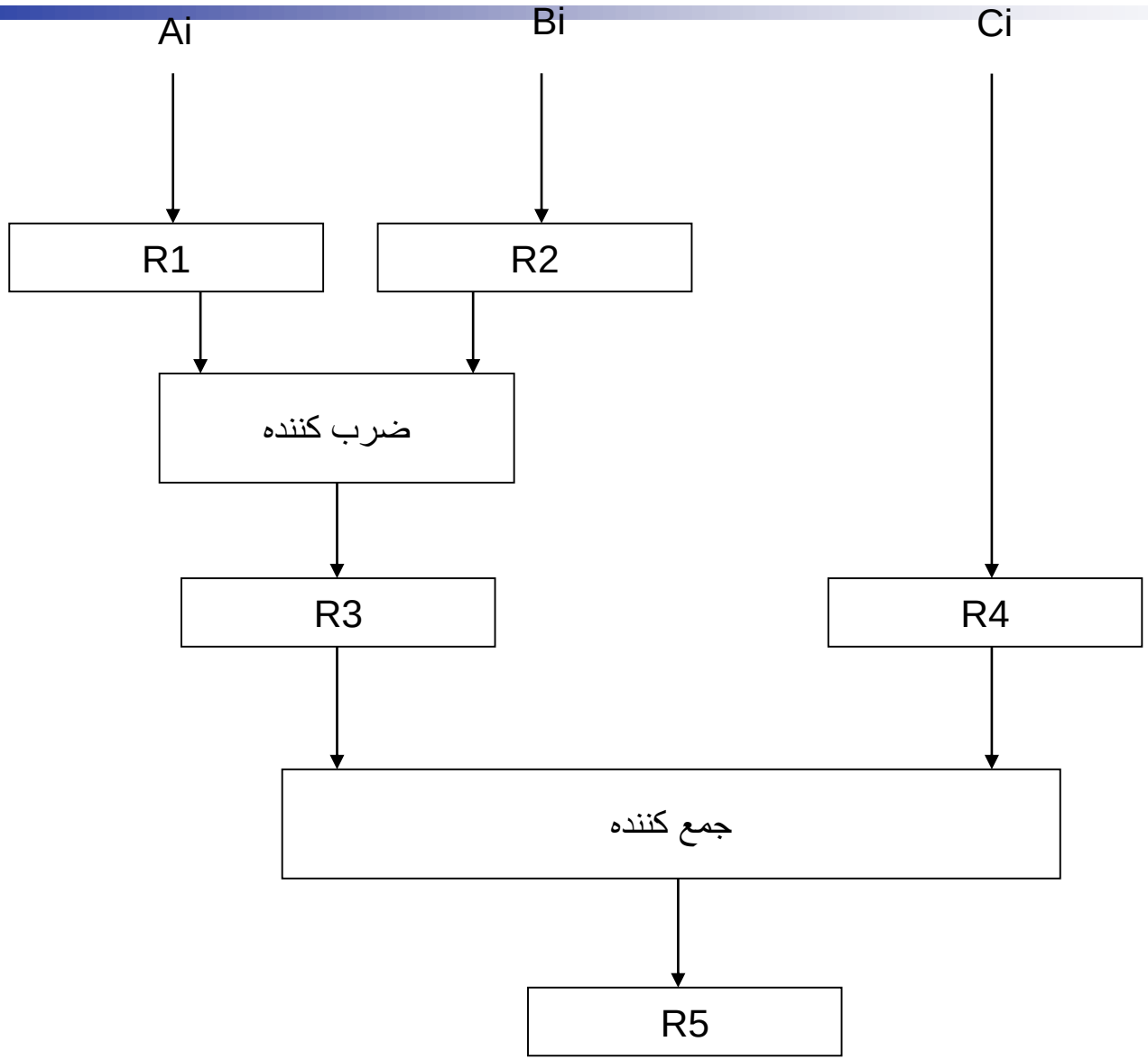
Chapter 4

The Processor

[Adapted from *Computer Organization and Design, & 5th Edition*,
Patterson & Hennessy, © 2011, MK]

در کامپیوتر ها برای موازی سازی و افزایش کارآیی
از 2 نوع خط لوله استفاده می شود:
خط لوله محاسباتی و خط لوله دستور العمل

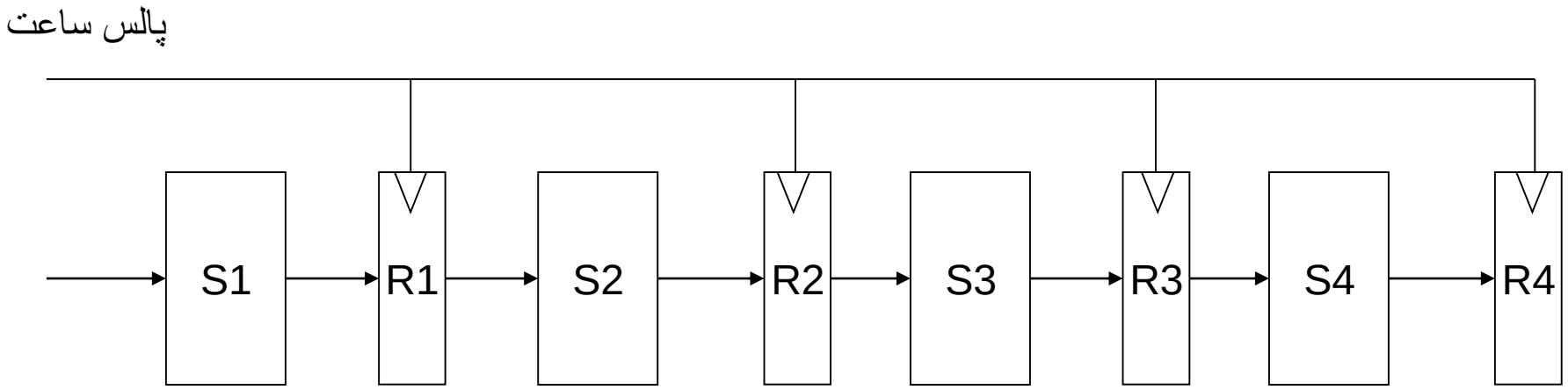
Arithmetic Pipelineمثالی از خط لوله محاسباتی (شکل از فصل 9 مانو):



محتویات ثبات ها در مثال خط لوله محاسباتی (شکل از مانو)

تعداد پالس ساعت	قطعه 1 - R1	R2	قطعه 2 R3	R4	قطعه 3 R5
1	A1	B1	-	-	-
2	A2	B2	A1×B1	C1	-
3	A3	B3	A2×B2	C2	A1×B1+C1
4	A4	B4	A3×B3	C3	A2×B2+C2
5	A5	B5	A4×B4	C4	A3×B3+C3
6	A6	B6	A5×B5	C5	A4×B4+C4
7	A7	B7	A6×B6	C6	A5×B5+C5
8	-	-	A7×B7	C7	A6×B6+C6
9	-	-	-	-	A7×B7+C7

ملاحظات عمومی خط لوله (شکل از مانو)



خط لوله چهارقسمتی

قسمت ها	1	2	3	4	5	6	7	8	9	سیکل ساعت
1	T1	T2	T3	T4	T5	T6				
2		T1	T2	T3	T4	T5	T6			
3			T1	T2	T3	T4	T5	T6		
4				T1	T2	T3	T4	T5	T6	

دیاگرام زمانی برای خط لوله

در یک خط لوله با مشخصات زیر:

تعداد قسمت ها K

زمان پالس ساعت t_P

تعداد تکالیف n $T1,T2,\ldots,T_n$

نسبت تسریع یا ازدیاد سرعت(Speed-up ratio):

$$S=\frac{nt_n}{(n+k-1)t_P}$$

در صورتیکه n>>k-1 :

$$S=\frac{t_n}{t_P}$$

معمولا $t_n<kt_P$

درشرایط ایده آل اگر $t_n=kt_P$:

حداکثر نسبت تسریع:

$$S=\frac{kt_P}{t_P}=k$$

مثال:

در یک خط لوله اگر

t_P=20ns
k=4
n=100

زمان اجرای کل در خط لوله =(k+n-1)t_P= $(4+99)\times 20 = 2060$

اگر فرض شود t_n=kt_P=4×20

$$S = \frac{nkt_p}{(k+n-1)t_P} = \frac{8000}{2060} = 3.88$$

اگر $t_n = 60ns$

$$S = \frac{t_n}{t_p} = \frac{60}{20} = 3$$

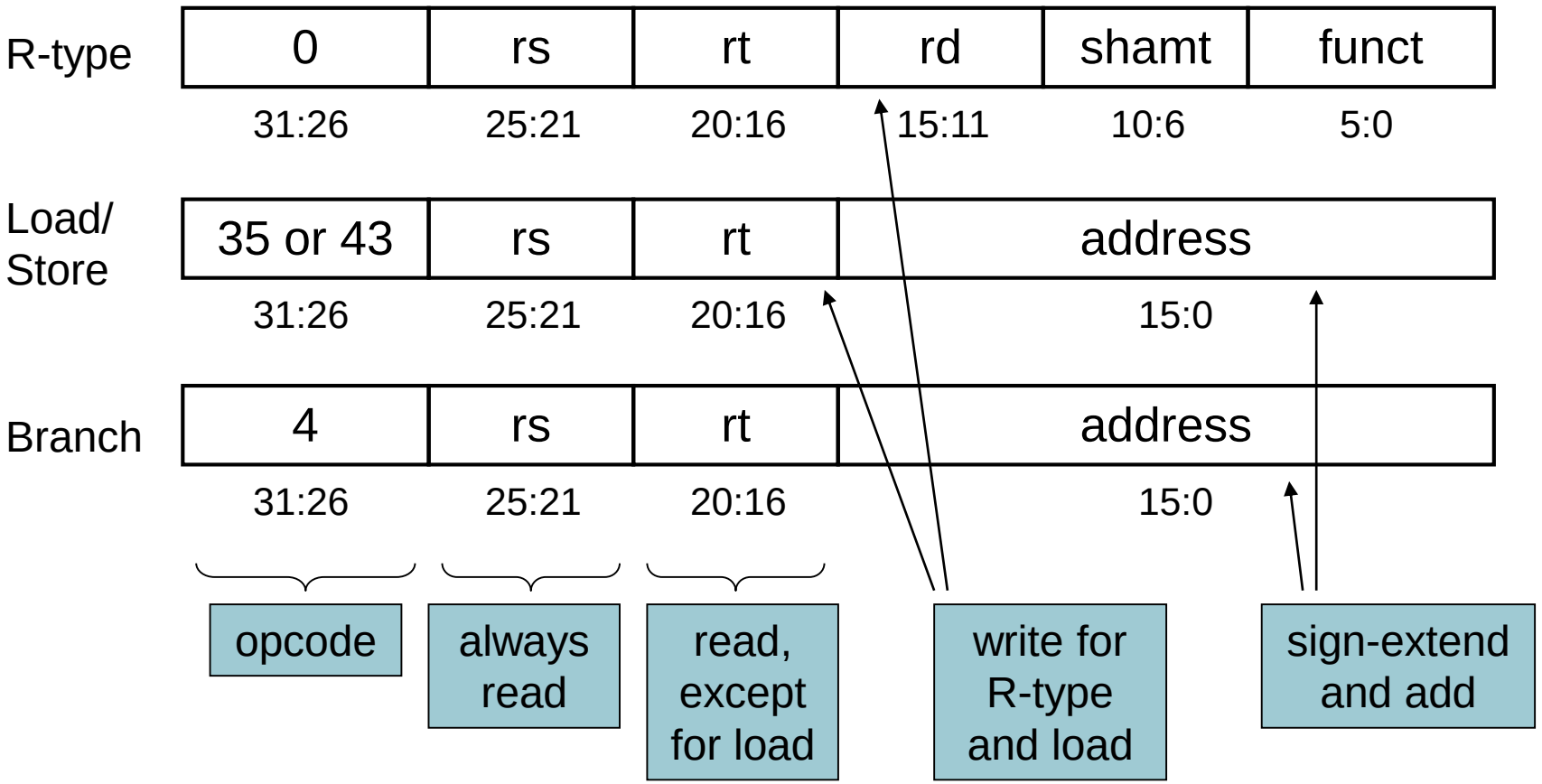
در خط لوله مذکور فرض می کنیم که تاخیر چهارقسمت خط لوله به ترتیب برابر با $t_2 = 70, t_1 = 60$ $t_4 = 80, t_3 = 100$ نانو ثانیه و ثبات های واسط نیز تاخیری برابر $t_r = 10$ نانوثانیه داشته باشند.در این صورت زمان پالس ساعت حداقل برابر $t_p = t_3 + t_r = 110$ نانو ثانیه می شود.

تاخیر مدار غیر خط لوله‌ای مدار برابر خواهد بود با:

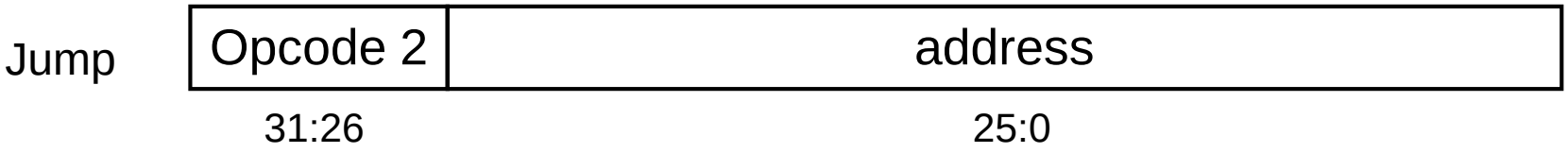
$$t_n = t_1 + t_2 + t_3 + t_4 + t_r = 320$$ نانو ثانیه

که دراین صورت مدار خط لوله دارای ازدیاد سرعت 320/110=2.9 برابر مدارغیر خط لوله ای خواهد بود.

Instruction Formats for the MIPS CPU



Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00 as the 2 lower bits

MIPS Instruction Pipeline

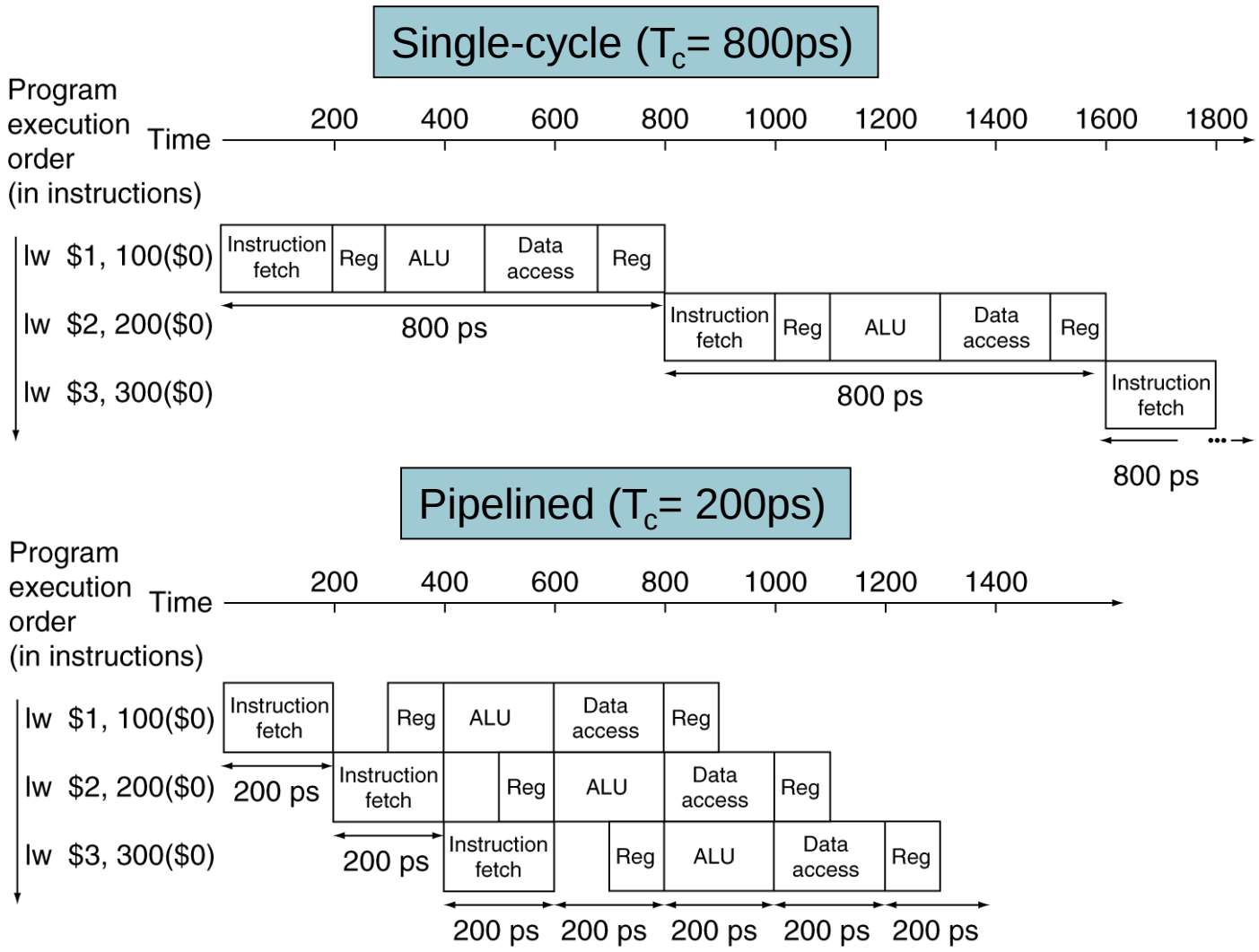
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance



Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions pipelined =
  - $$\frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}$$
  - Tp= Tn/k ==➔ Tp= 1000/5=200ps S=Tn/Tp=5
- If not balanced, speedup is less
  - S=Tn/Tp=4
- Speedup is due to increased throughput
  - Latency (time for each instruction) does not decrease

Pipelining and ISA Design

- MIPS ISA (Instruction Set Architecture) is designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - But for x86 family: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle

Hazards

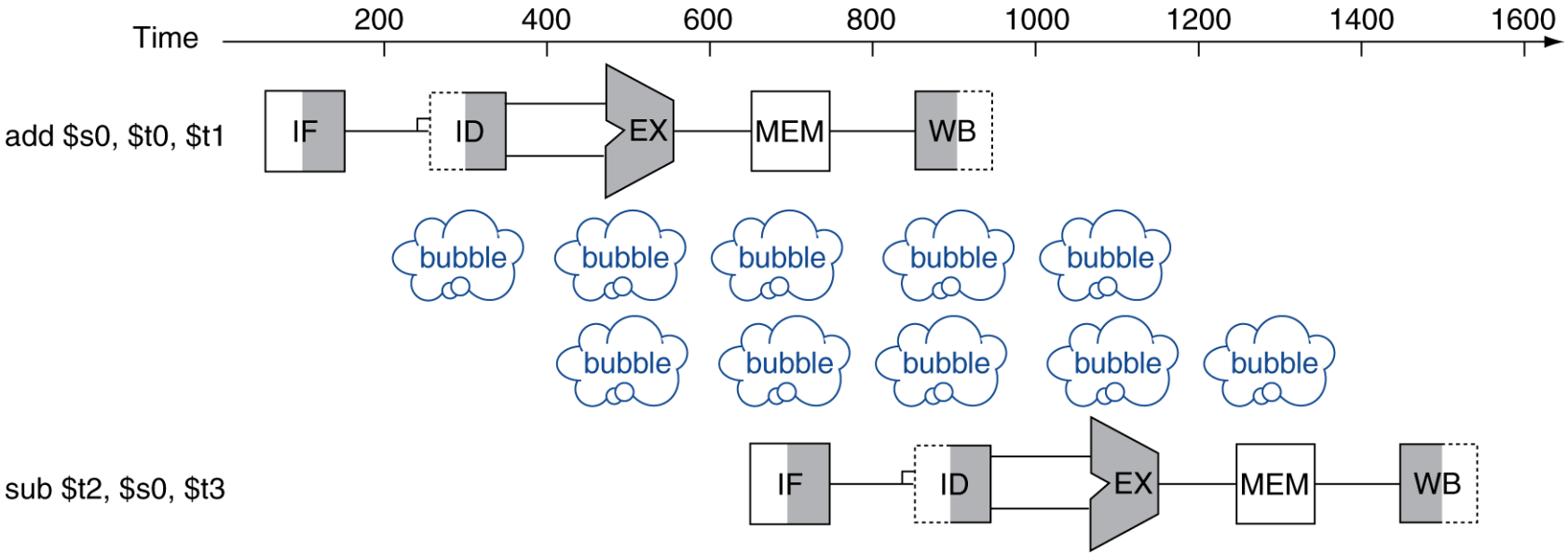
- Situations that prevent starting the next instruction in the next cycle
- **Structural hazards**
  - A required resource is busy
- **Data hazard**
  - Need to wait for previous instruction to complete its data read/write
- **Control hazard**
  - Deciding on branch action depends on the conditional branch result

Structural Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches

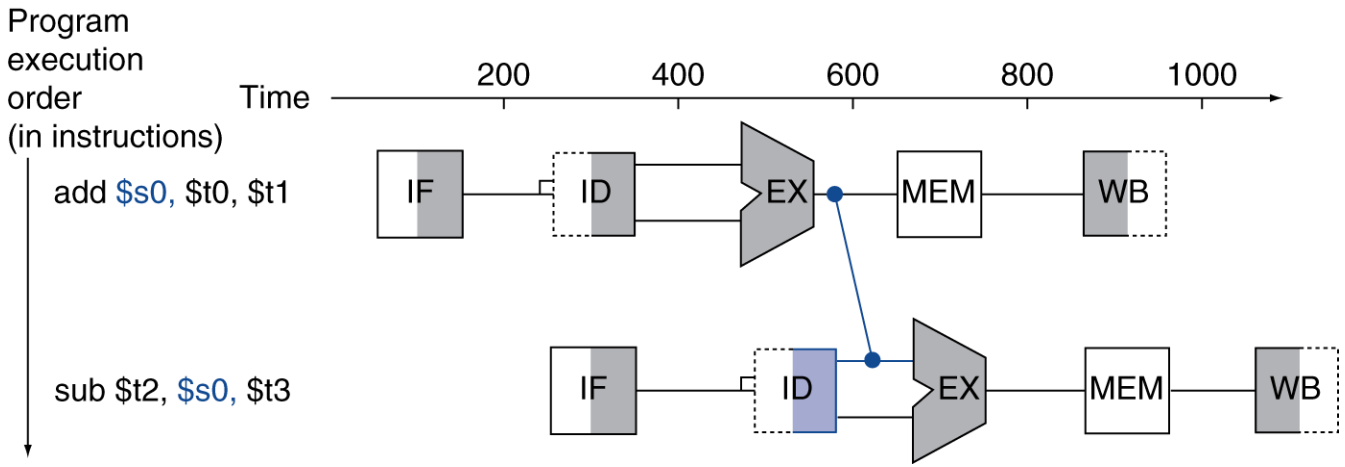
Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



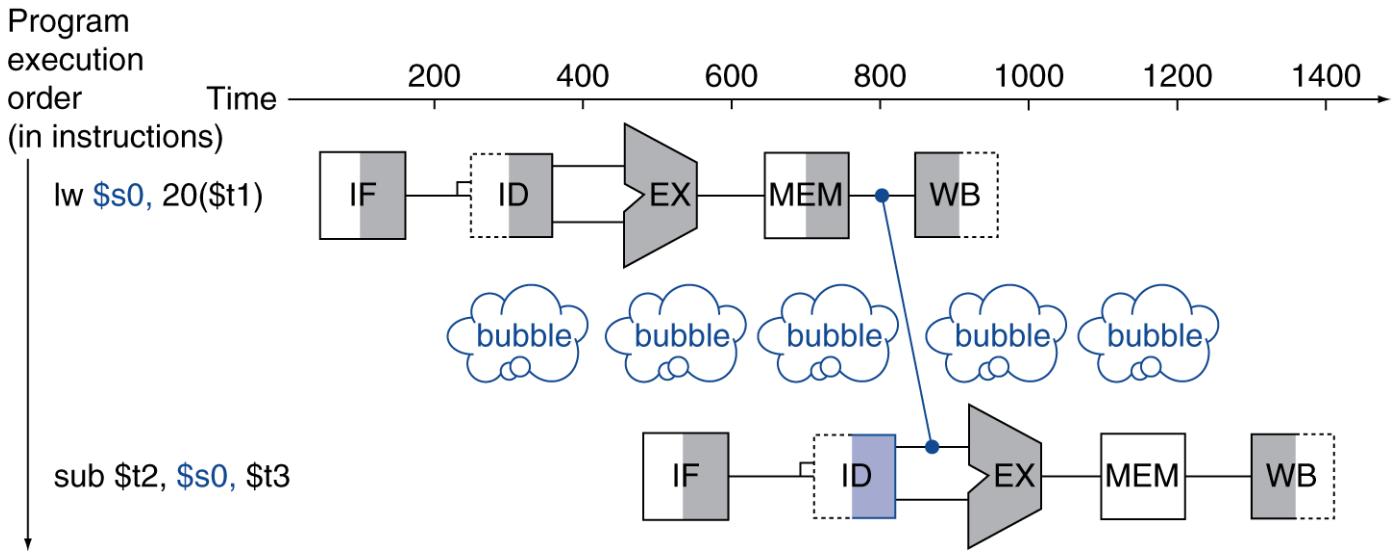
Forwarding (aka (also known as) Bypassing)

- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires extra connections in the datapath



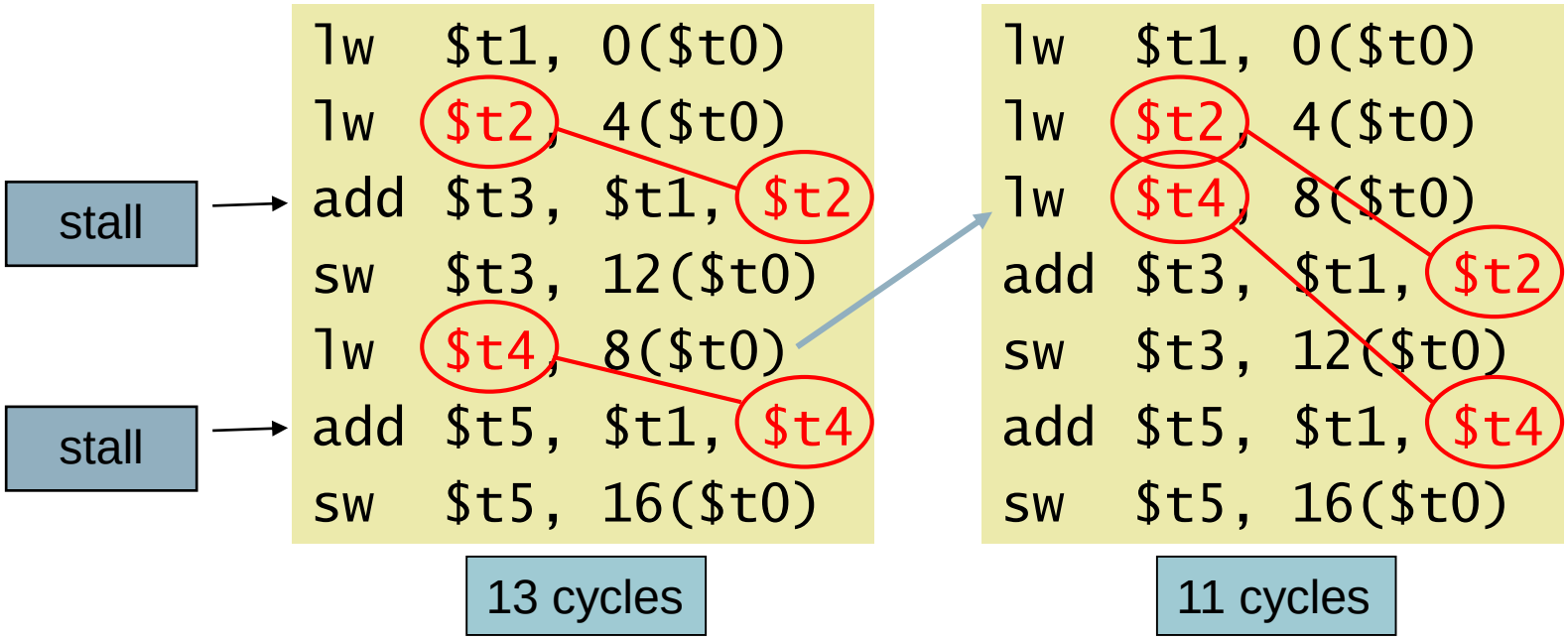
Load-Use Data Hazard

- Can’t always avoid stalls by forwarding
  - If value not computed when needed
  - Can’t forward backward in time!



Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`

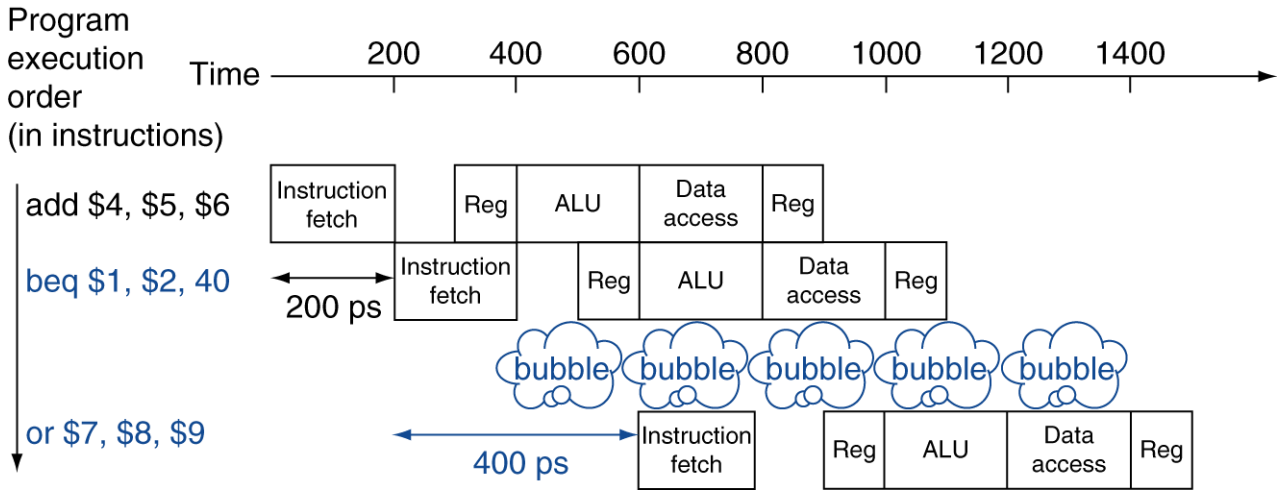


Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can’t always fetch correct instruction
    - Still working on ID stage of branch
- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

Stall on Branch

- Wait until branch outcome determined before fetching next instruction

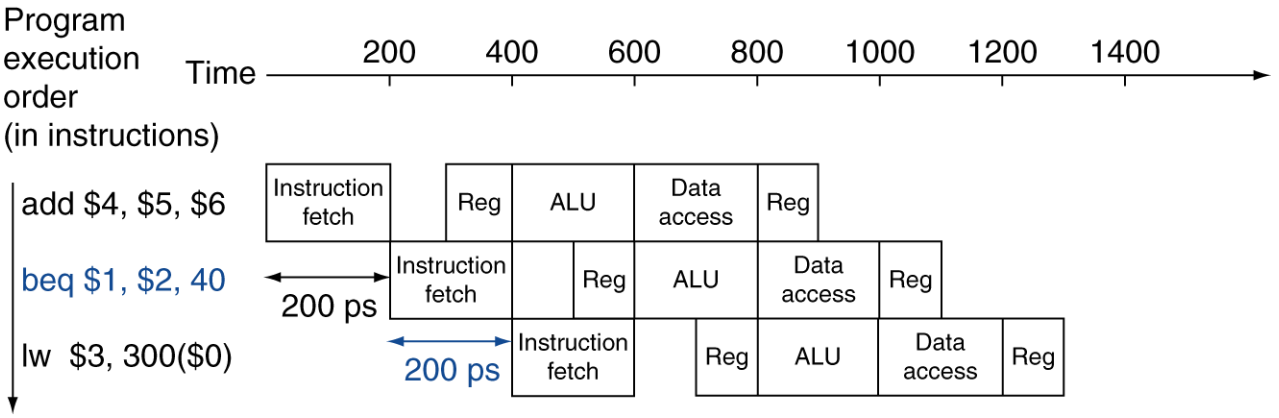


Branch Prediction

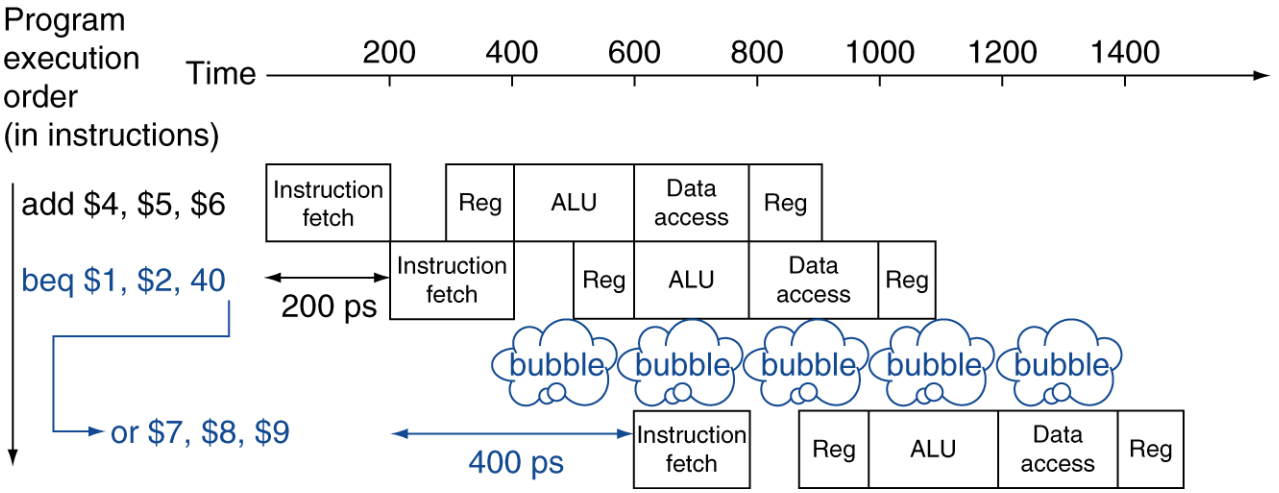
- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

MIPS with Predict Not Taken

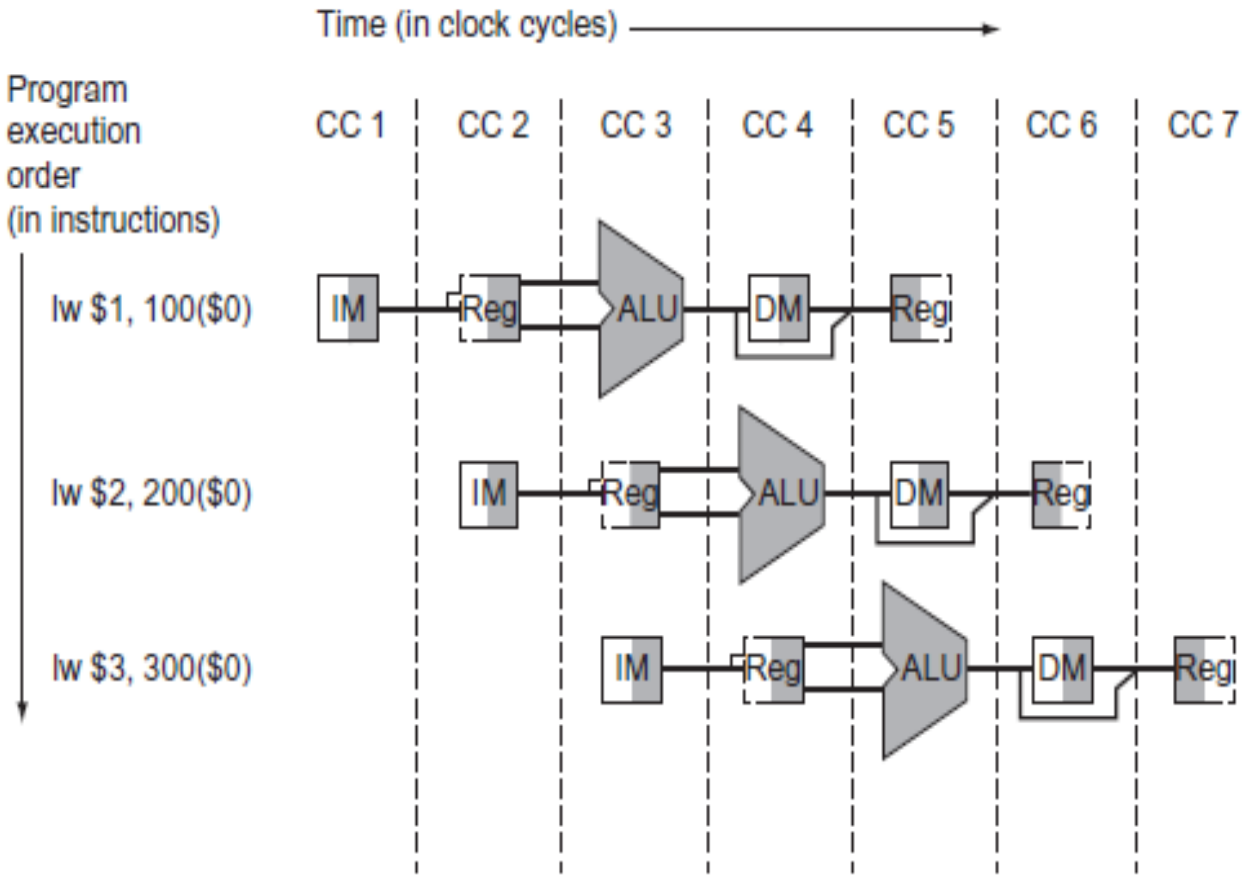
Prediction correct



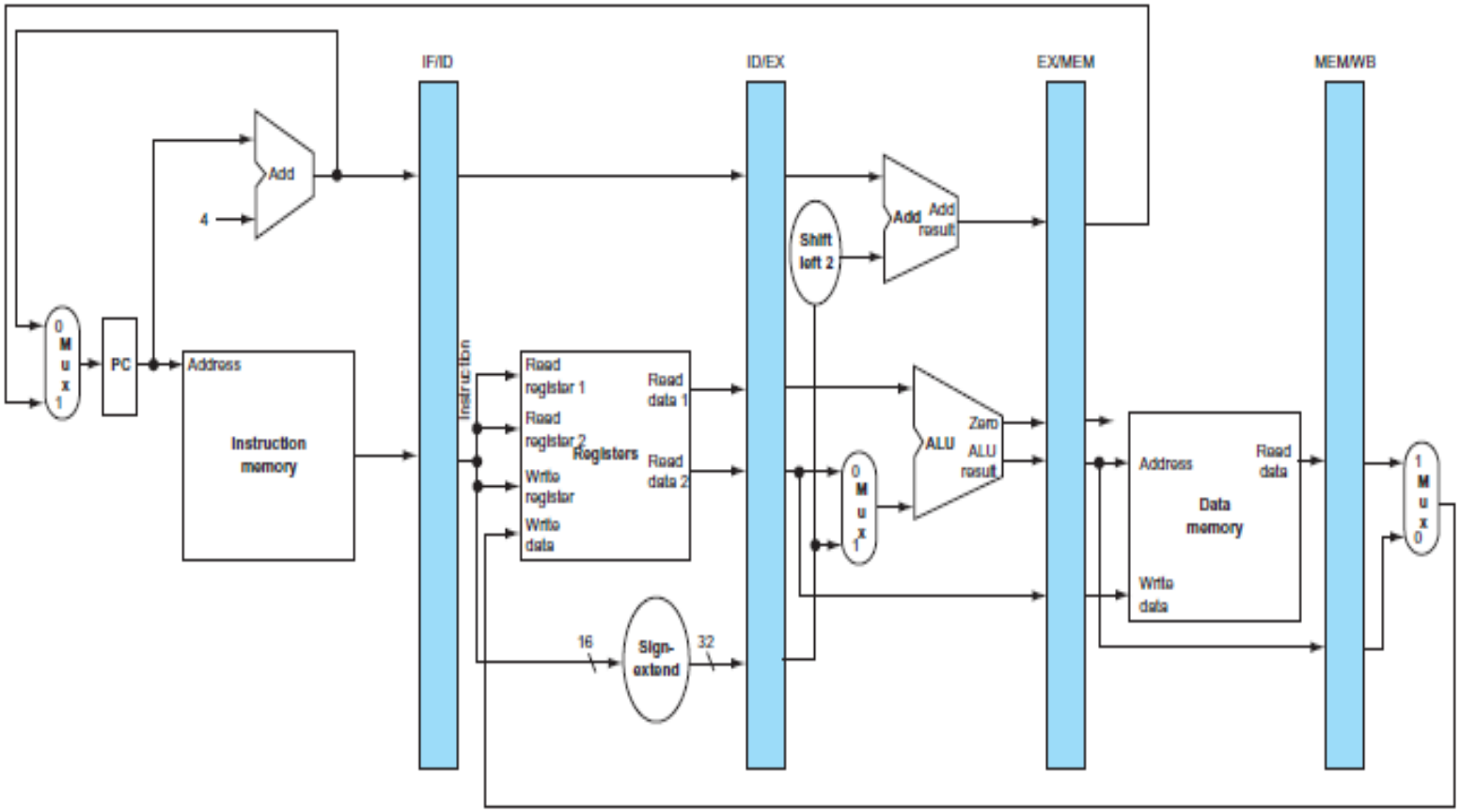
Prediction incorrect



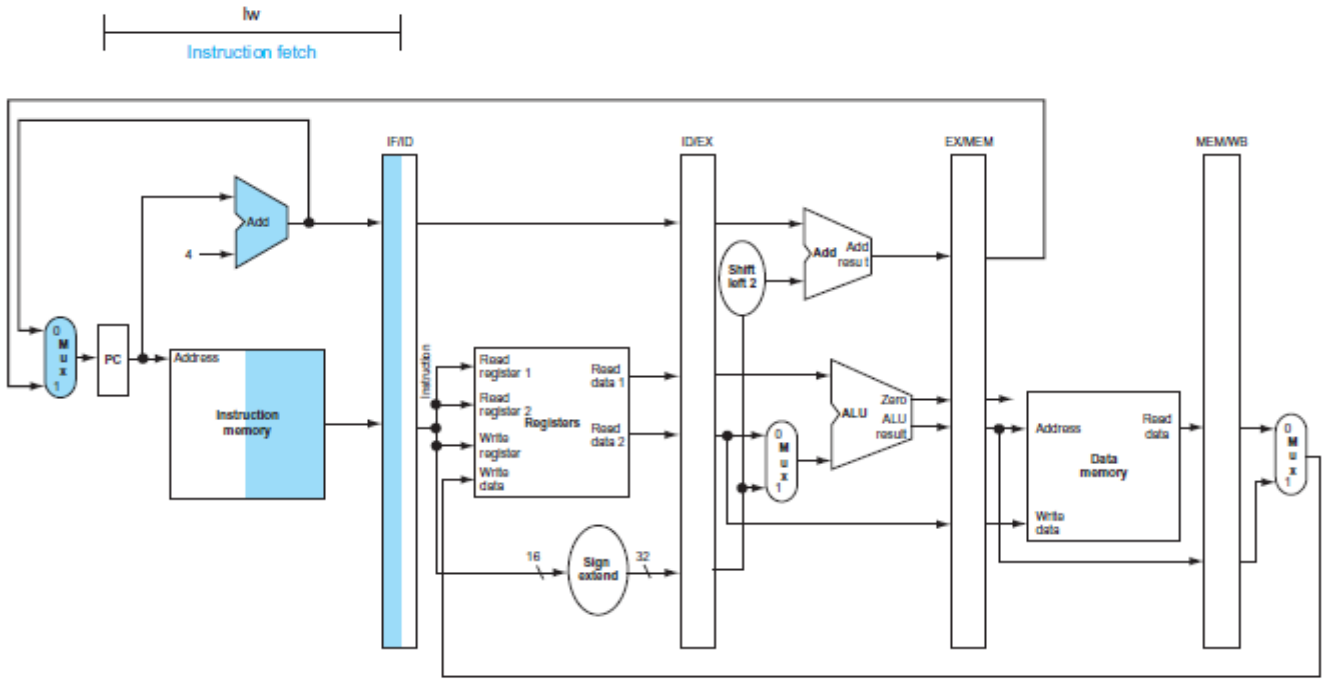
ورود 3 دستور متوالی به قسمت های مختلف خط لوله و اجرای آنها در پالسهای ساعت 1 تا 7(شکل 34 کتاب)



افزودن ثباتها بین قسمتهای خط لوله برای نگهداری نتایج قسمت قبل و تحویل به قسمت بعد (شکل 35).
این ثباتها بسته به اینکه چه اطلاعاتی را نگهداری می کنند اندازه متفاوتی دارند.

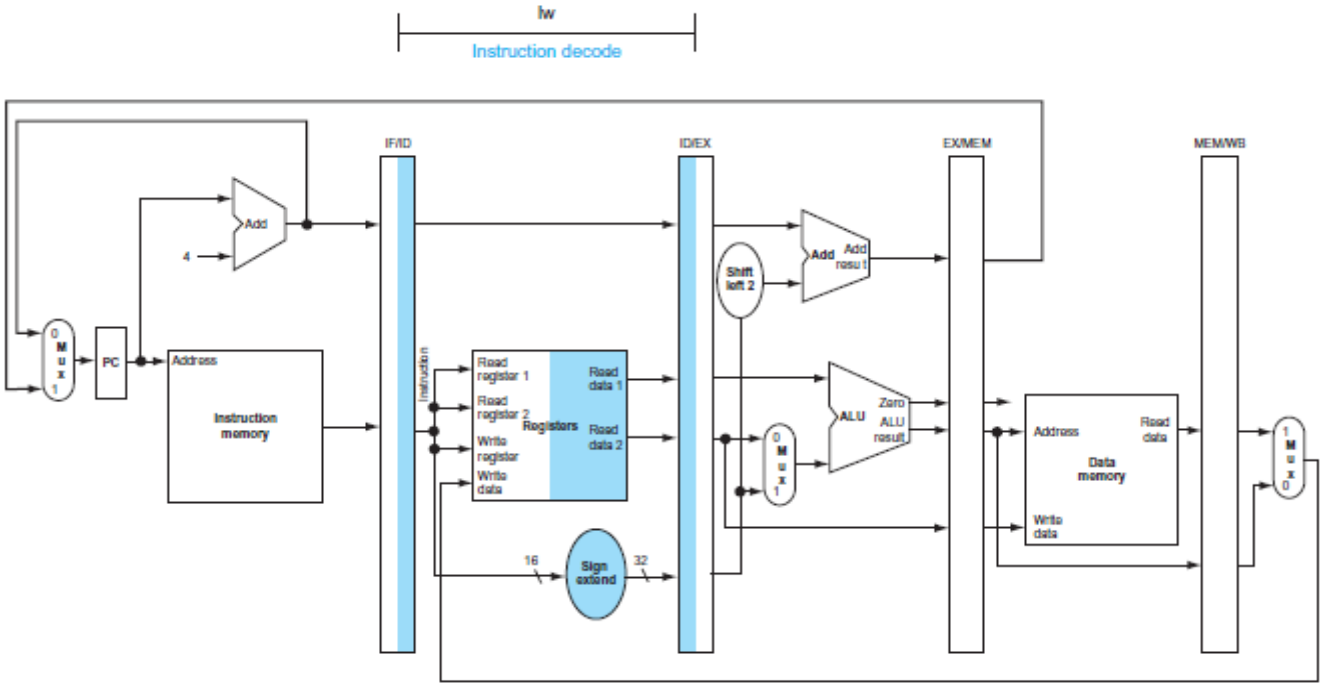


وظیفه قسمت اول خط لوله فراخوانی دستورالعمل از حافظه نهان دستورالعمل است(شکل 36- بالا)



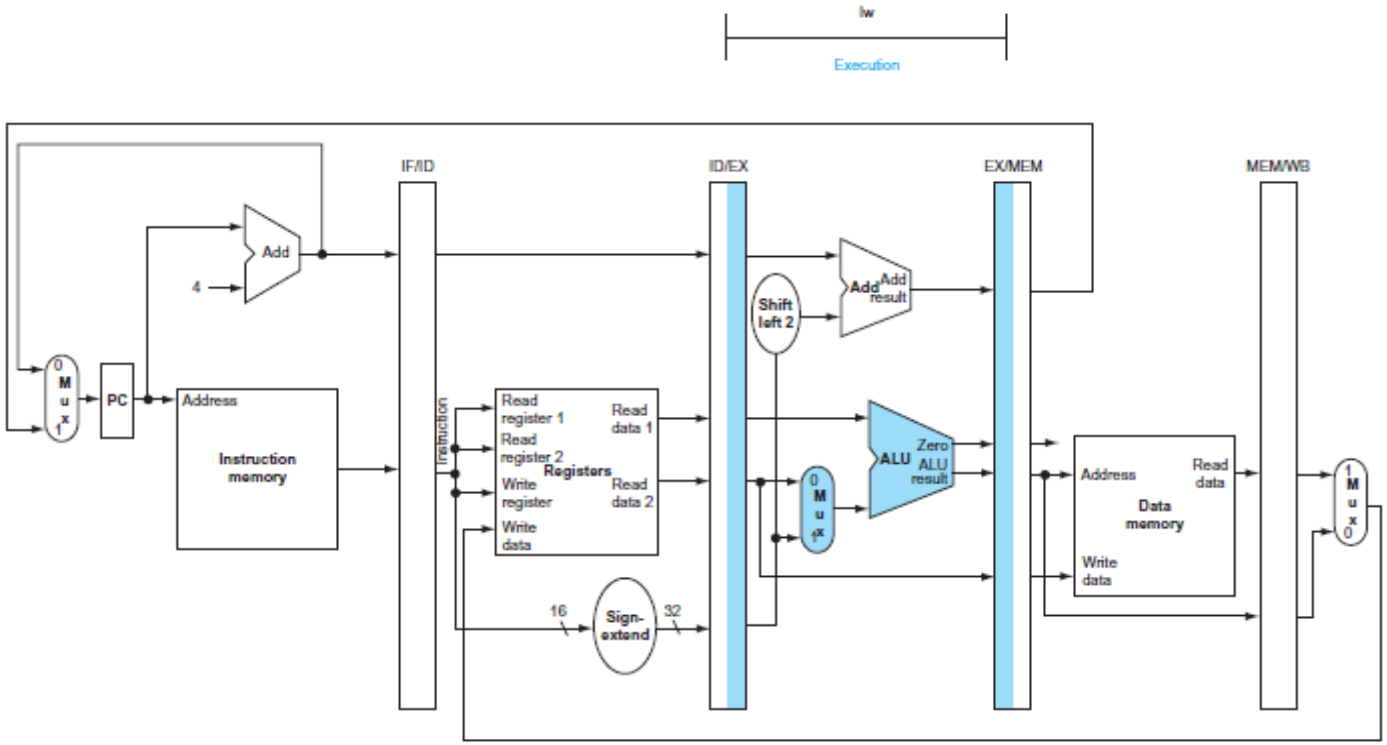
1. *Instruction fetch:* The top portion of Figure 4.36 shows the instruction being read from memory using the address in the PC and then being placed in the IF/ID pipeline register. The PC address is incremented by 4 and then written back into the PC to be ready for the next clock cycle. This incremented address is also saved in the IF/ID pipeline register in case it is needed later for an instruction, such as `beq`. The computer cannot know which type of instruction is being fetched, so it must prepare for any instruction, passing potentially needed information down the pipeline.

وظیفه قسمت دوم خط لوله دیکود دستورالعمل و خواندن عملوند هاست است(شکل 36 - پایین)



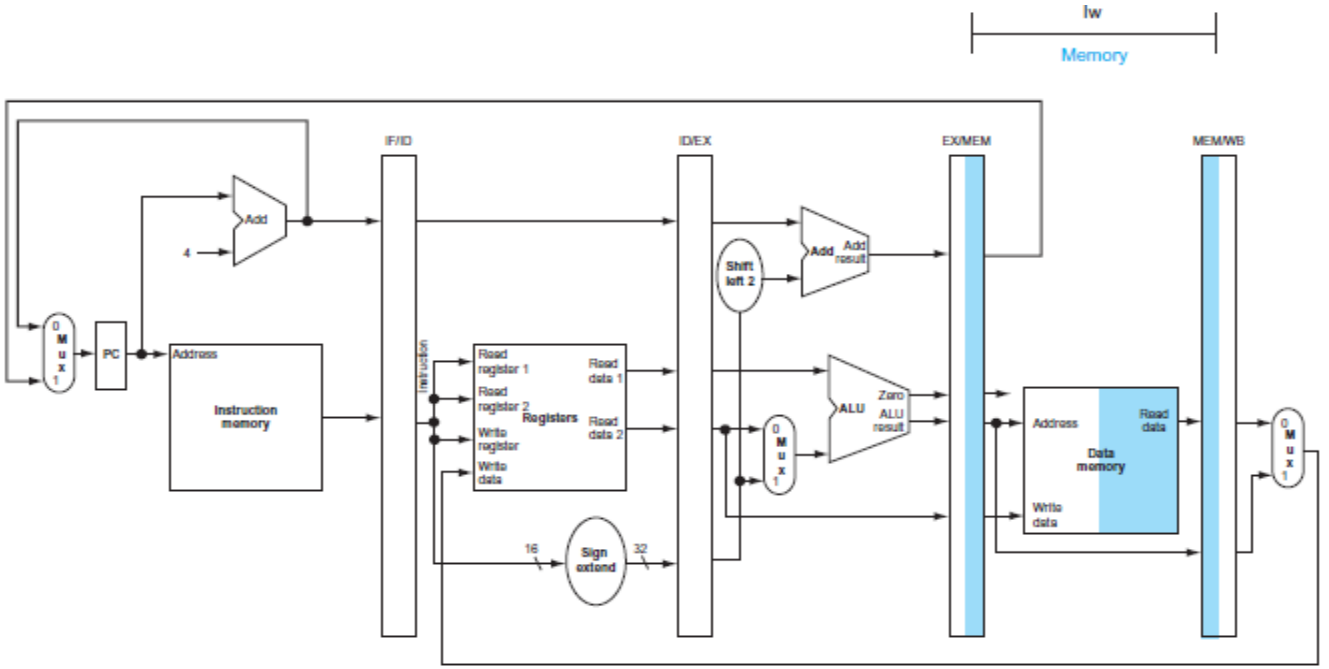
2. *Instruction decode and register file read:* The bottom portion of Figure 4.36 shows the instruction portion of the IF/ID pipeline register supplying the 16-bit immediate field, which is sign-extended to 32 bits, and the register numbers to read the two registers. All three values are stored in the ID/EX pipeline register, along with the incremented PC address. We again transfer everything that might be needed by any instruction during a later clock cycle.

عملکرد قسمت سوم خط لوله برای دستور بارکردن، محاسبه آدرس موثر می باشد (شکل 37)



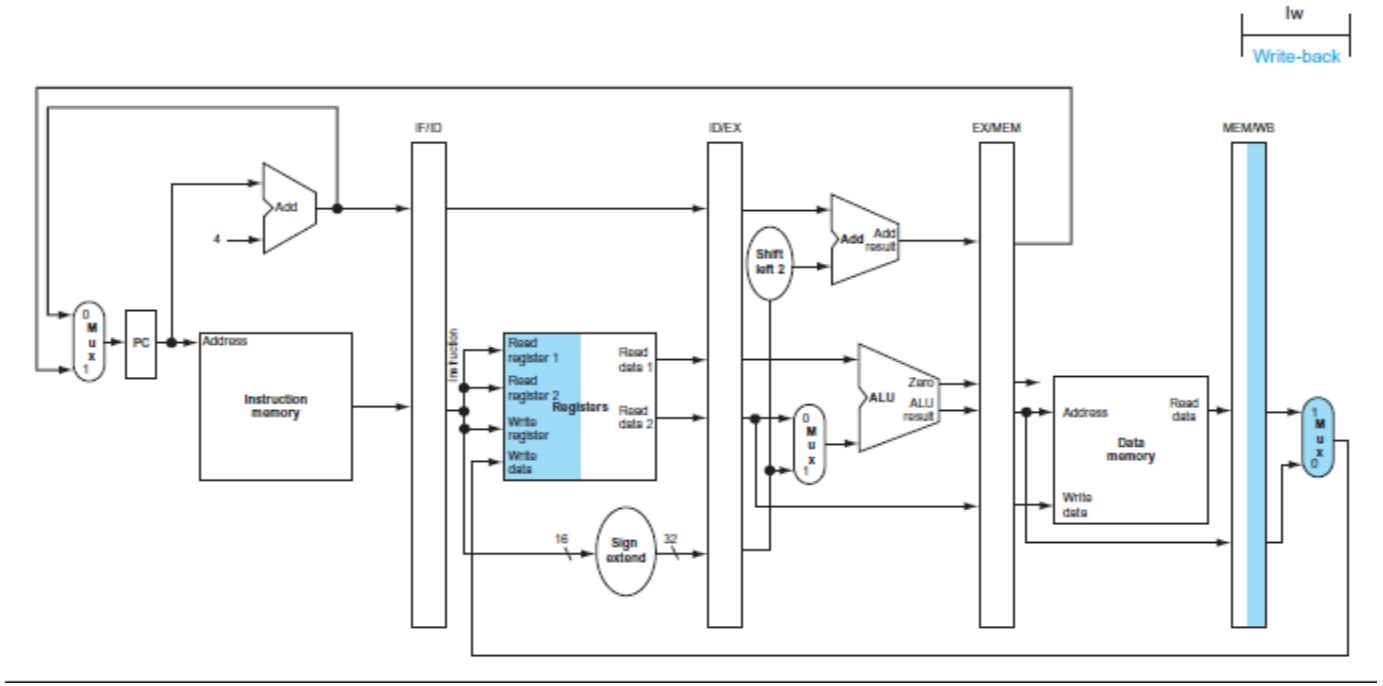
3. *Execute or address calculation:* Figure 4.37 shows that the load instruction reads the contents of register 1 and the sign-extended immediate from the ID/EX pipeline register and adds them using the ALU. That sum is placed in the EX/MEM pipeline register.

عملکرد قسمت چهارم خط لوله برای دستور بارکردن، خواندن داده از حافظه نهان داده است (شکل 38 - بالا).



4. *Memory access*: The top portion of Figure 4.38 shows the load instruction reading the data memory using the address from the EX/MEM pipeline register and loading the data into the MEM/WB pipeline register.

عملکرد قسمت پنجم خط لوله برای دستور بارکردن، نوشتن داده در ثبات مقصد می باشد (شکل 38 - پایین)



5. *Write-back*: The bottom portion of Figure 4.38 shows the final step: reading the data from the MEM/WB pipeline register and writing it into the register file in the middle of the figure.

عملکرد قسمت سوم خط لوله برای دستور ذخیره سازی (محاسبه آدرس موثر)

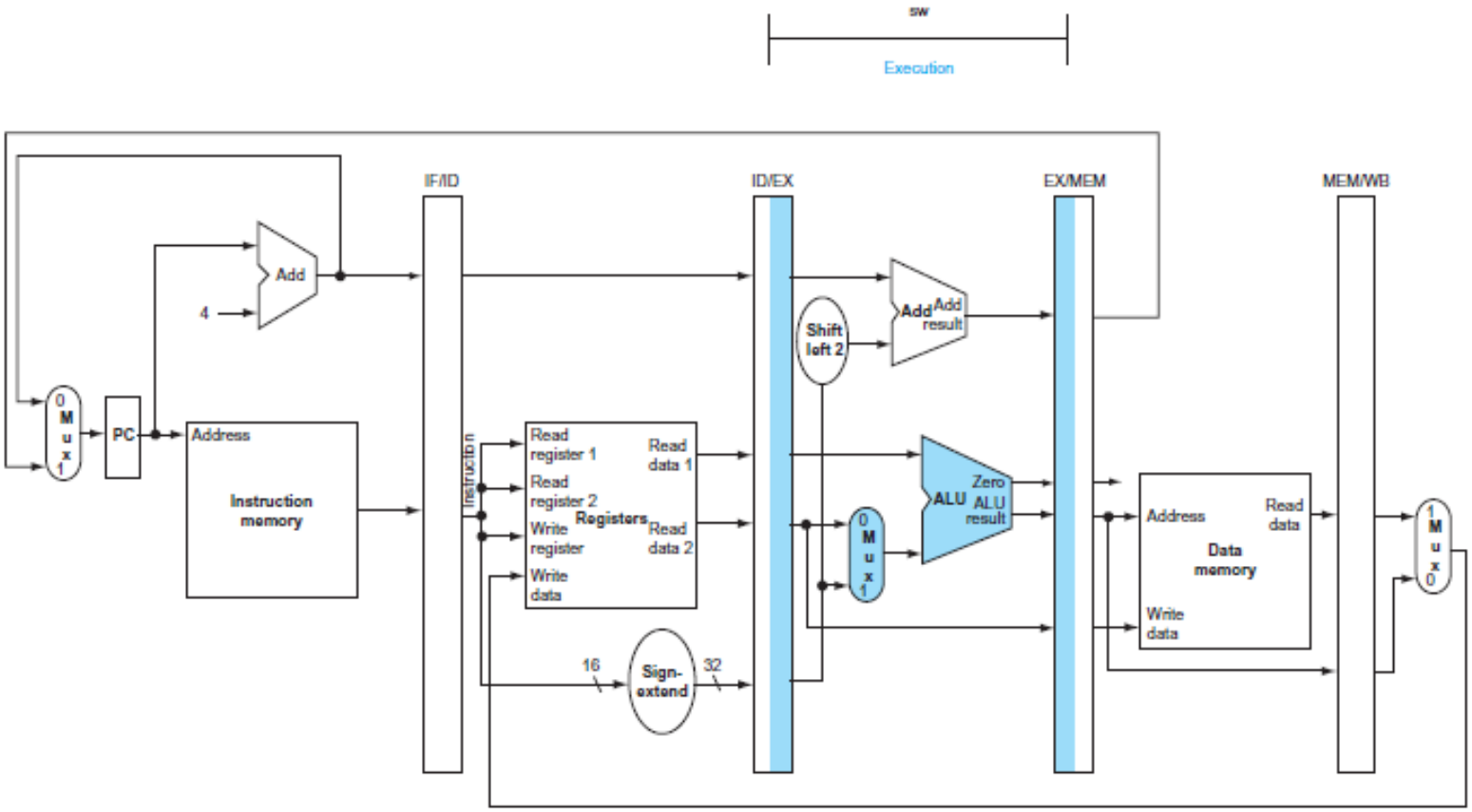


FIGURE 4.39 EX: The third pipe stage of a store instruction. Unlike the third stage of the load instruction in Figure 4.37, the second register value is loaded into the EX/MEM pipeline register to be used in the next stage. Although it wouldn't hurt to always write this second register into the EX/MEM pipeline register, we write the second register only on a store instruction to make the pipeline easier to understand.

عملکرد قسمت چهارم خط لوله برای دستور ذخیره سازی. در این قسمت کار دستور پایان می یابد. در اسلاید بعدی ملاحظه می شود که قسمت پنجم خط لوله برای دستور ذخیره سازی کاری انجام نمی دهد.

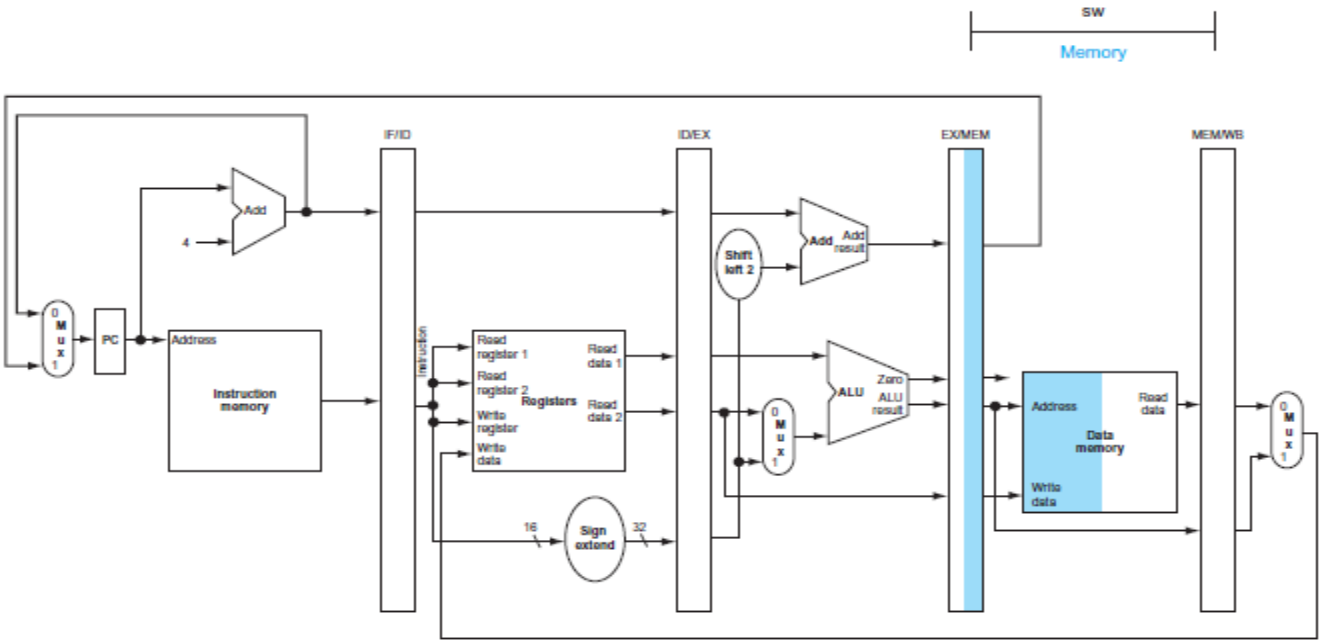
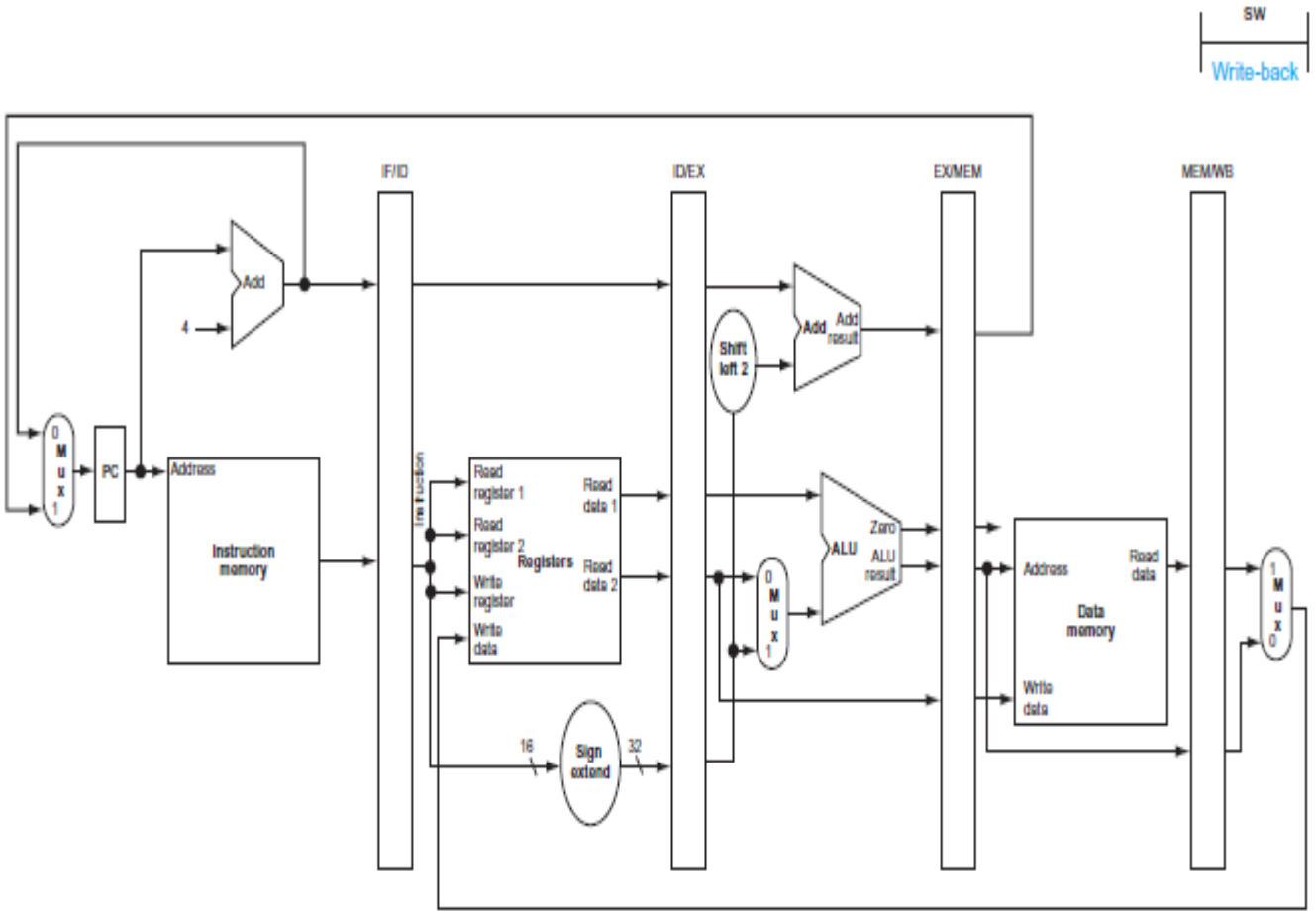


FIGURE 4.40 MEM and WB: The fourth and fifth pipe stages of a store instruction. In the fourth stage, the data is written into data memory for the store. Note that the data comes from the EX/MEM pipeline register and that nothing is changed in the MEM/WB pipeline register. Once the data is written in memory, there is nothing left for the store instruction to do, so nothing happens in stage 5.

قسمت پنجم خط لوله برای دستور ذخیره سازی کاری انجام نمی دهد.



شکل اصلاح شده مسیر داده در خط لوله برای دستور بار کردن؛ آدرس ثبات مقصد، پس از مشخص شدن در قسمت دوم خط لوله جلو می آید و در قسمت پنجم برای نوشتن استفاده می شود.

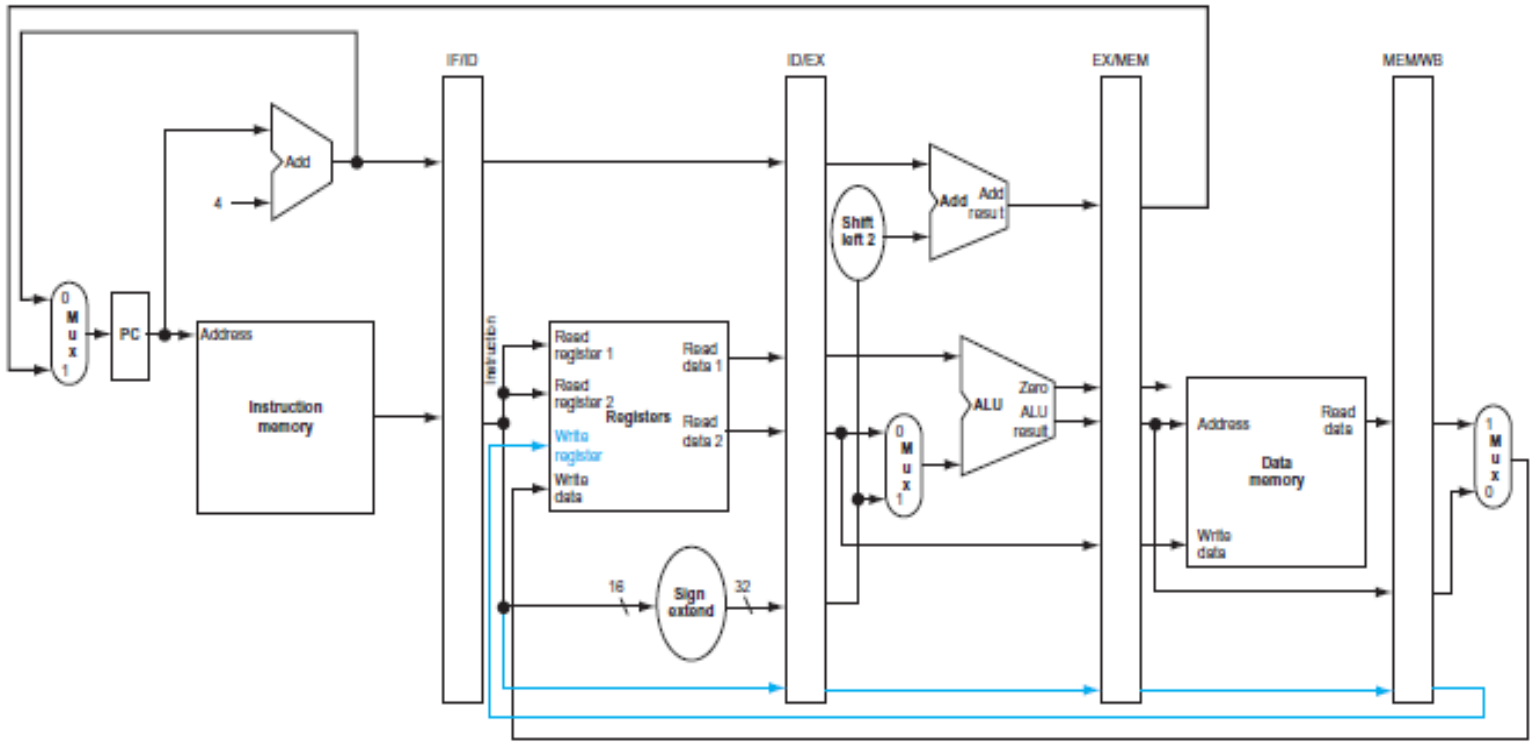


FIGURE 4.41 The corrected pipelined datapath to handle the load instruction properly. The write register number now comes from the MEM/WB pipeline register along with the data. The register number is passed from the ID pipe stage until it reaches the MEM/WB pipeline register, adding five more bits to the last three pipeline registers. This new path is shown in color.

در این شکل تمام اجزای مسیر داده که در 5 قسمت خط لوله و در 5 پالس ساعت متوالی برای دستور بار کردن استفاده می شوند نشان داده شده اند.

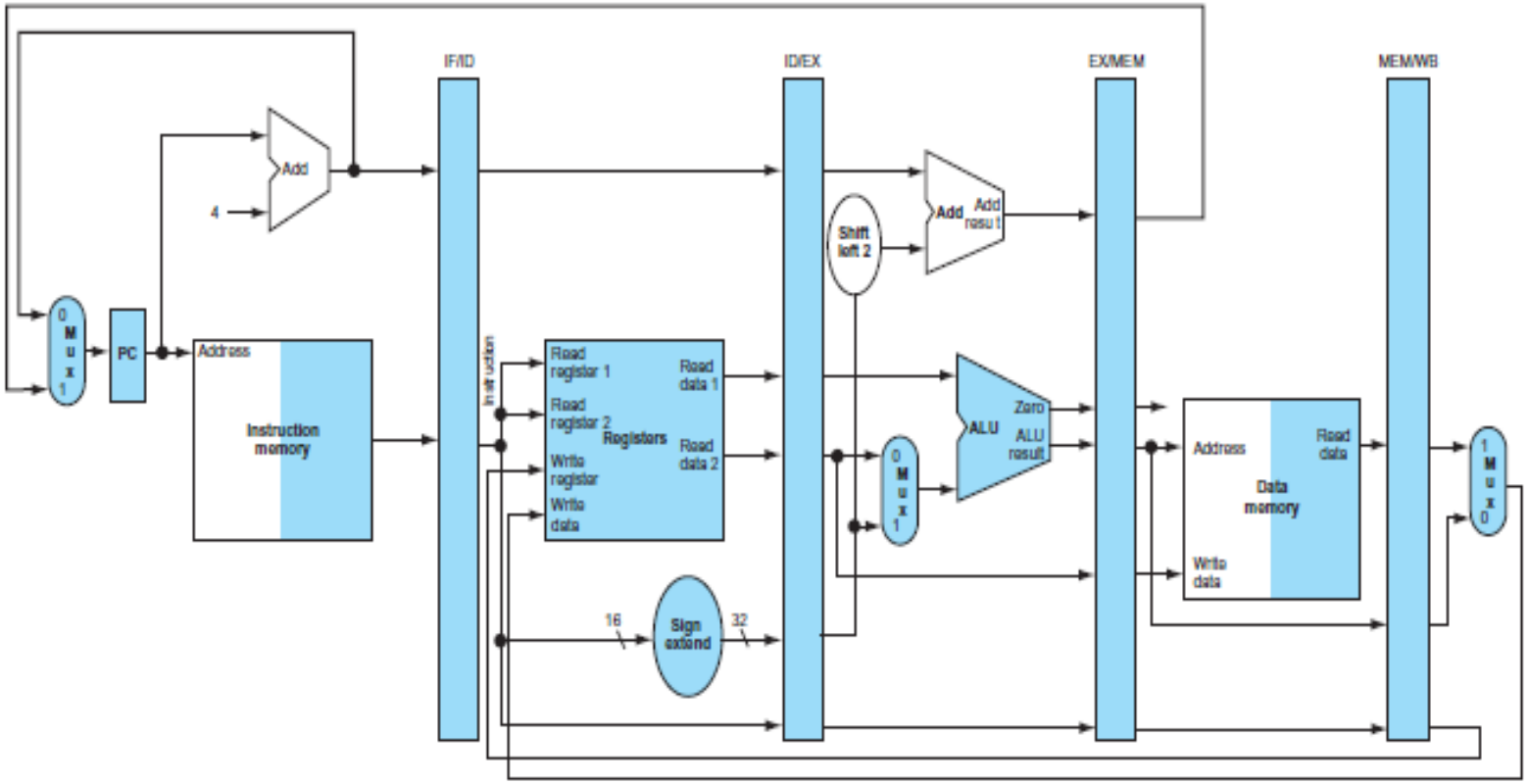


FIGURE 4.42 The portion of the datapath in Figure 4.41 that is used in all five stages of a load instruction.

نحوه نمایش خط لوله در چند پالس ساعت متوالی برای 5 دستورالعمل نمونه

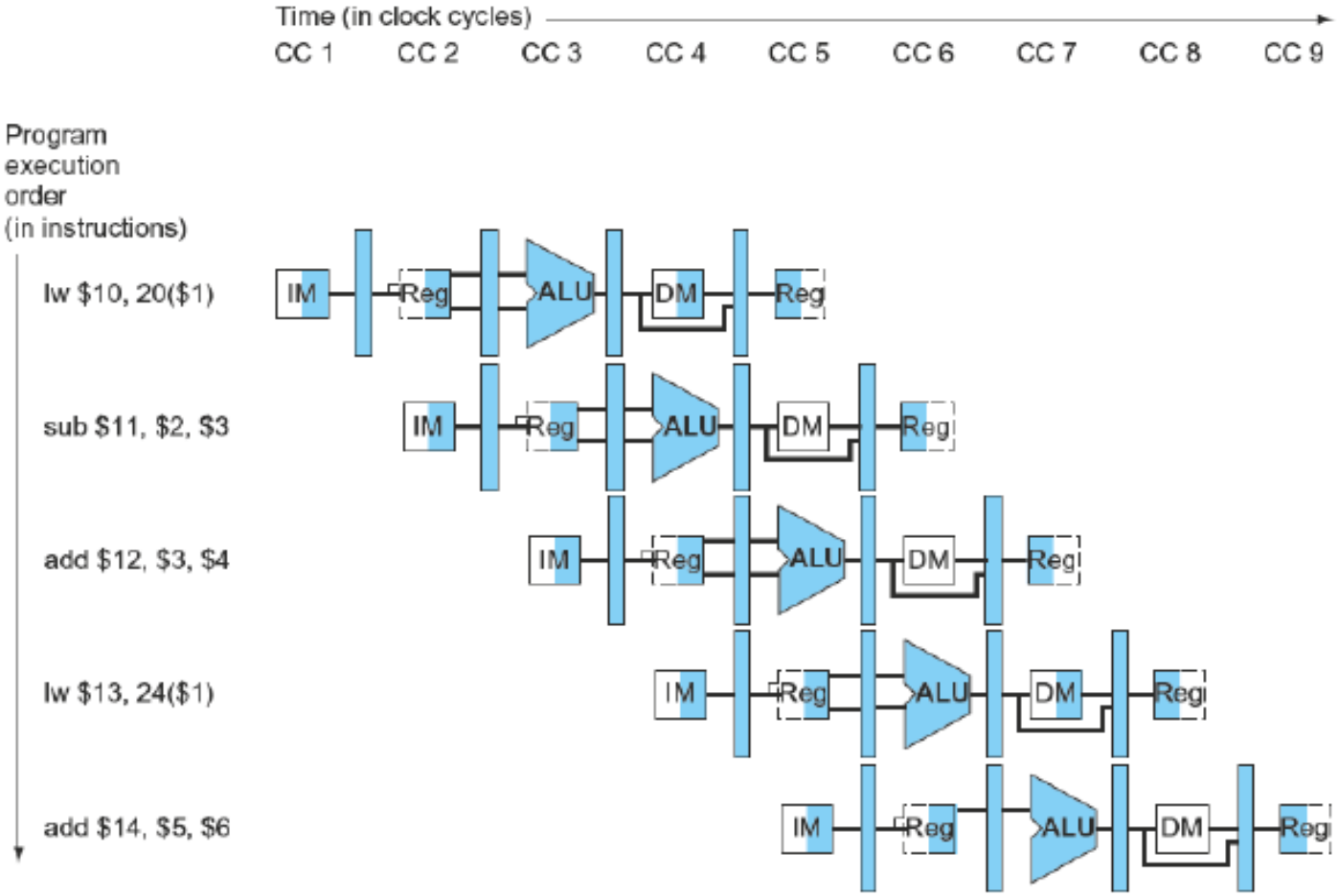


FIGURE 4.43 Multiple-clock-cycle pipeline diagram of five instructions. This style of pipeline representation shows the complete execution of instructions in a single figure. Instructions are listed in instruction execution order from top to bottom, and clock cycles move from left to right. Unlike Figure 4.28, here we show the pipeline registers between each stage. Figure 4.44 shows the traditional way to draw this diagram.

نوع دیگری از نمایش چند پالس ساعتی خط لوله اسلاید قبل

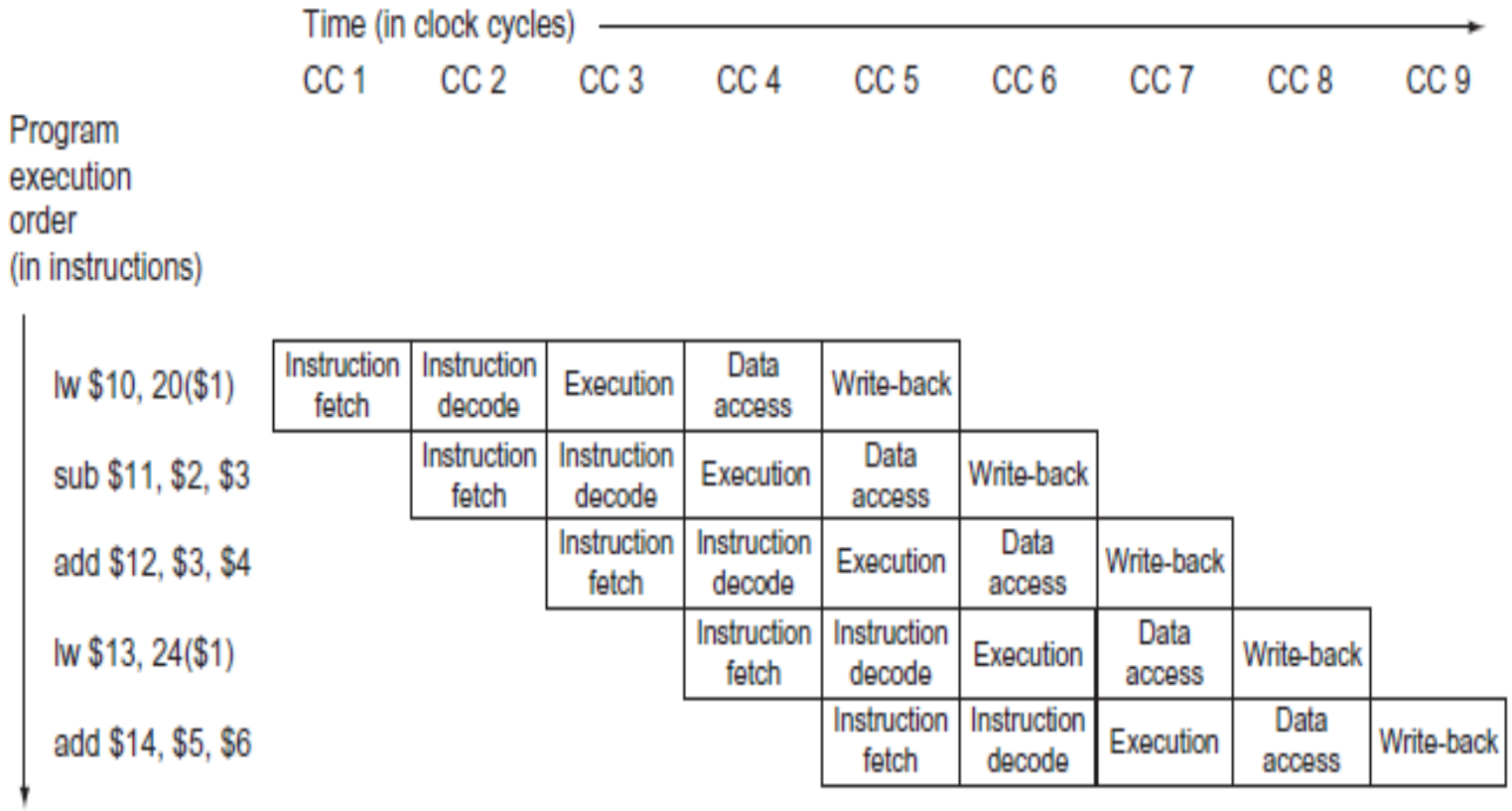


FIGURE 4.44 Traditional multiple-clock-cycle pipeline diagram of five instructions in Figure 4.43.

نحوه دیگری از نمایش خط لوله؛ در این حالت عملکرد قسمتهای مختلف خط لوله در یک پالس ساعت مشخص (مثلا پالس پنجم در شکلهای 43 و 44) نمایش داده می شود. مشاهده می شود که هر قسمت خط لوله مشغول کار روی یک دستور متفاوت است.

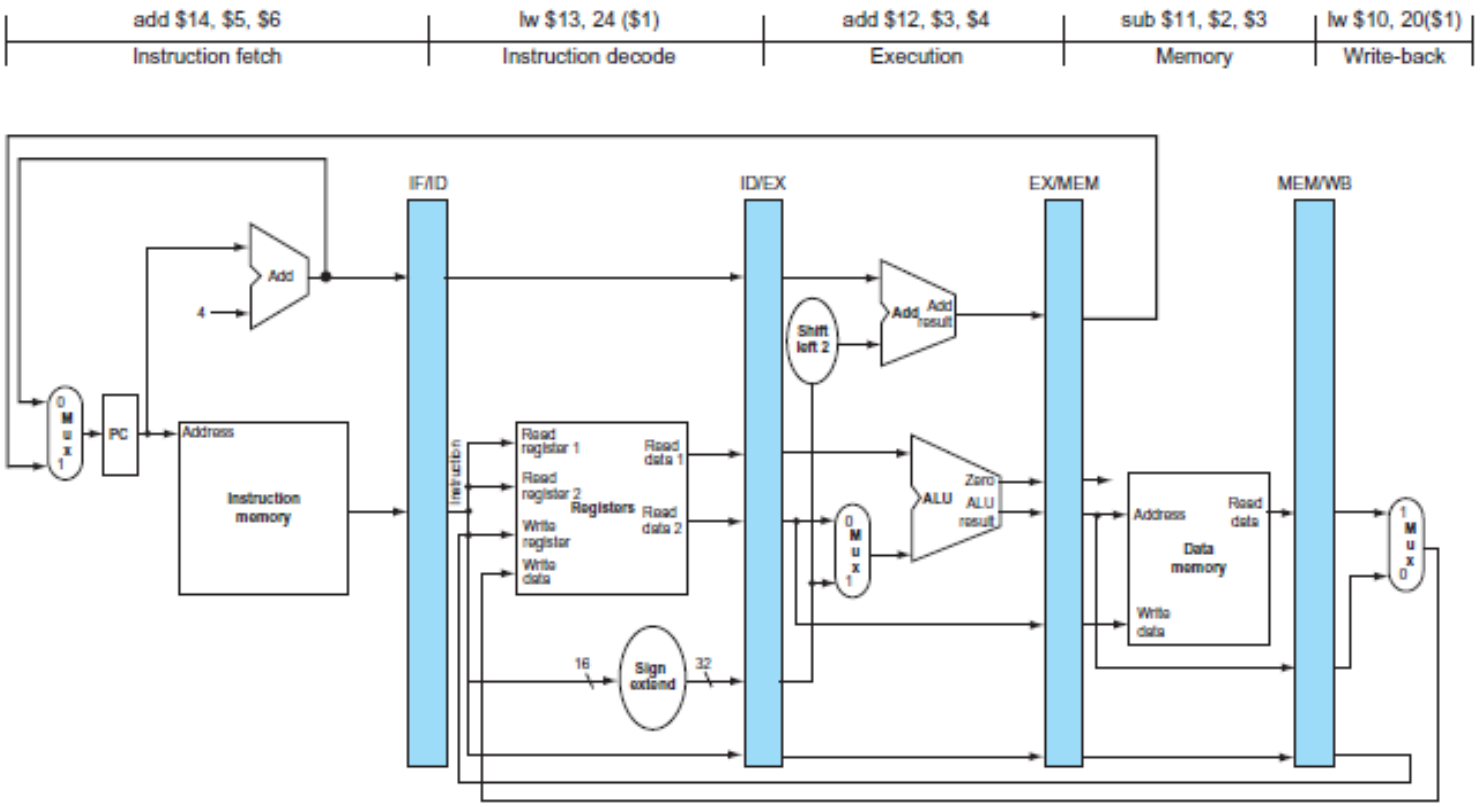


FIGURE 4.45 The single-clock-cycle diagram corresponding to clock cycle 5 of the pipeline in Figures 4.43 and 4.44. As you can see, a single-clock-cycle figure is a vertical slice through a multiple-clock-cycle diagram.

در این شکل سیگنالهای کنترلی که متناسب با عملکرد دستورات مختلف به اجزای خط لوله باید اعمال شوند نشان داده شده اند. سیگنالهای کنترلی قسمتهای فراخوانی و دیکود دستورالعمل که برای همه دستورات یکسان هستند نشان داده نشده اند.

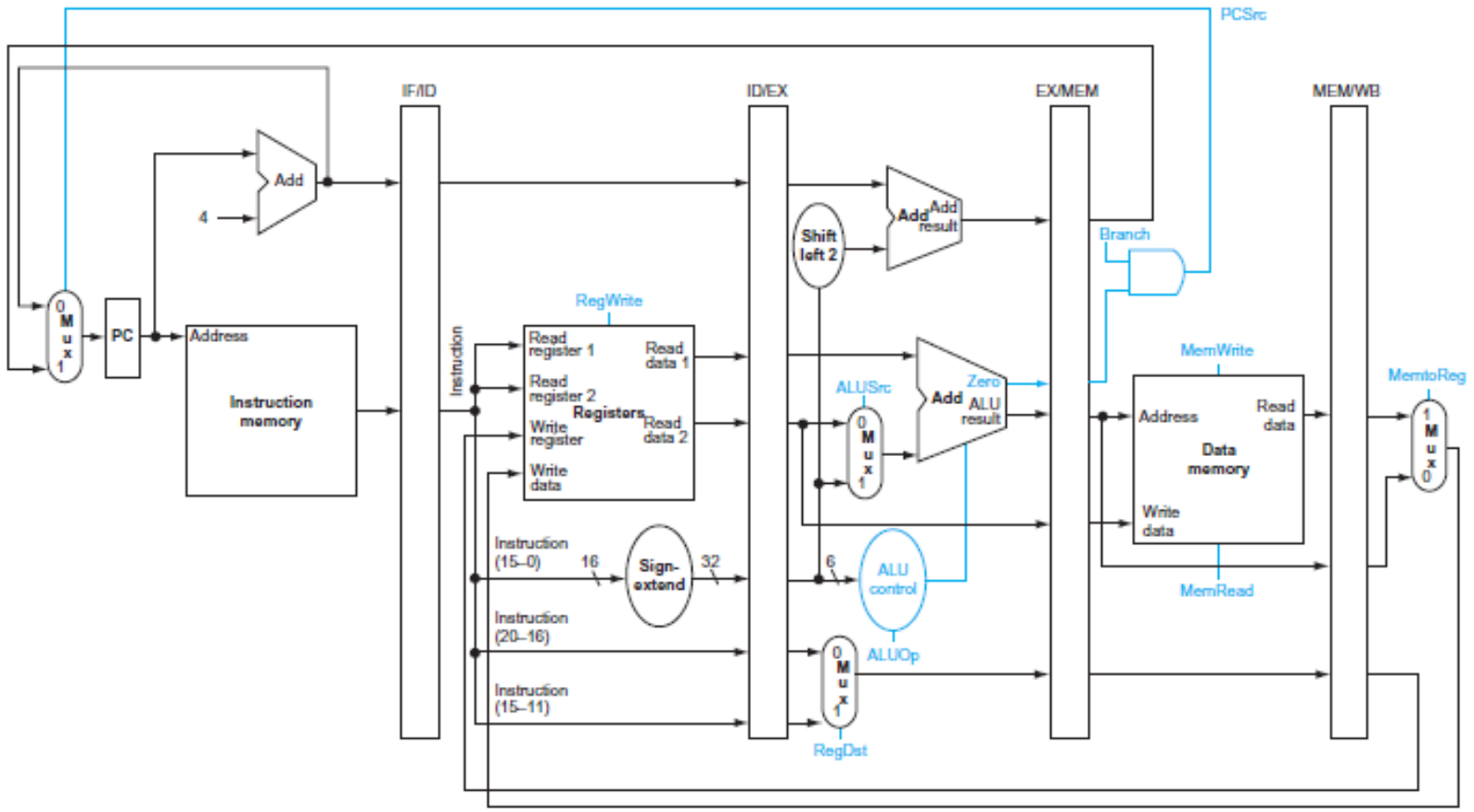


FIGURE 4.46 The pipelined datapath of Figure 4.41 with the control signals identified. This datapath borrows the control logic for PC source, register destination number, and ALU control from Section 4.4. Note that we now need the 6-bit funct field (function code) of the instruction in the EX stage as input to ALU control, so these bits must also be included in the ID/EX pipeline register. Recall that these 6 bits are also the 6 least significant bits of the immediate field in the instruction, so the ID/EX pipeline register can supply them from the immediate field since sign extension leaves these bits unchanged.

نحوه تولید سیگنالهای کنترل ALU

- سیگناهای کنترلی ALU توسط واحد ALU Control ساخته می شوند.
- ورودی های این واحد 6 بیت function هستند که از دستورالعمل گرفته می شوند و دو بیت دیگر بنام ALU Operation توسط واحد کنترل ساخته می شوند. شکل صفحه بعد نحوه ساختن 4 بیت کنترلی واحد حساب و منطق را نشان می دهد.

نحوه تولید سیگنالهای کنترل ALU

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

FIGURE 4.47 A copy of Figure 4.12. This figure shows how the ALU control bits are set depending on the ALUOp control bits and the different function codes for the R-type instruction.

بیتهای کنترلی مربوط به سه قسمت آخر خط لوله:

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

FIGURE 4.49 The values of the control lines are the same as in Figure 4.18, but they have been shuffled into three groups corresponding to the last three pipeline stages.

واحد کنترل 6 بیت کد عمل را از دستورالعمل می گیرد و سه دسته سیگنالهای کنترلی را برای 3 قسمت آخر خط لوله می سازد. این بیتها در ثباتهای خط لوله جلو می روند و در قسمت مربوطه استفاده می شوند.

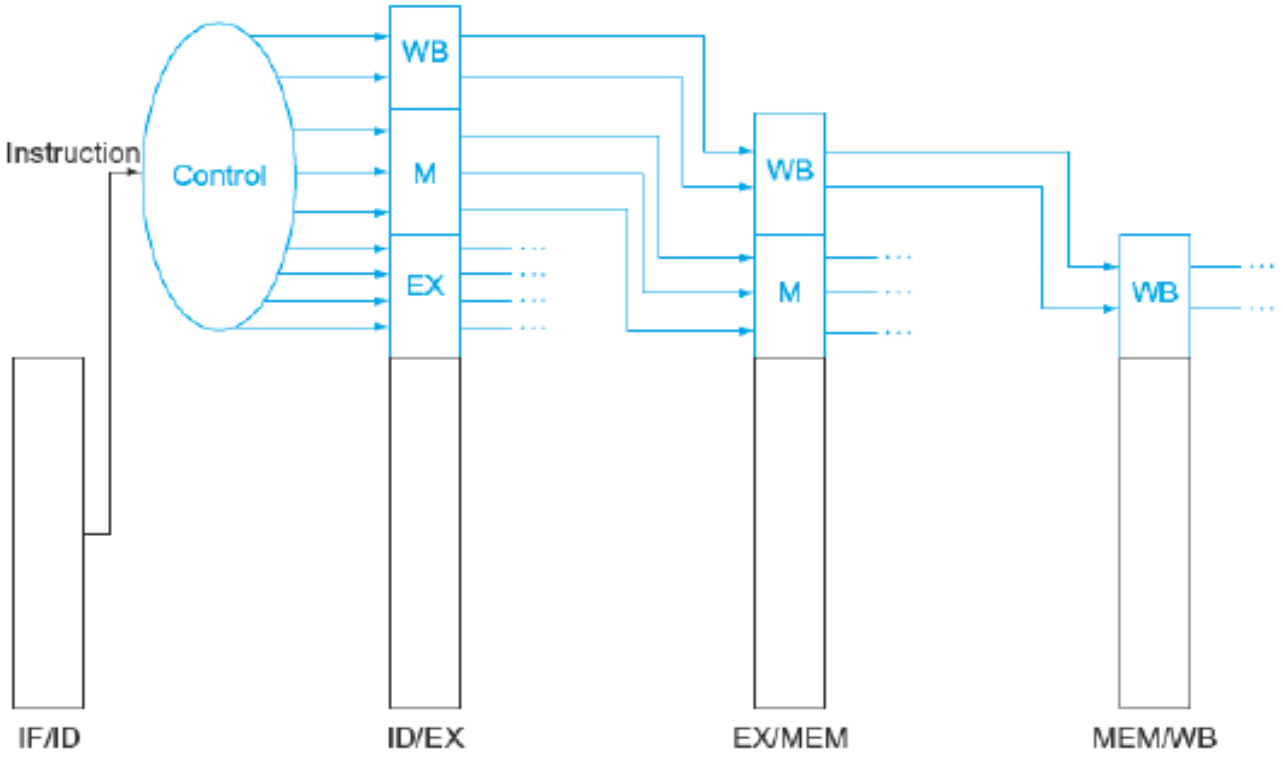


FIGURE 4.50 The control lines for the final three stages. Note that four of the nine control lines are used in the EX phase, with the remaining five control lines passed on to the EX/MEM pipeline register extended to hold the control lines; three are used during the MEM stage, and the last two are passed to MEM/WB for use in the WB stage.

نمایش خط لوله با واحد کنترل و سیگنالهای کنترلی

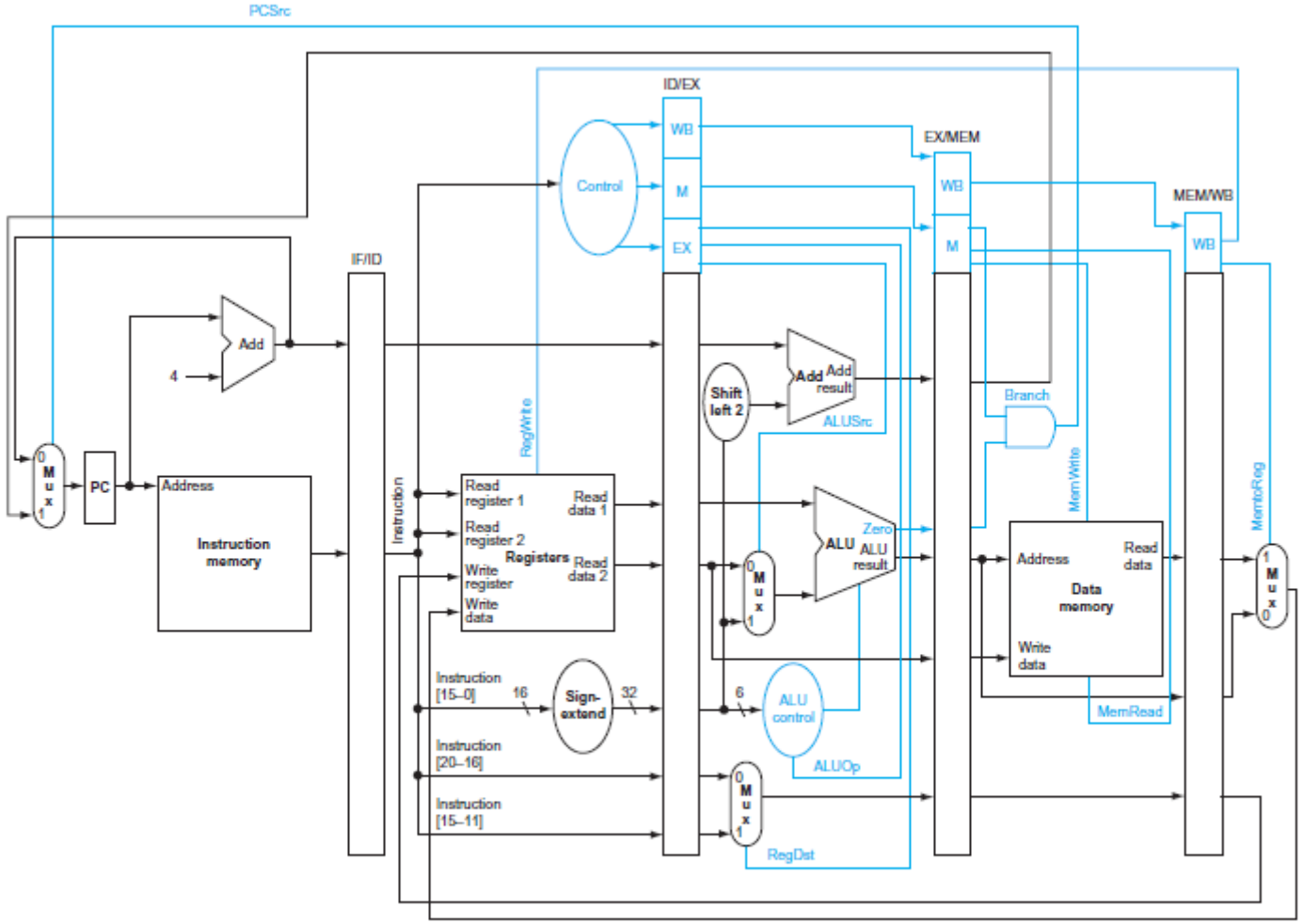
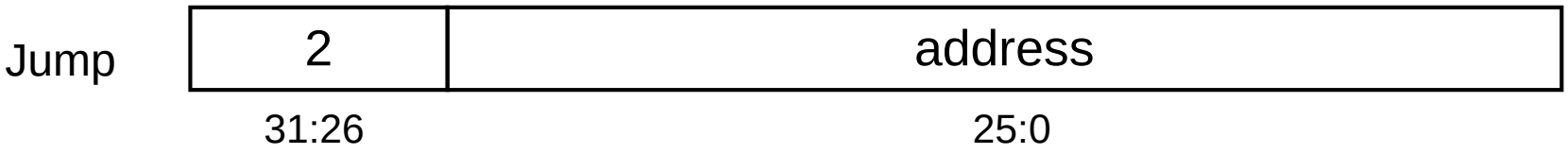


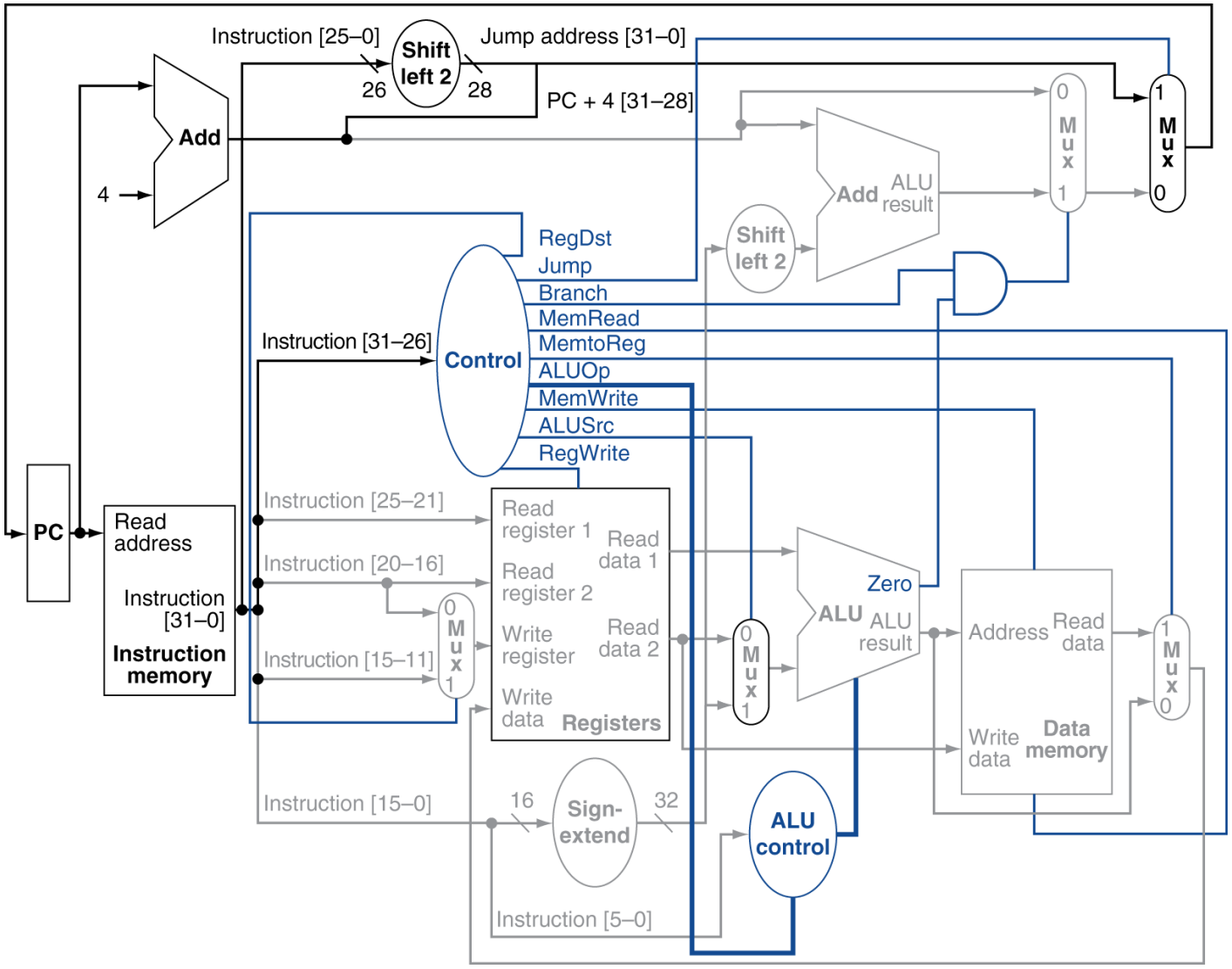
FIGURE 4.51 The pipelined datapath of Figure 4.46, with the control signals connected to the control portions of the pipeline registers. The control values for the last three stages are created during the instruction decode stage and then placed in the ID/EX pipeline register. The control lines for each pipe stage are used, and remaining control lines are then passed to the next pipeline stage.

Implementing Jumps

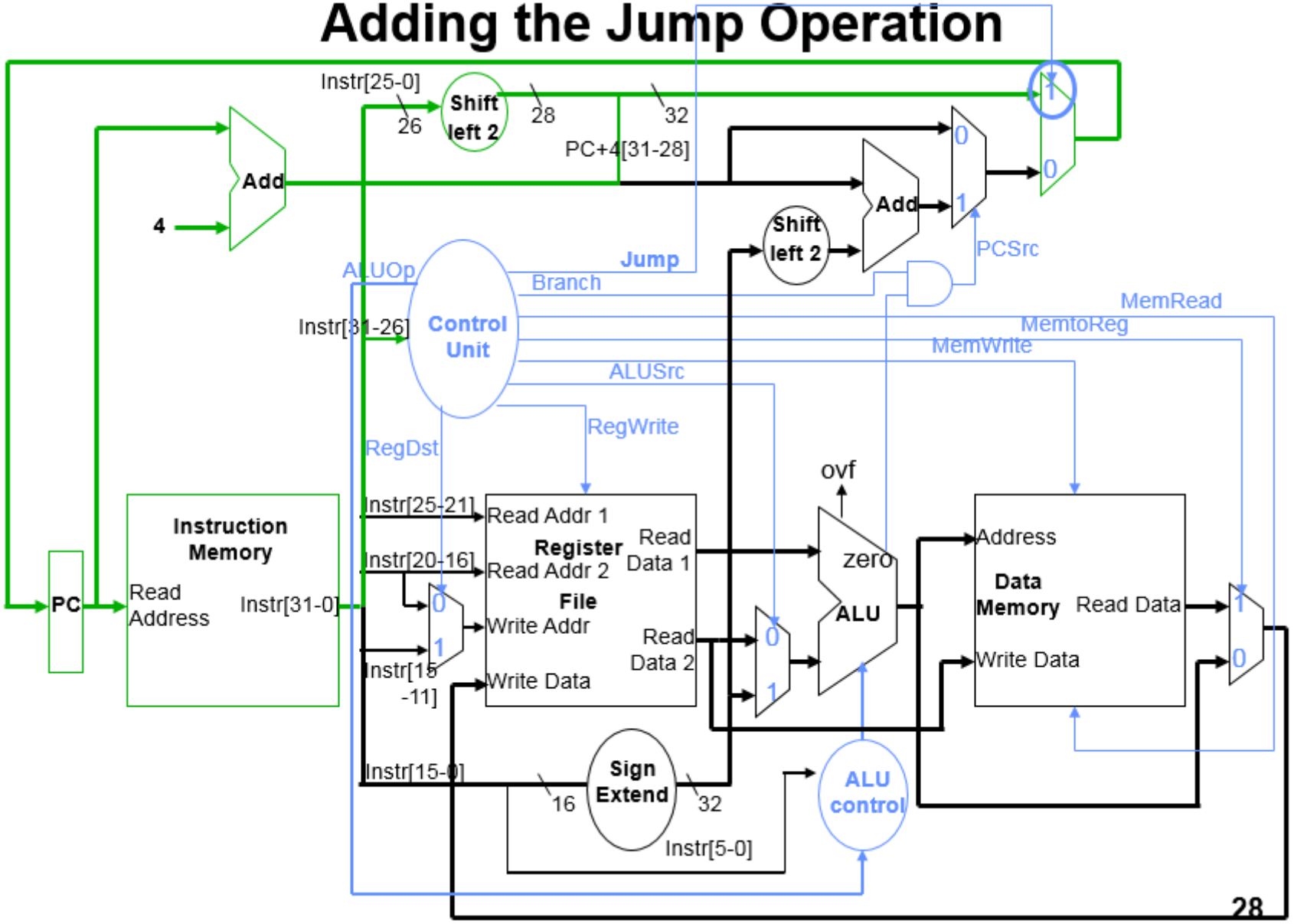


- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of PC+4
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode

Datapath With Jumps Added



Adding the Jump Operation



مثال 1: در این قطعه کد، 4 دستور آخر به دستور اول وابستگی داده دارند. چگونه بین دستورات NOPبگذاریم تا برنامه بدون مخاطره اجرا شود؟

Let's look at a sequence with many dependences, shown in color:

```
sub   $2, $1,$3      # Register $2 written by sub
and   $12,$2,$5      # 1st operand($2) depends on sub
or    $13,$6,$2      # 2nd operand($2) depends on sub
add   $14,$2,$2      # 1st($2) & 2nd($2) depend on sub
sw    $15,100($2)    # Base ($2) depends on sub
```

آیا با استفاده از فورواردینگ می توان مشکل را حل کرد یا باز هم نیاز به No Operation (توسط کامپایلر) و یا توقف خط لوله (توسط سخت افزار) داریم؟

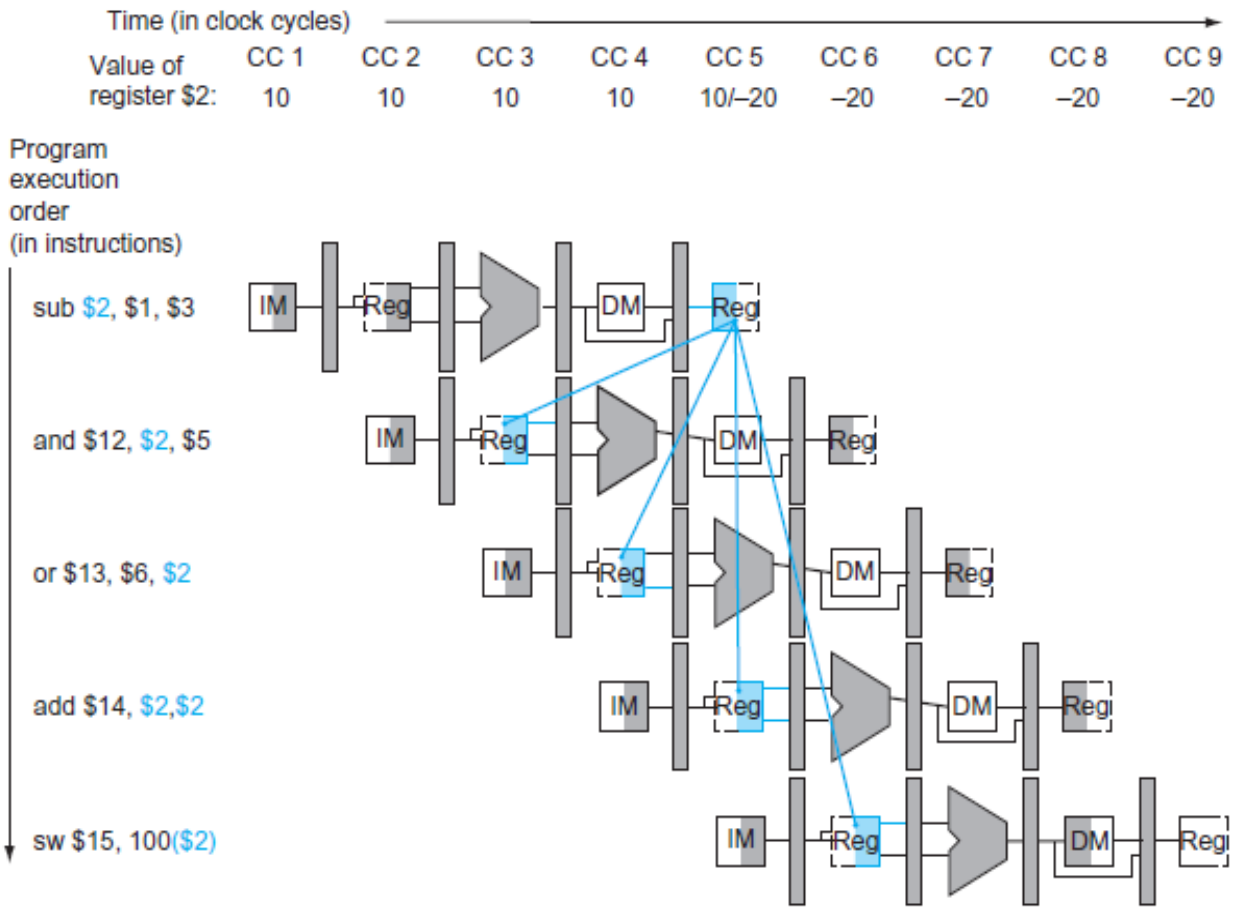


FIGURE 4.52 Pipelined dependences in a five-instruction sequence using simplified datapaths to show the dependences. All the dependent actions are shown in color, and "CC 1" at the top of the figure means clock cycle 1. The first instruction writes into $2, and all the following instructions read $2. This register is written in clock cycle 5, so the proper value is unavailable before clock cycle 5. (A read of a register during a clock cycle returns the value written at the end of the first half of the cycle, when such a write occurs.) The colored lines from the top datapath to the lower ones show the dependences. Those that must go backward in time are *pipeline data hazards.*

بله در این حالت که دستورات ثباتی به هم وابسته هستند می توان با فورواردینگ مشکل را حل کرد.

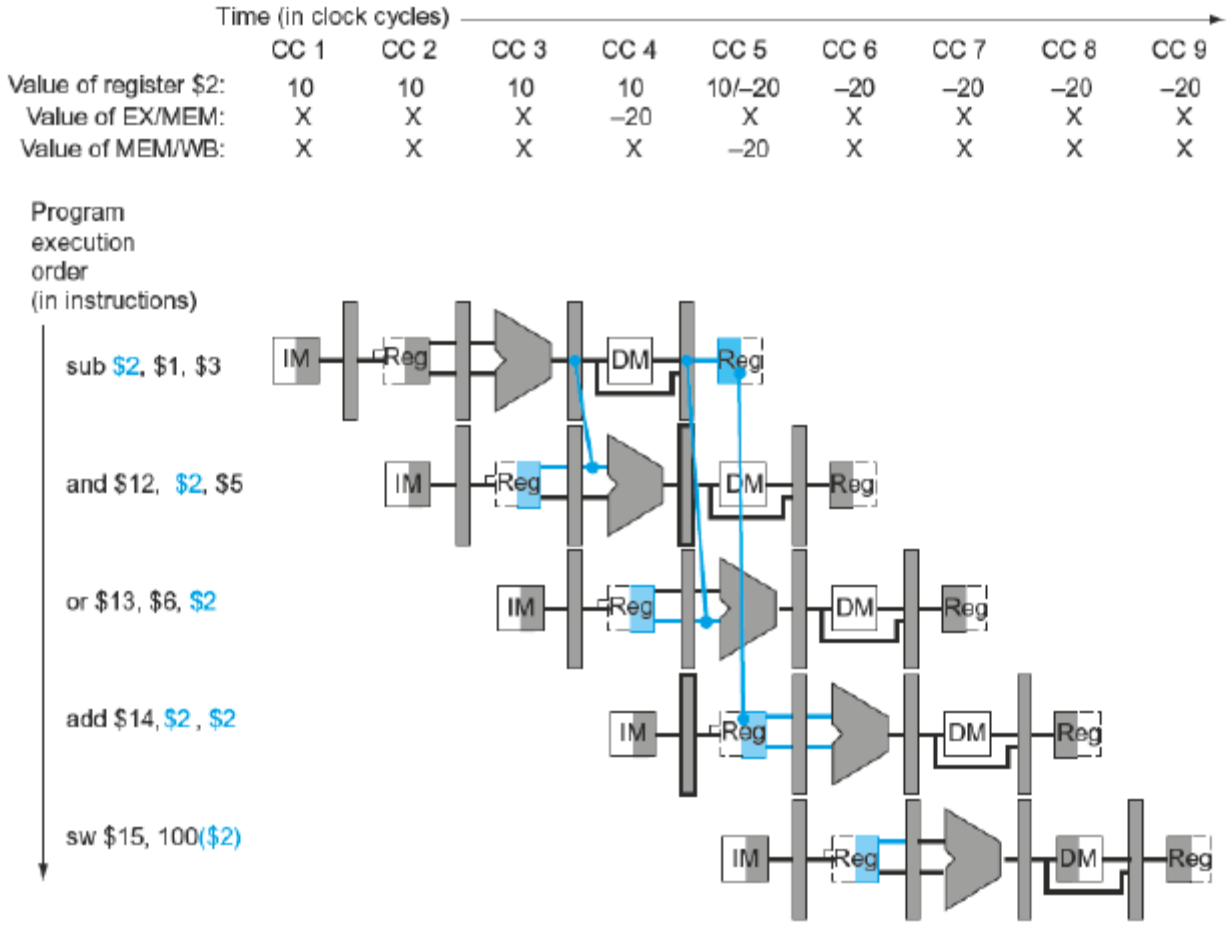


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and –20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

مثال 2: اگر دستور ثباتی به دستور بار کردن وابسته باشد آیا با فورواردینگ مشکل حل می شود؟

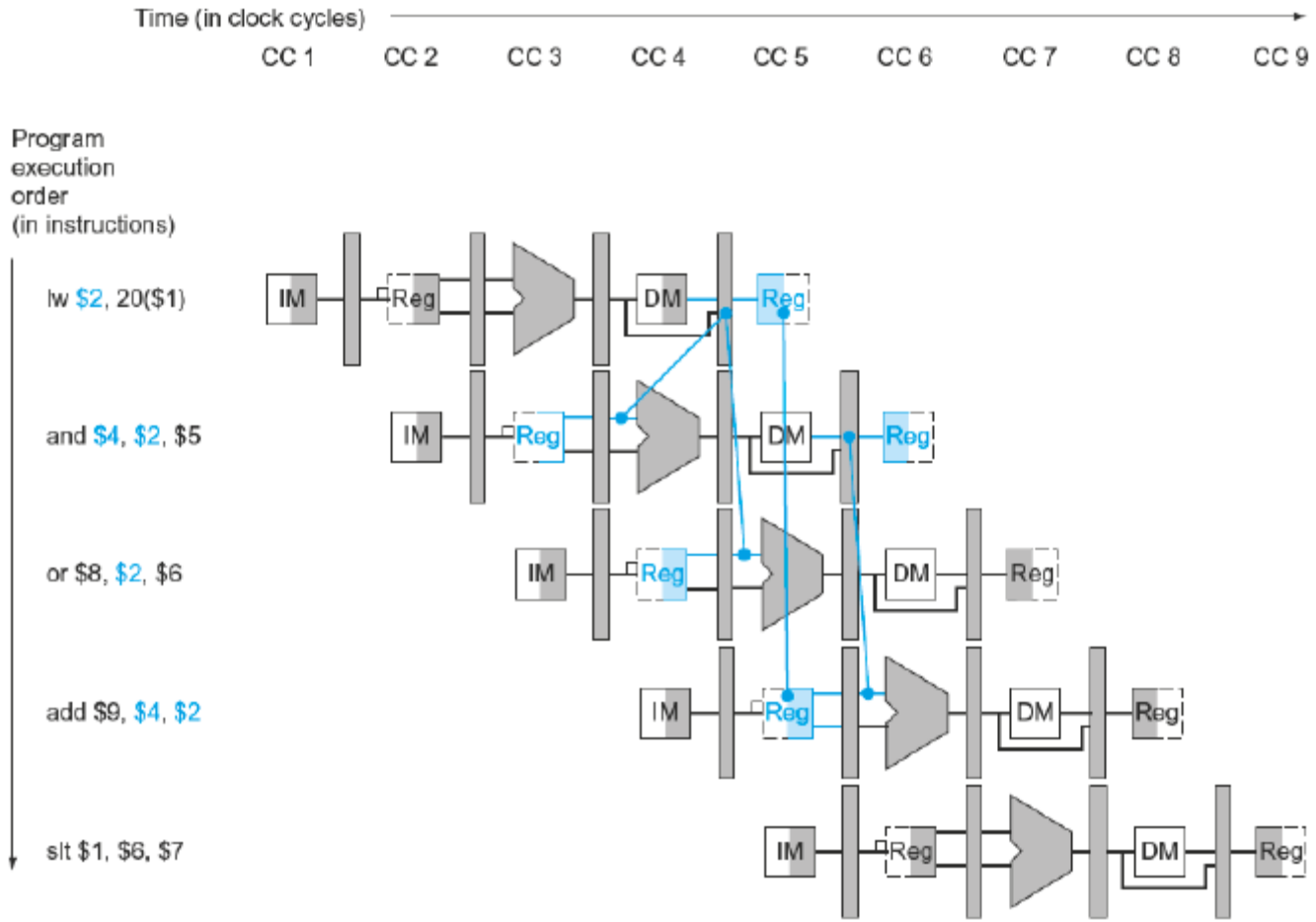


FIGURE 4.58 A pipelined sequence of instructions. Since the dependence between the load and the following instruction (and) goes backward in time, this hazard cannot be solved by forwarding. Hence, this combination must result in a stall by the hazard detection unit.

خیر. یا باید توسط کامپایلر پس از دستور بارکردن یک دستور NOP اضافه شود و یا توسط سخت افزار، مطابق شکل، دسنور دوم پس از مرحله دیکود تبدیل به حباب شود. در این حالت، پس از ایجاد یک سیکل حباب، با استفاده از فورواردینگ مشکل وابستگی داده حل خواهد شد.

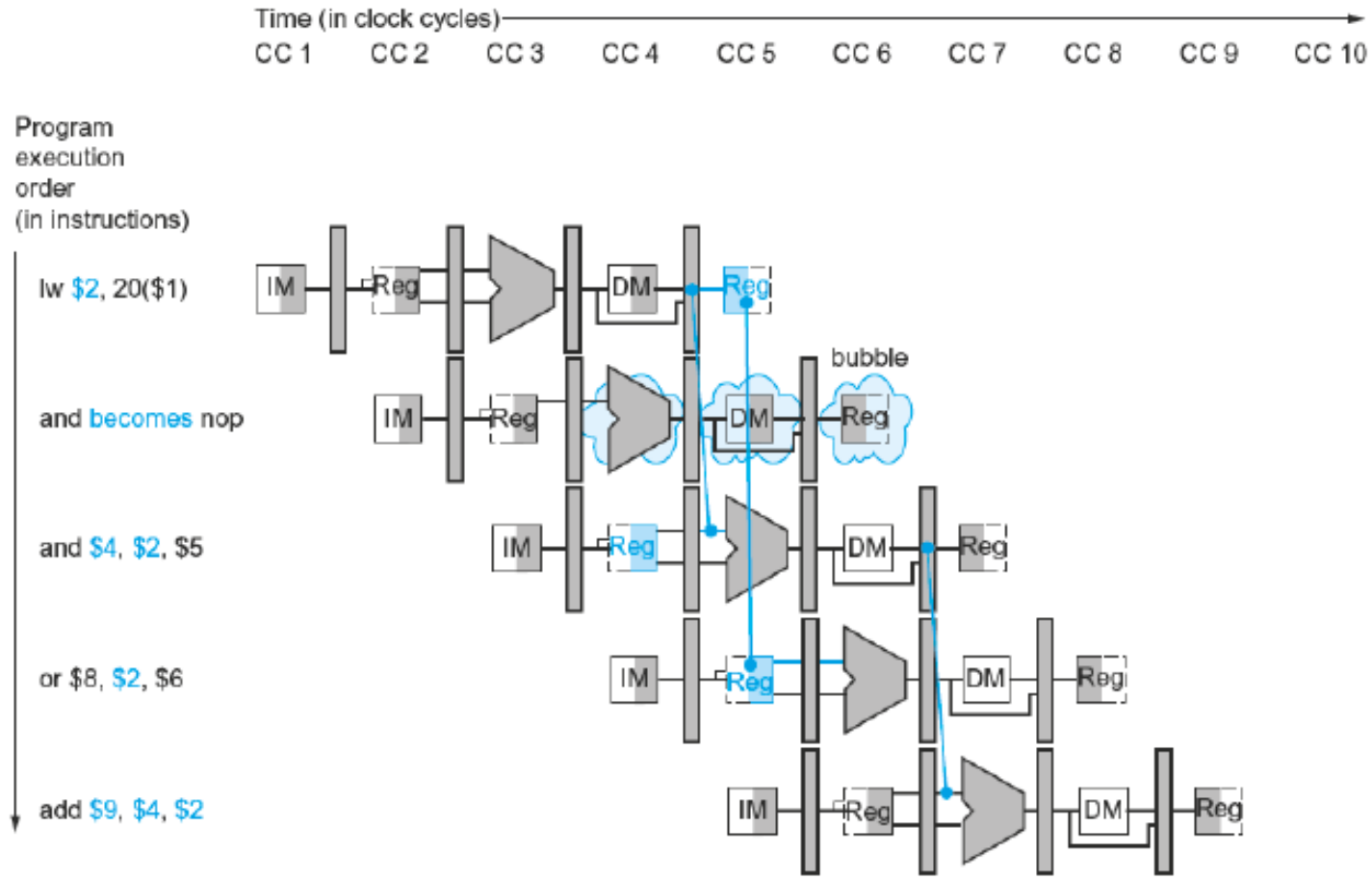


FIGURE 4.59 The way stalls are really inserted into the pipeline. A bubble is inserted beginning in clock cycle 4, by changing the and instruction to a nop. Note that the and instruction is really fetched and decoded in clock cycles 2 and 3, but its EX stage is delayed until clock cycle 5 (versus the unstalled position in clock cycle 4). Likewise the OR instruction is fetched in clock cycle 3, but its ID stage is delayed until clock cycle 5 (versus the unstalled clock cycle 4 position). After insertion of the bubble, all the dependences go forward in time and no further hazards occur.

Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Data hazards when
  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

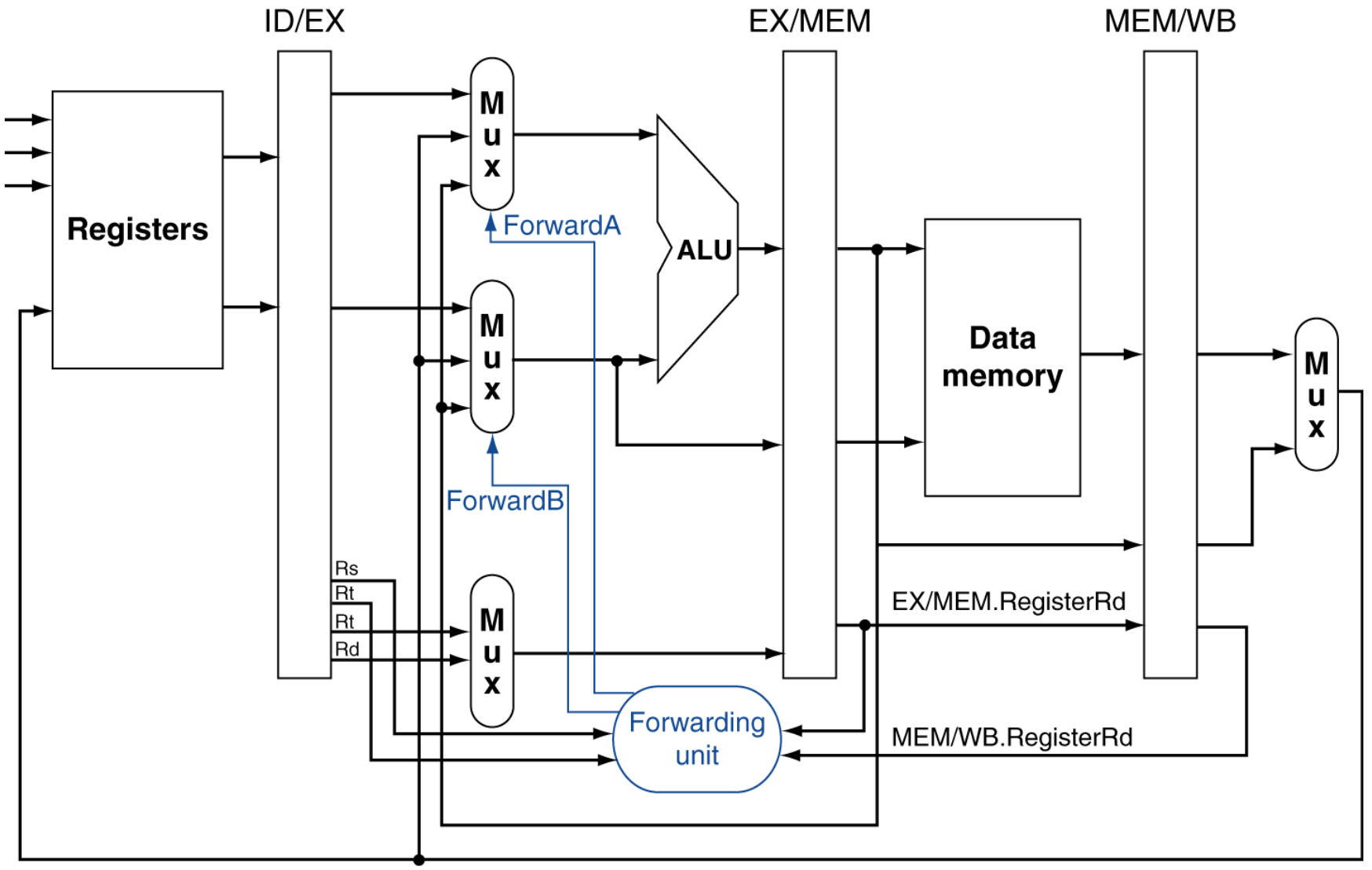
Fwd from EX/MEM pipeline reg

Fwd from MEM/WB pipeline reg

Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not $zero
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

Forwarding Paths



b. With forwarding

Forwarding Conditions

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Double Data Hazard

- Consider the sequence:

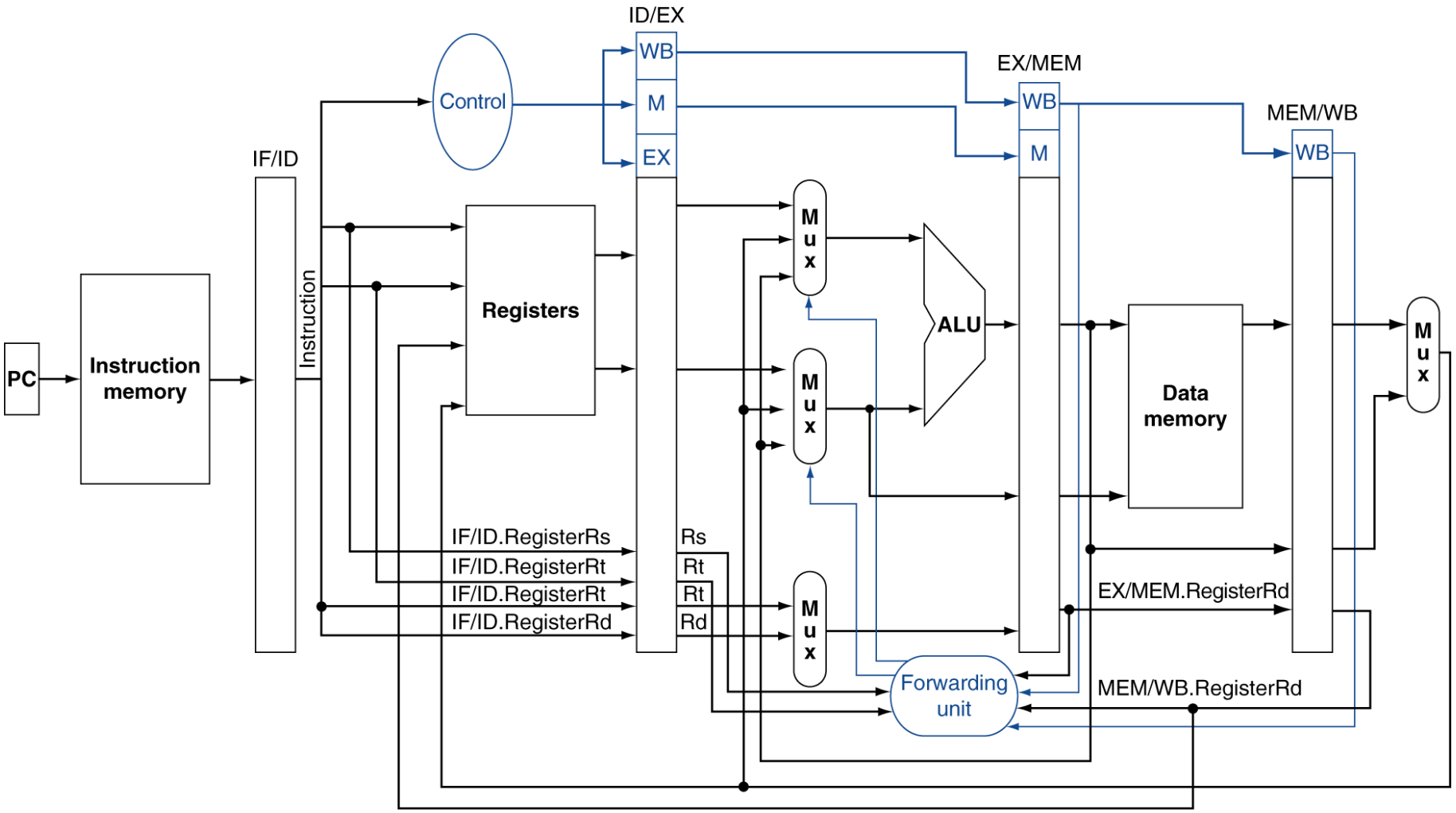
```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

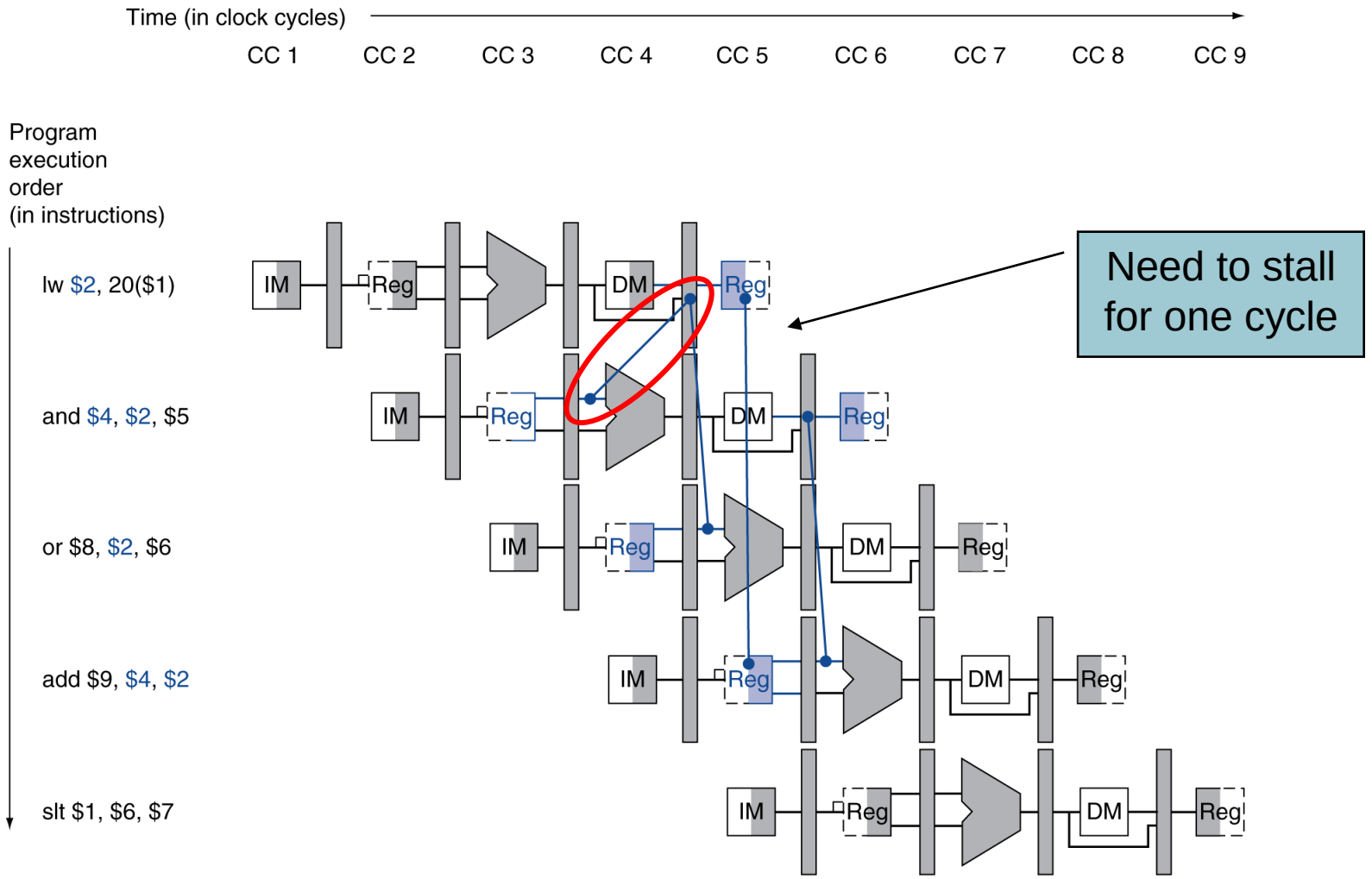
Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Datapath with Forwarding



Load-Use Data Hazard



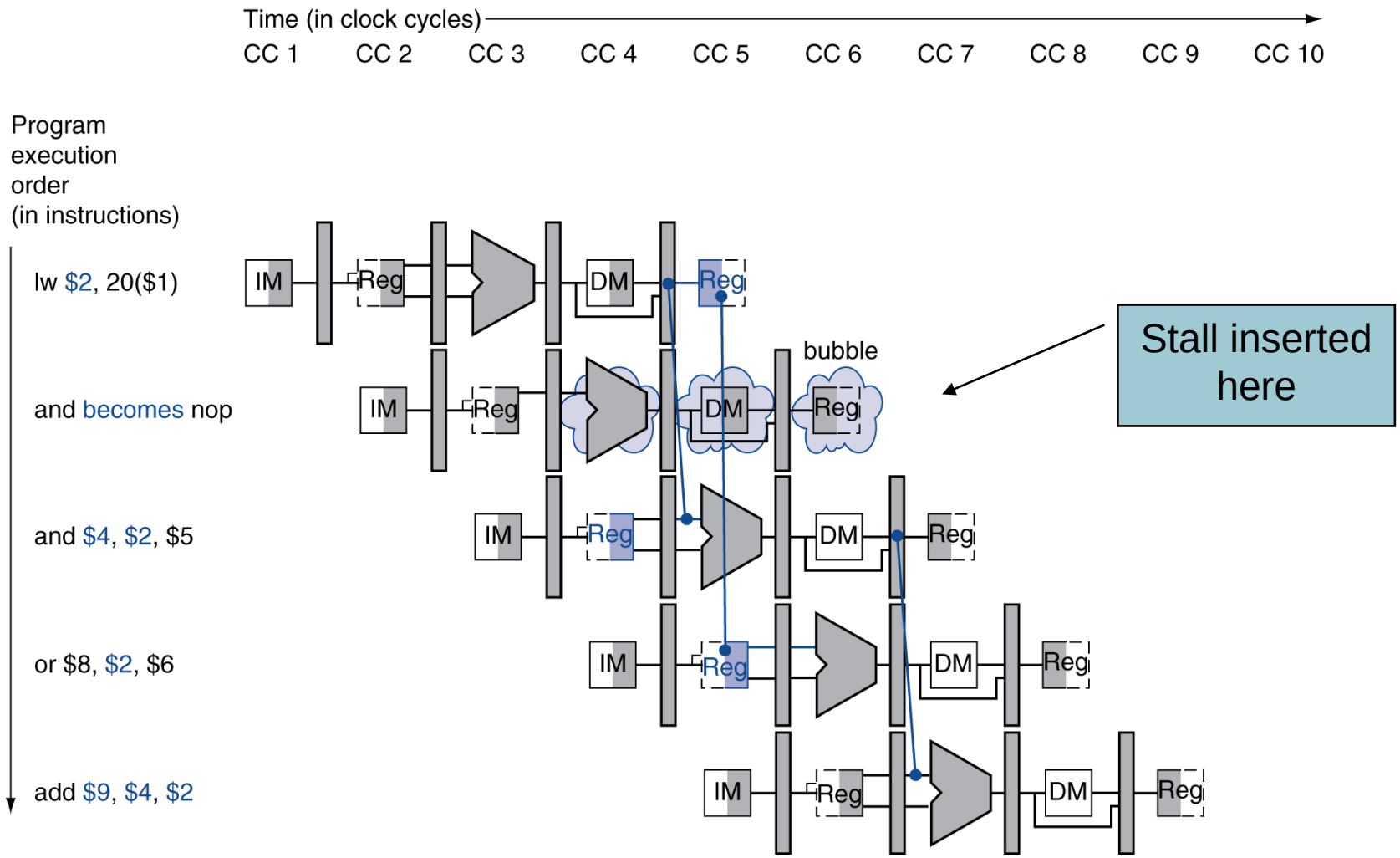
Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

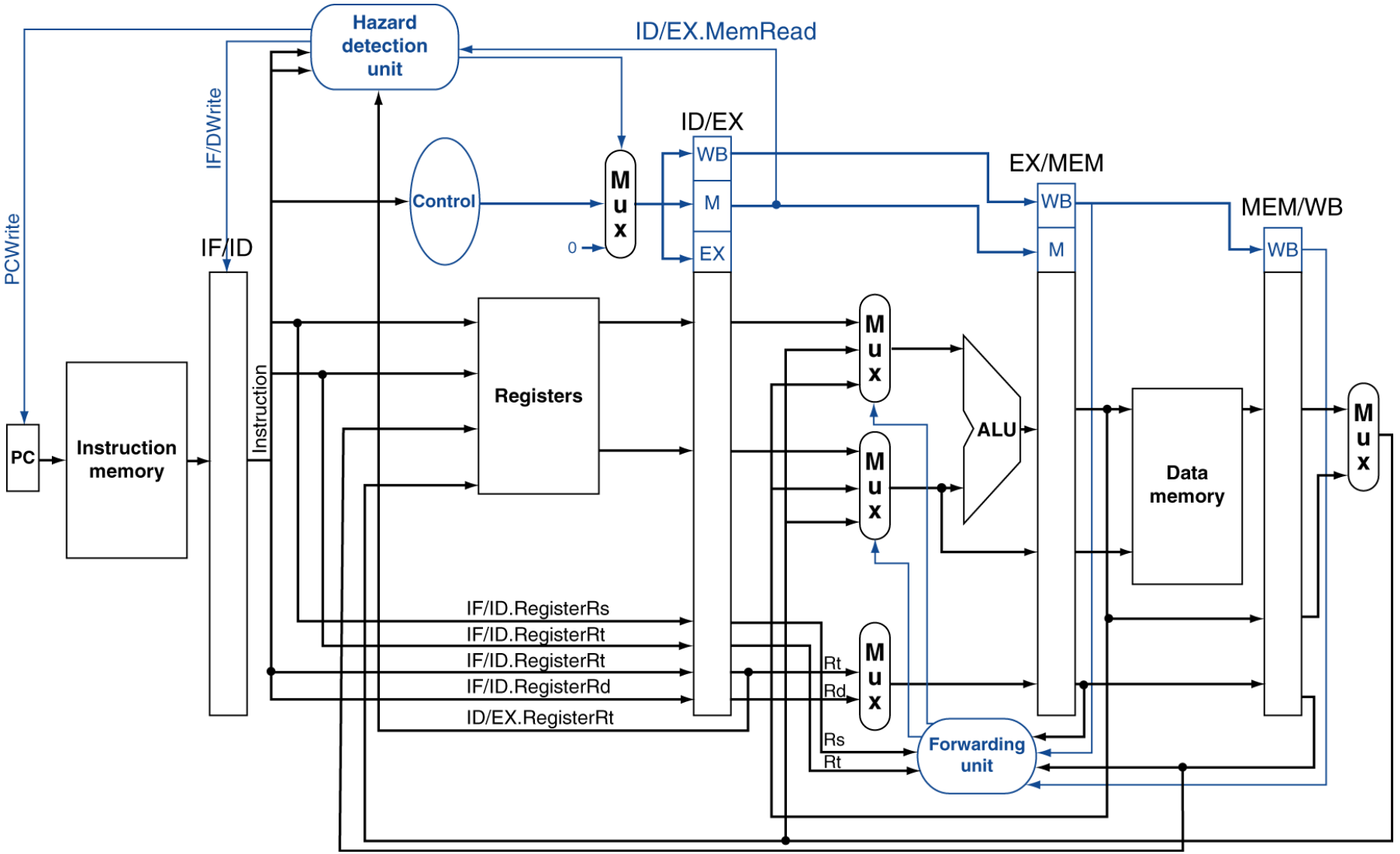
How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for lw
    - Can subsequently forward to EX stage

Stall/Bubble in the Pipeline



Datapath with Hazard Detection

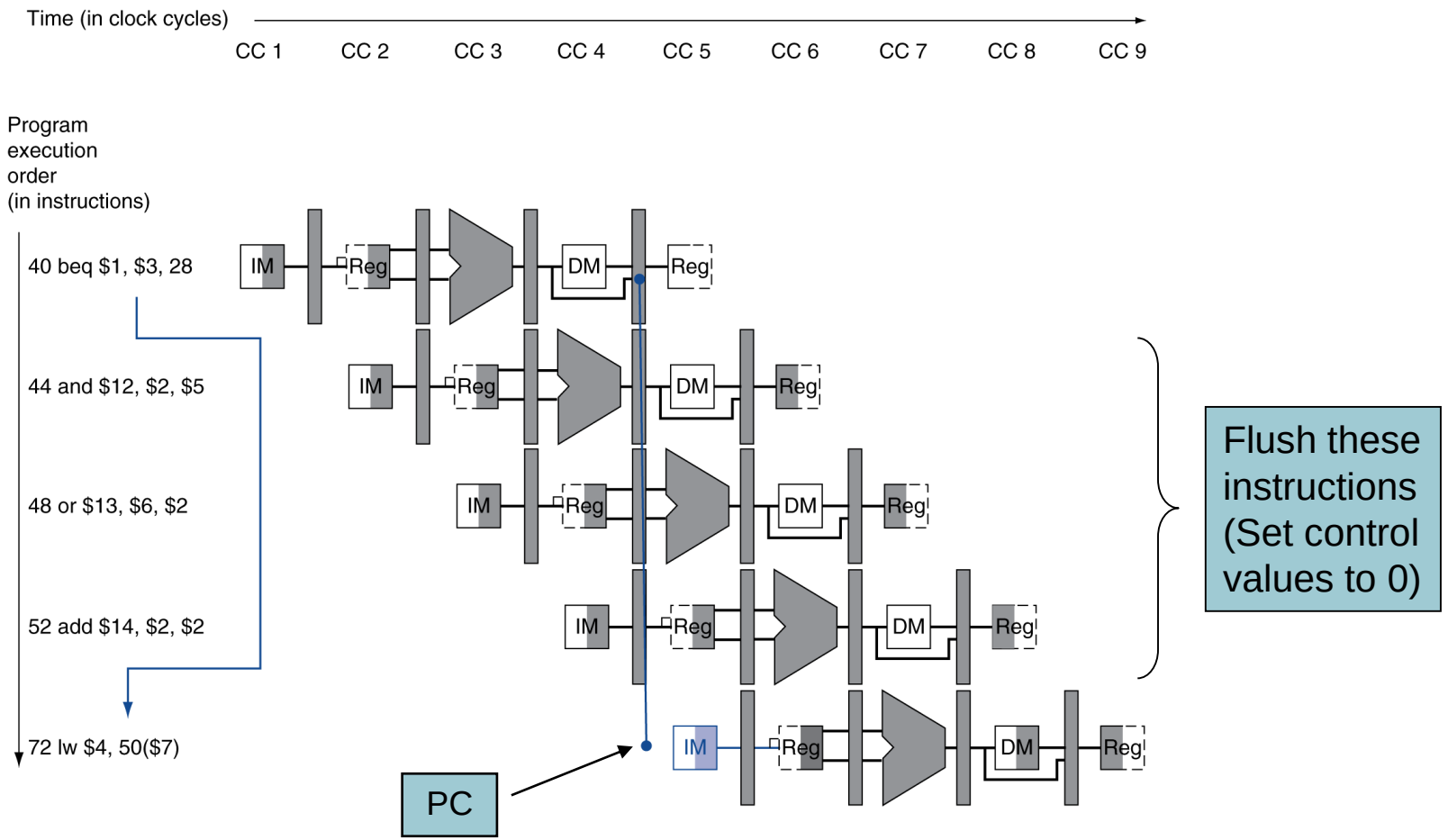


Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

Branch Hazards

- If branch outcome determined in MEM

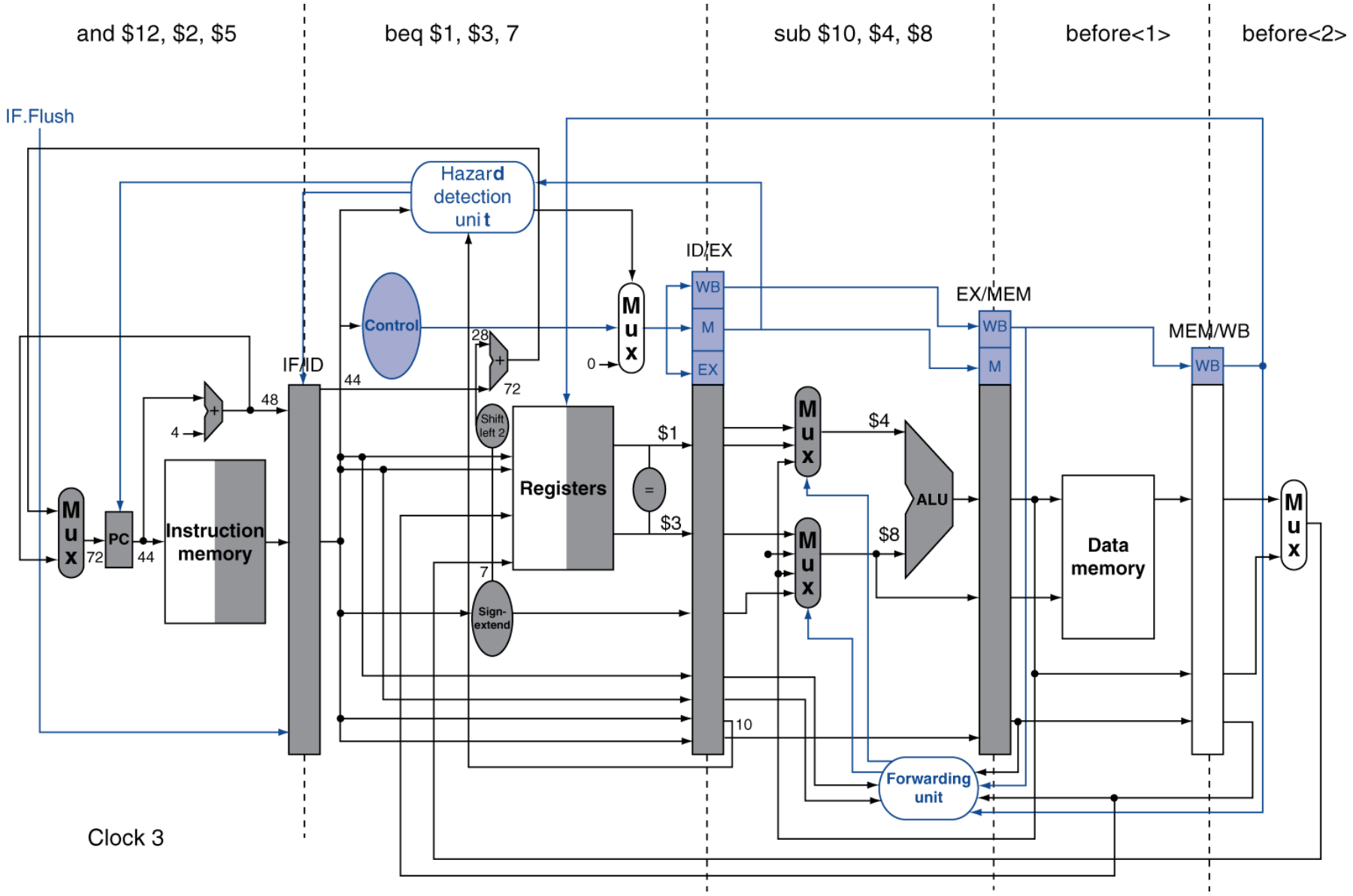


Reducing Branch Delay

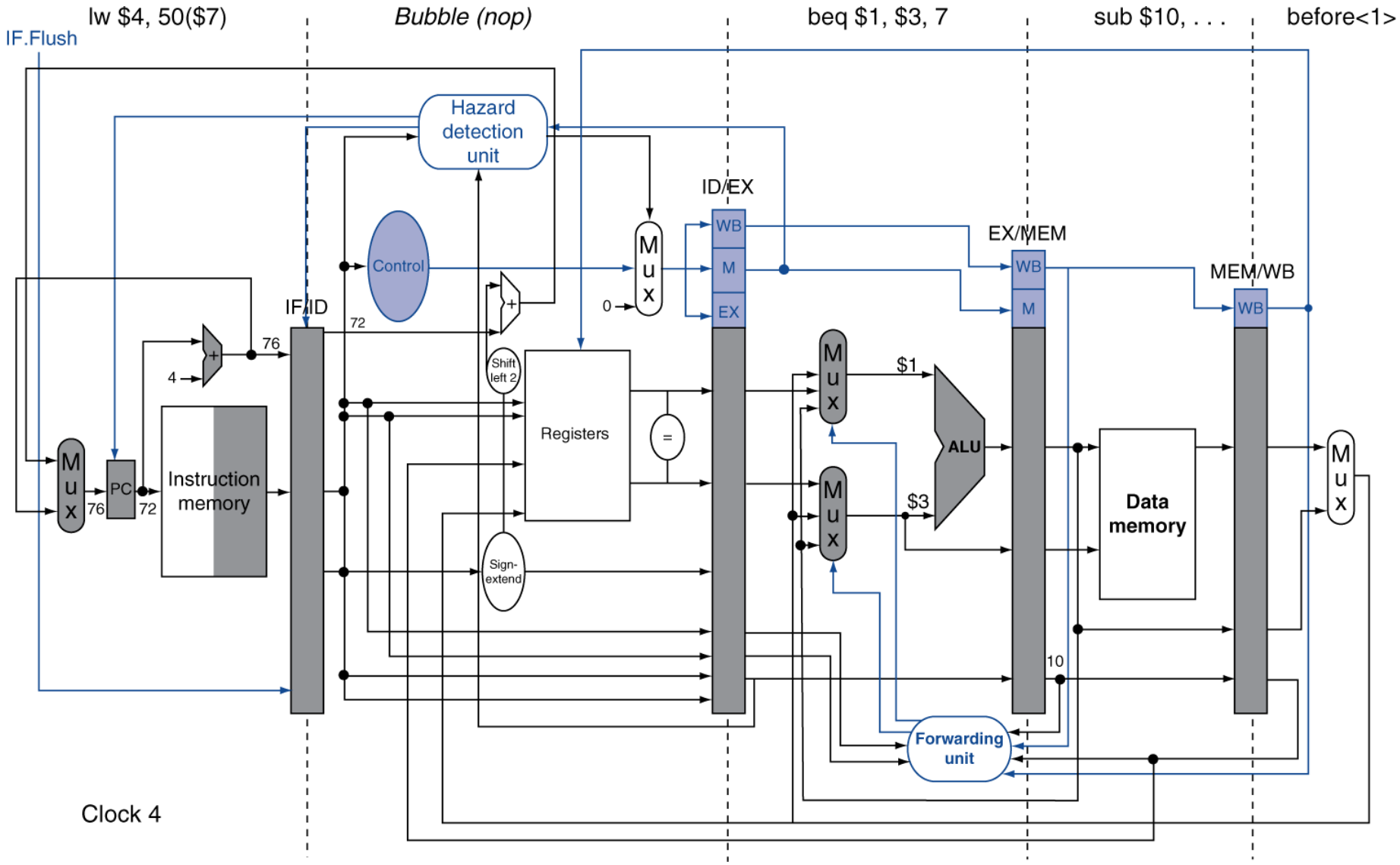
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
48:   or    $13, $2, $6
52:   add   $14, $4, $2
56:   slt   $15, $6, $7
      ...
72:   lw    $4, 50($7)    #44+7x4=72
```

Example: Branch Taken

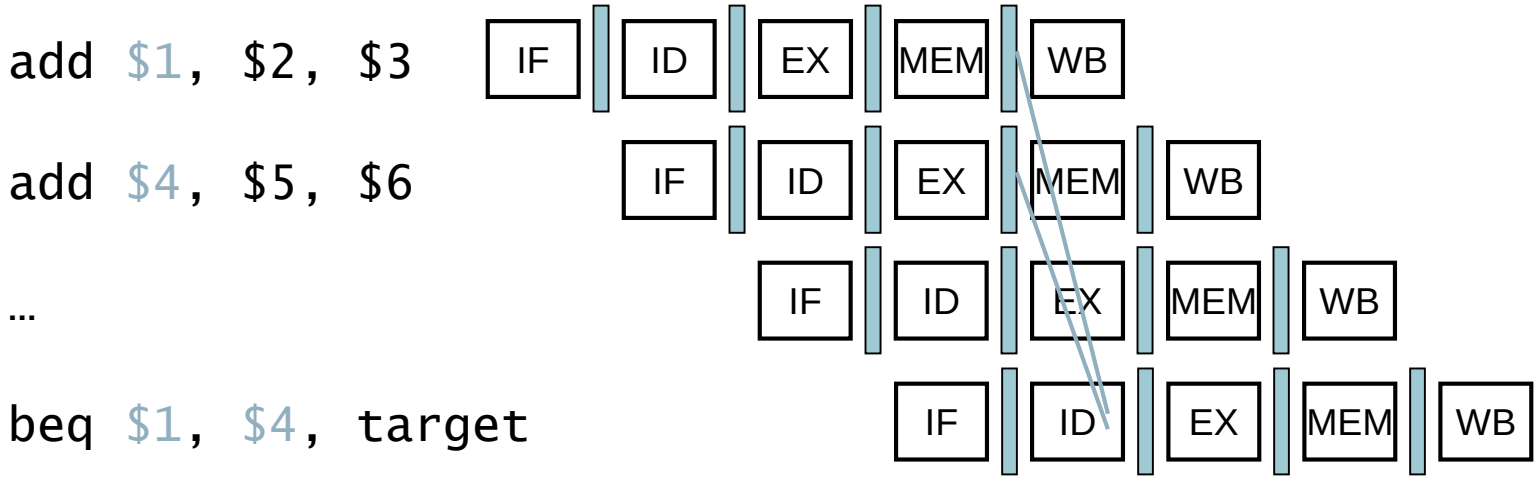


Example: Branch Taken



Data Hazards for Branches

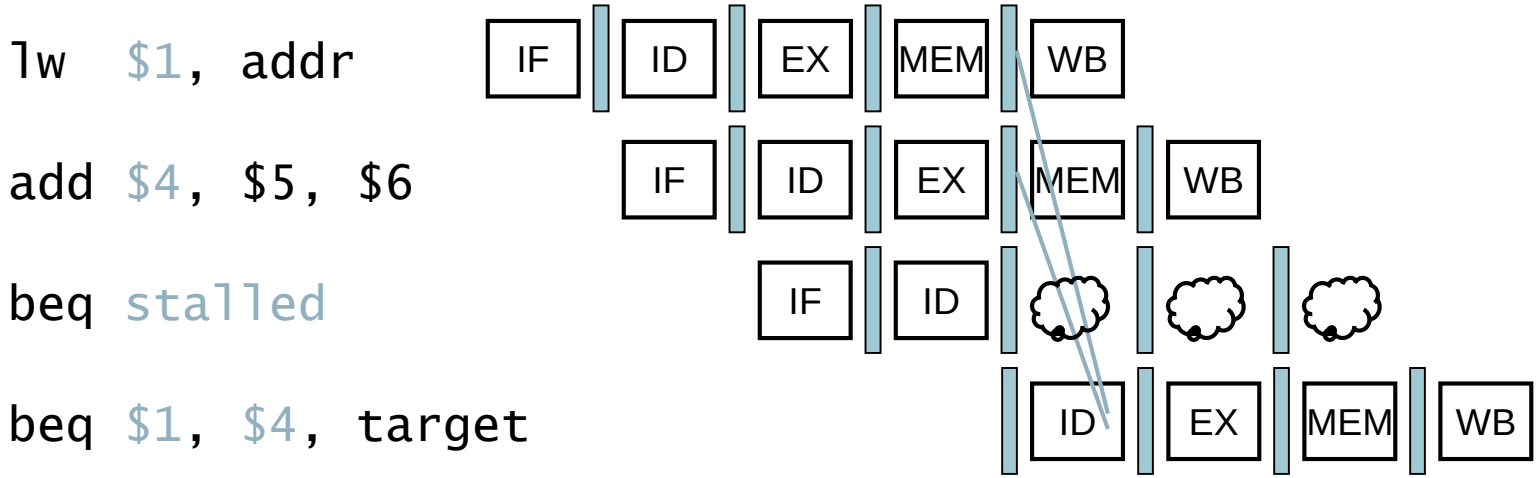
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction



- Can resolve using forwarding

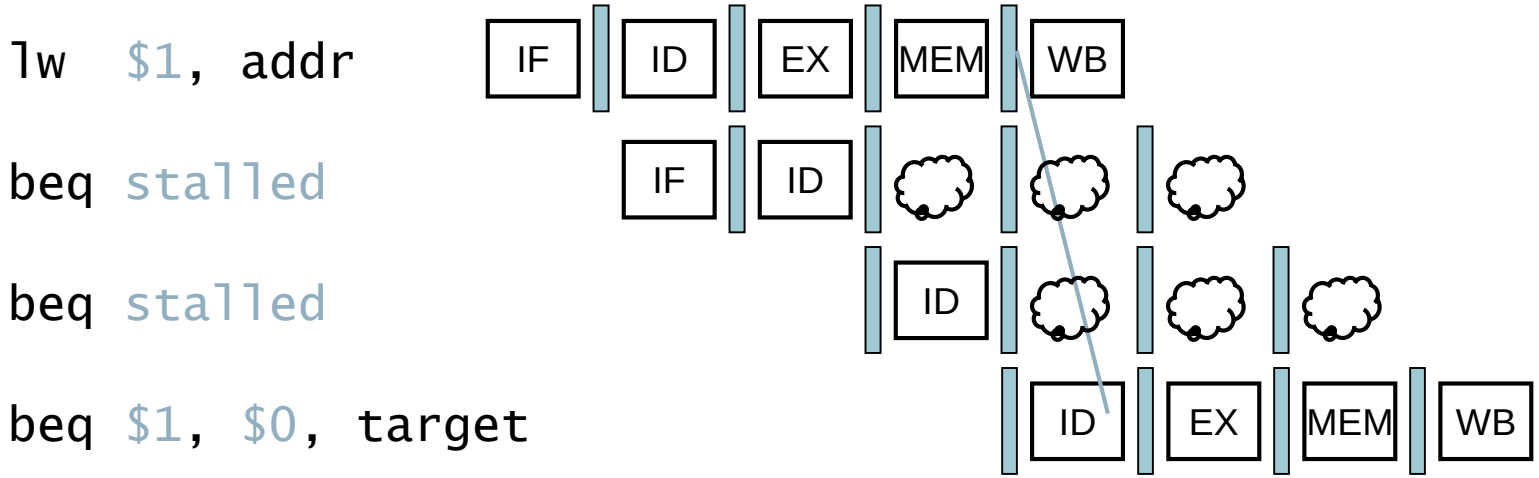
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle



Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles



Power Efficiency

- Complexity of dynamic scheduling and speculations requires power
- Multiple simpler cores may be better

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

Cortex A8 and Intel i7

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline stages	14	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	2-level	2-level
1st level caches/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-1024 KiB	256 KiB
3rd level caches (shared)	-	2- 8 MB