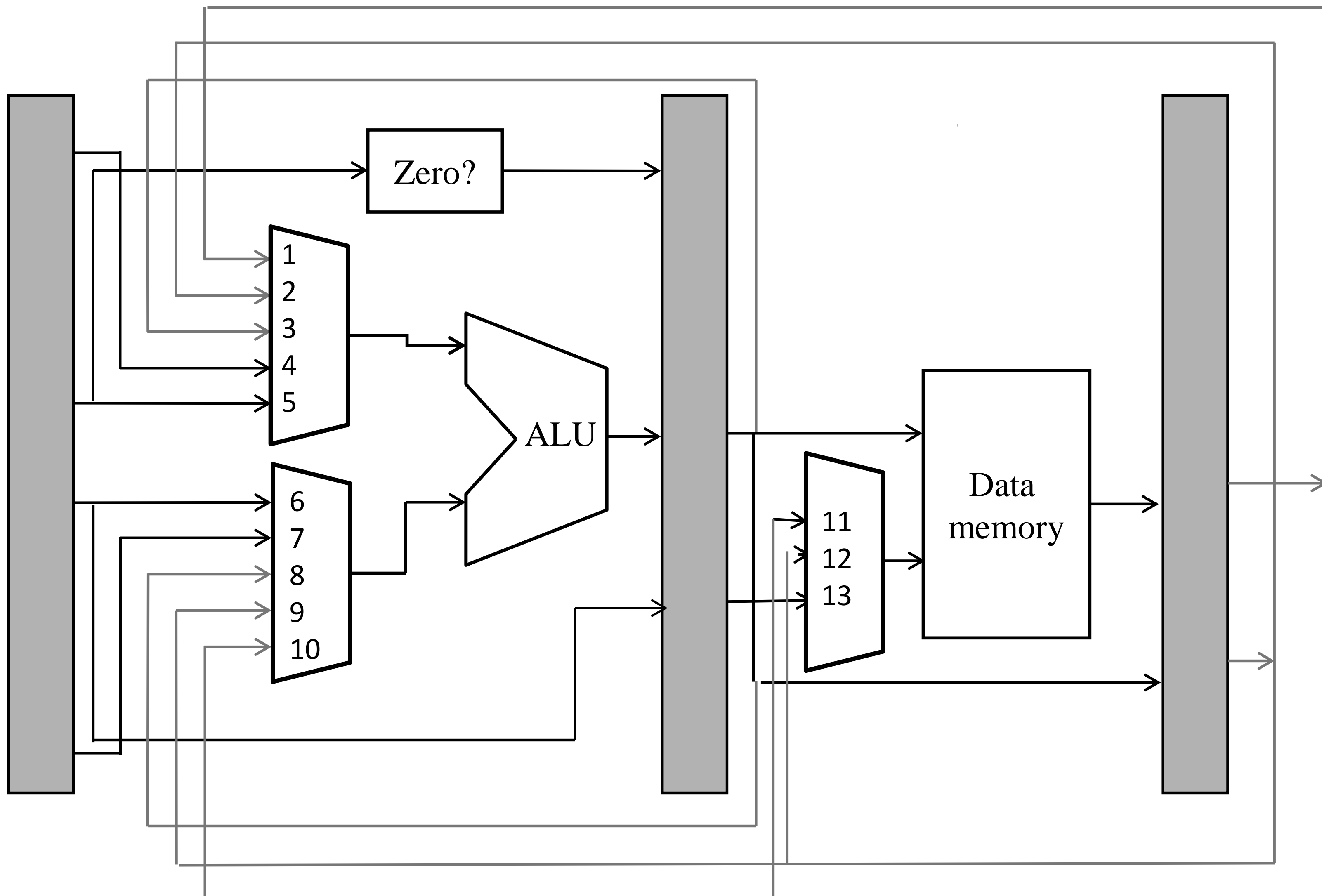
2. A specific forwarding control logic chooses the corresponding input of the ALU-Mux.


Zeit (in Takten)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
ADD R1, R2, R3
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
IM
Reg
ALU
IM
Reg


time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
ADD R1, R2, R3
LW R4, 0(R1)
SW 12(R1), R4
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM

Required paths for forwarding

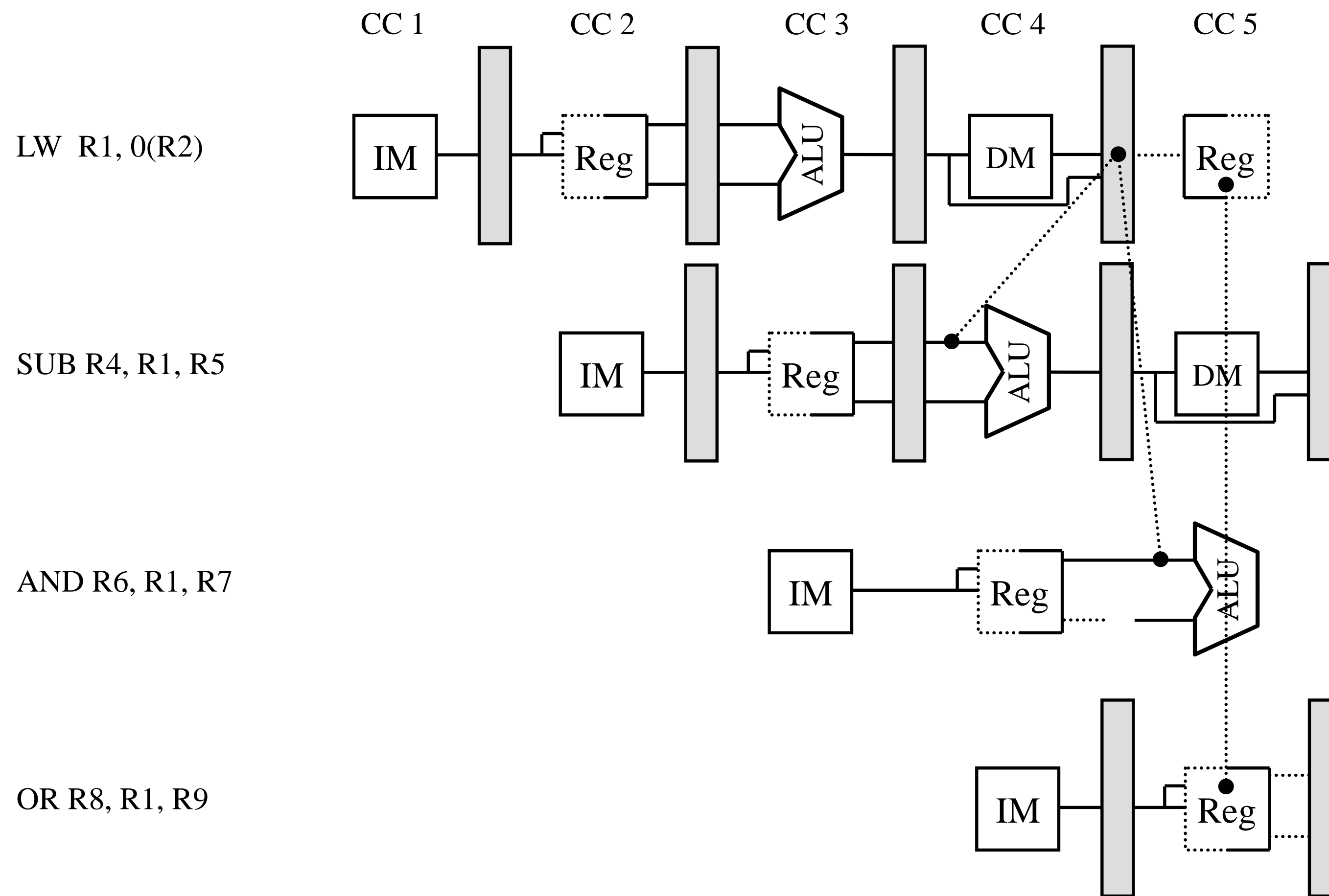
ID/EX EX/MEM MEM/WB



Load problem

There are hazards that cannot be repaired by forwarding

LW	**R1, 0(R2)**
SUB	**R4, R1, R3**
AND	**R6, R1, R7**
OR	**R8, R1, R9**

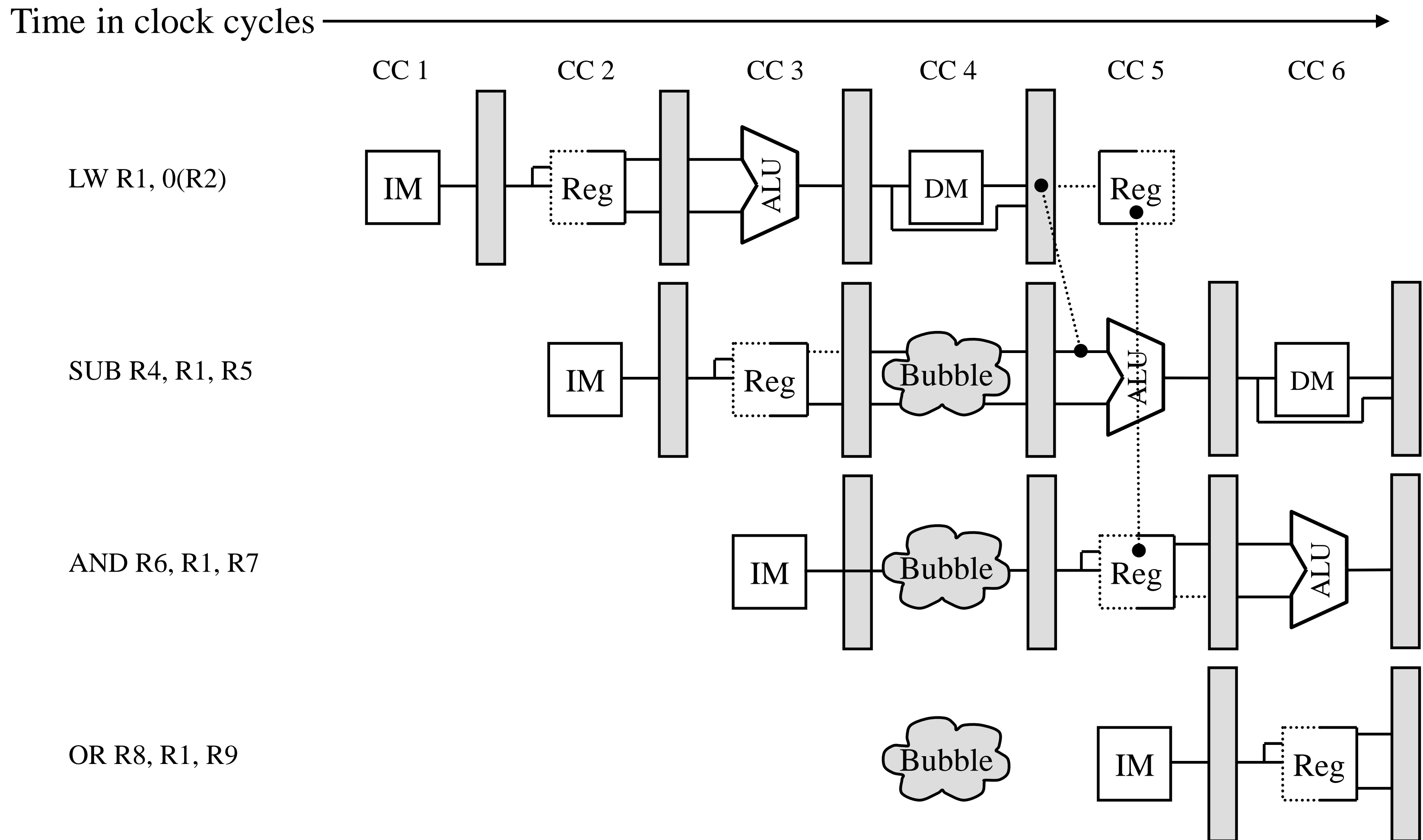

time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
LW R1, 0(R2)
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
IM
Reg
ALU
IM
Reg

This requires a pipeline interlock

Only the instructions before LW are executed

The instrudctions after LW are stalled for one clock cycle.

The pause between LW and the following instruction is calles a pipeline bubble


Time in clock cycles
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
LW R1, 0(R2)
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
IM
Reg
ALU
DM
Reg
IM
Reg
Bubble
ALU
DM
IM
Bubble
Reg
ALU
Bubble
IM
Reg

LW R1,0(R1)	IF	ID	EX	MEM	WB				
SUB R4,R1,R5		IF	ID	EX	MEM	WB			
AND R6,R1,R7			IF	ID	EX	MEM	VB		
OR R8,R1,R9				IF	ID	EX	MEM	WB	

LW R1,0(R1)	IF	ID	EX	MEM	WB				
SUB R4,R1,R5		IF	ID	stall	EX	MEM	WB		
AND R6,R1,R7			IF	stall	ID	EX	MEM	WB	
OR R8,R1,R9				stall	IF	ID	EX	MEM	WB