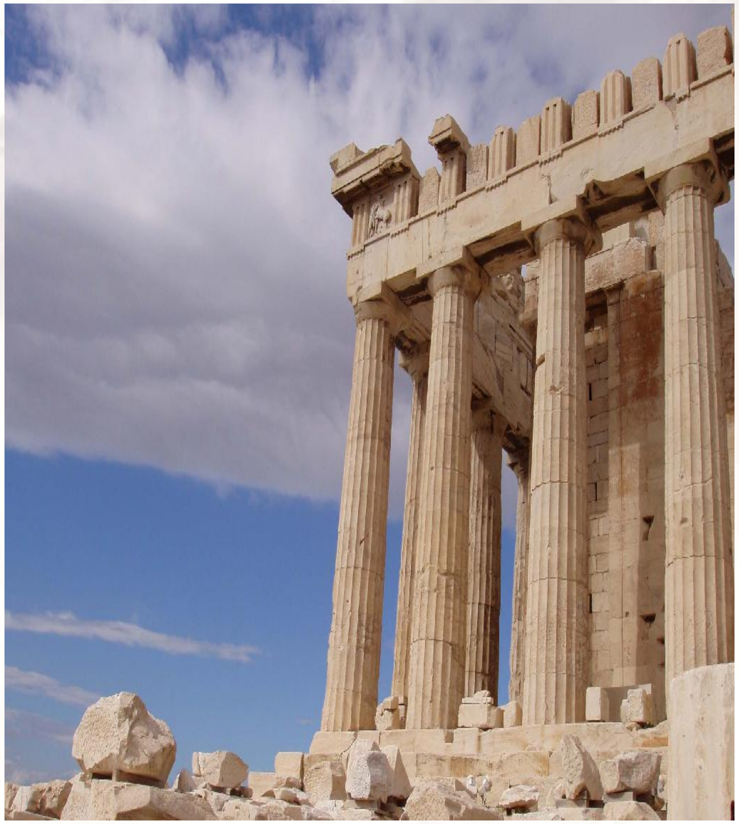
Review



Final examination

- ❑Close-text, with a handwritten A4 memo sheet
- ❑Selections 26%
- ❑Answering Questions in 150 words. 20%
- ❑Calculation 10%
- ❑Storage 10%
- ❑Cache /Memory 12%
- ❑Pipeline 22%

- ❑July 3, 14:00-16:00, Building 4-410(Yu Quan)

Chapter1: Fundamentals

- What's Computer architecture ?
- What's a computer architect's task ?
- What should a computer architect need to know ?

What's Computer architecture ?

- *Computer Architecture* is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance, cost and power goals.

Tradeoff

- *Computer architecture* is the conceptual design and fundamental operational structure of a computer system. Computer architecture is a blueprint and functional description of requirements (especially speeds and interconnections) and design implementations for the various parts of a computer — focusing largely on the way by which the central processing unit (CPU) performs internally and accesses addresses in memory.

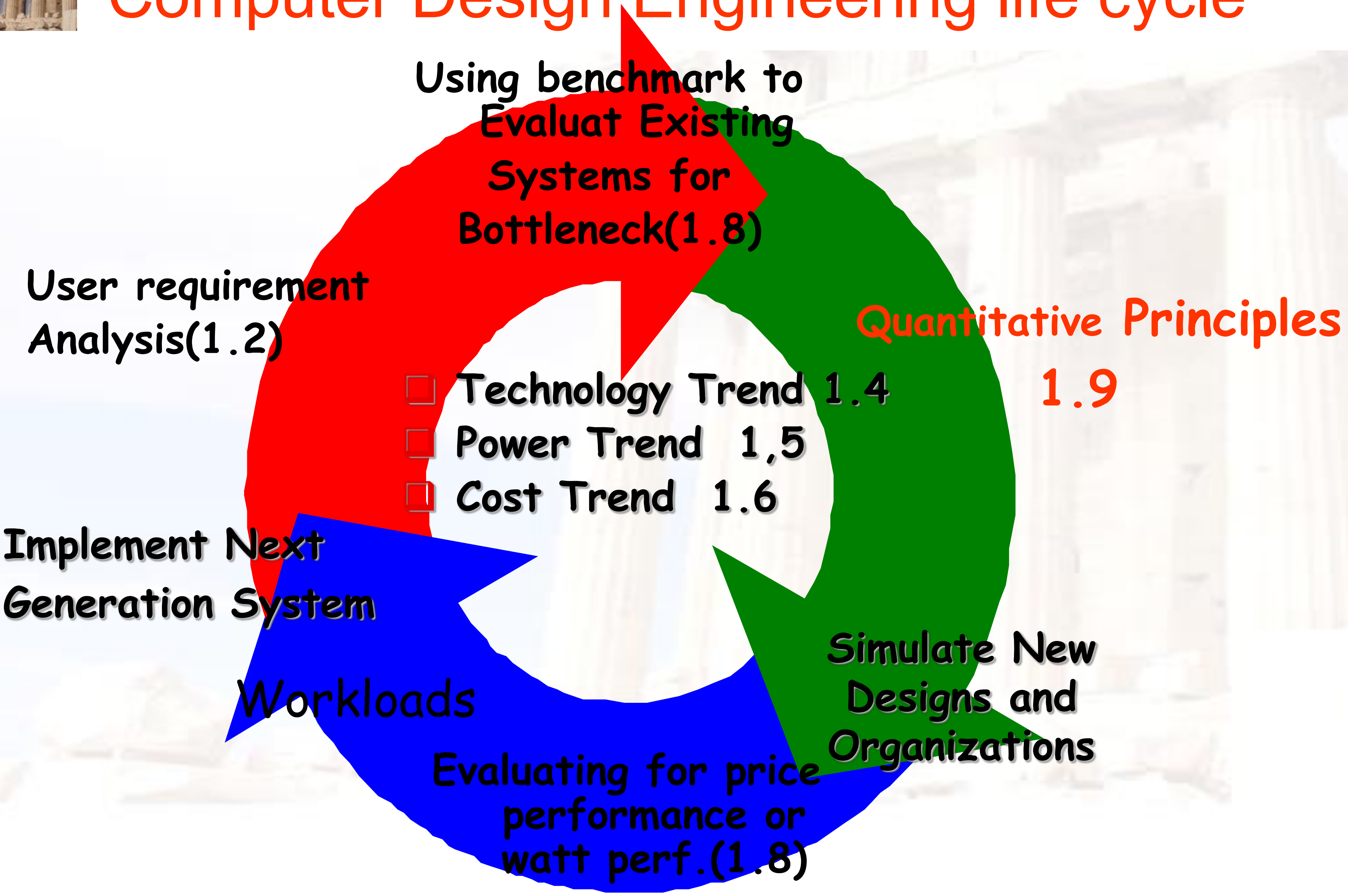
What's Computer Architect's task ?

- Define the requirements
  - Application field, computer market(1.2)
  - Functional requirement
  - non-functional requirements

$\left\{ \right.$ Supercomputer
PC, workstation
Embedded system
Server

- Architecture design (blueprint)
  - ISA
  - Organization
  - Hardware
- Optimization

Computer Design Engineering life cycle


Using benchmark to Evaluat Existing Systems for Bottleneck(1.8)
Quantitative Principles 1.9
Simulate New Designs and Organizations
Evaluating for price performance or watt perf.(1.8)
Workloads
Implement Next Generation System
User requirement Analysis(1.2)
Technology Trend 1.4
Power Trend 1,5
Cost Trend 1.6

1.4 Technology Trends

- Designers often design for the next technology.
- Technology improves continuously, an impact of this improvements can be in discrete leaps.
- Moore Law
  - In 1965 , number of components the industry would be able to place on a computer chip would double every year.
  - In 1975, he updated his prediction to once every two years.
- Rule of thumb
  - Bandwidth grow rate ~ improvement in latency2
  - transistor perf. Improves linearly with decreasing feature size.
  - signal delay for a wire increase in proportion to the product of its resistance and capacitance.

1.5 Trends in Power

- Power also provide challenges as device scaled
  - first microprocessor: 1/10watt --> 2GHz P4: 135watt
- Challenges:
  - distributing the power
  - removing the heat
  - preventing hot spot
- What to do? power-aware design

1.6 Cost trend

- Understanding cost trends of component is important, since we design for tomorrow !
- How to learn ? Learning curve.
- The impact factors for cost:
  - Time, Volume, modification
- Rules of Thumb
  - Twice the yield will have half the cost.
  - Cost decrease about 10% for each doubling of volume.

Cost vs. Price

- Component costs
  - Raw material cost.
- Direct cost:
  - Costs incurred to make a single item. Adds 20% to 40% to component cost.
- Gross margin (Indirect cost):
  - Overhead not associated with a single item, i.e. R&D, marketing, manufacturing equipment, taxes, etc.
- Average Selling Price (ASP):
  - Component cost + direct cost + indirect cost.
- List price :
  - Not ASP. Stores add to the ASP to get their cut. Want 50% to 75% of list price.

Small changes in cost can have an unexpected large increase in price.

1.7 Dependability

- Dependability is a deliberately broad term to encompass many facets including reliability, security and availability.
- Service level Agreements(SLA)/Service level objectives (SLO)
  - Service accomplishment
  - Service interruption
- Failures: $S_{accomplishment} \rightarrow S_{interruption}$
- Restorations: $S_{interruption} \rightarrow S_{accomplishment}$
- Redundancy:
  - Time redundancy: repeat the operation again to see if it is still in erroneous.
  - Resource redundancy: have other components to take over from the one that failed.

Measurements of Dependability

- Module reliability: continuous service accomplishment
  - MTTF: Mean Time To Failure
  - MTTR: Mean Time To Repair
  - FIT: Failure In Time = 1/MTTR
  - MTBF: Mean Time Between Failure = MTTF+MTTR
- Module availability
  - $$\frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

1.8 Measuring and Reporting Perf.

- **Performance Metrics**
  - response time or elapsed time – user perception
  - Throughput --administrator perception
  - User CPU time – computer architect
  - MIPS - millions of instructions per second
- **Comparing Machines Using Sets of Programs**
  - Choosing which program to evaluate performance
    - Benchmark Suites
  - Different Means: Arithmetic, Harmonic, and Geometric Means

Summary of performance metrics

- Response (Execution) time
  - user perception
  - system performance
  - **the only unimpeachable measure of performance**
- CPU time
  - designer perception
  - CPU performance
- Throughput
  - administrator perception
- MIPS
  - merchant perception

1.9 Quantitative Principles

- ❑Take Advantage of Parallelism
- ❑Focus on the common case
- ❑Principle of Locality
- ❑Amdahl's Law
- ❑CPU Performance Equation

Where the principles are used ?
Could you give some examples ?

Exploiting the Parallelism

- Taking advantage of parallelism is one of the most important methods for improving performance.

- Many example
  - system level: multiple processors / multiple disk,
  - individual processor: pipelining
  - digital design level: set-associative caches

What the Amdahl's Law imply ?

- If an enhancement is only usable for a fraction of task, then the total speedup will be no more than 1/ (1-F).
- Serve the guide
  - to how much an enhancement will improve performance
  - to how to distribute resource to improve cost-performance
- Useful for comparing
  - the overall system performance of two alternatives,
  - two CPU design alternatives
- We can improve the performance by
  - increasing the $\text{Fraction}_{\text{enhanced}}$
  - or, increasing the $\text{Speedup}_{\text{enhanced}}$

The CPU Performance Equation

- *The "Iron Law" of processor performance:*
  - Often it is difficult to measure the improvement in time using a new enhancement directly.
- CPU Performance Equation

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Calculation of CPU Time

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

Or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

Component of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instructions (CPI)	Average number of clock cycles/instruction
Clock cycle time	Seconds per clock cycle

Related technologies

- CPU performance is dependent upon 3 characteristics:
  - clock cycle (or rate) (CC)
  - clock cycles per instruction (CPI)
  - instruction count. (IC)

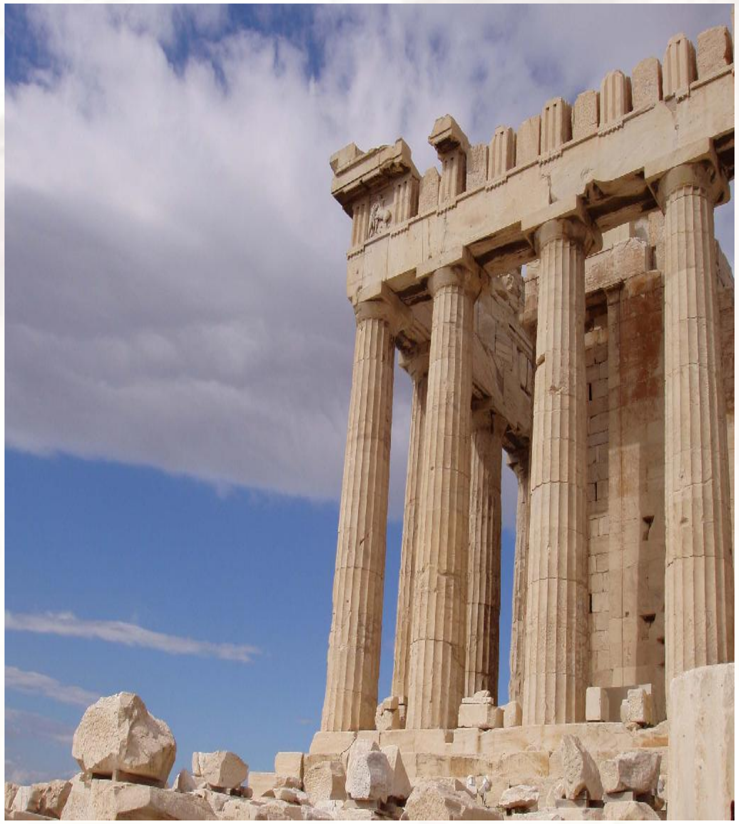
	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

- One difficulty: It is difficult to change one in isolation of the others.

Principle of Locality

- Programs tend to reuse data and instructions they have used recently.
- a program spends 90% of its execution time in only 10% of the code.
- Temporal locality
  - Recently accessed items are likely to be accessed in the near future.
- Spatial locality
  - Items whose addresses are near one another tend to be referenced close together in time.

ChapterB
Outline of ISA



Seven dimensions of ISA

- ❑Class of ISA
- ❑Memory addressing
- ❑Addressing modes
- ❑Types and sizes of operands
- ❑Operations
- ❑Control flow instructions
- ❑Encoding an ISA
- ❑The impact of compiler

Important step for ISA design

- To analyze and evaluate the existing machines with a large collection of programs before making architectural decisions.

Classifying Instruction Set Architectures

- The type of internal storage in CPU
  - Stack, accumulator, GPR(General-Purpose Register) architecture
- Max number of operands in ALU instruction.
  - R-R, R-M, M-M
  - What's Load/Store architecture ?

Other Taxonomy of Computer Architecture

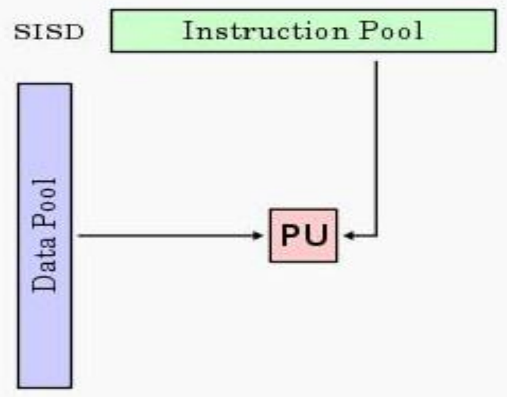
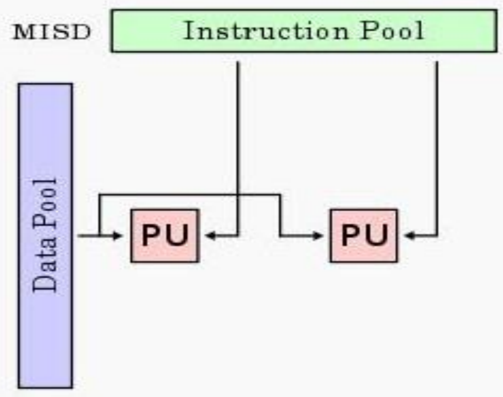
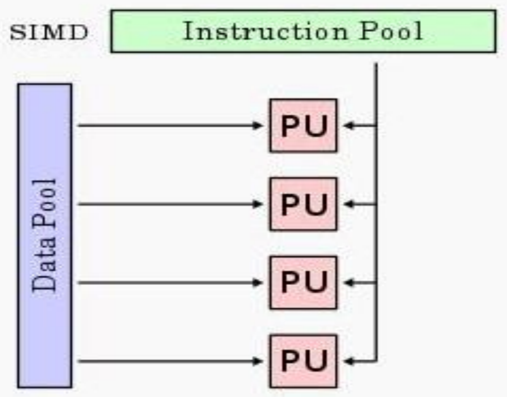
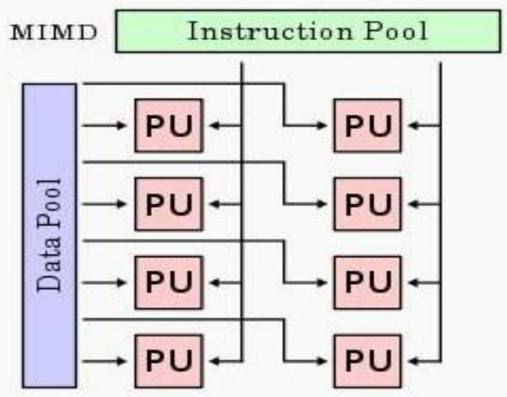
1989 IEEE taxonomy：

- Personal Computer (PC)
- WorkStation (WS)
- Mini Computer
- Mainframe
- Minisupercomputer
- Supercomputer

- Server
- Embedded systme
- Cluster、Grid Computer

Flynn taxonomy

- based on the number of streams of instructions and data on bottleneck

SISD	MISD : It is a type of parallel computing architecture where many functional units perform different operations on the same data.
	
SIMD	**MIMD**
	

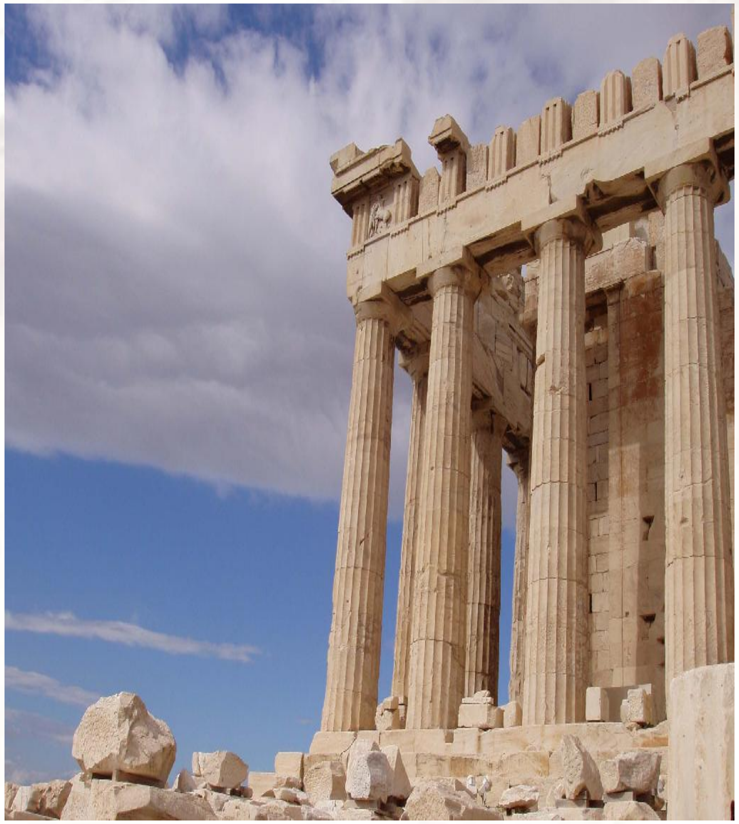
CSIC & RISC

- What's a RISC ?
- What are a RISC machine's characteristics ?
- Could you compare RISC & CISC ? From which viewpoint?

How you make decision when design ISA ?

- ❑ Measure the existing ISA
- ❑ Make the common case fast
  - ➢ Memory addressing mode:
    - aligned memory access
  - ➢ Oprand
  - ➢ Operation
  - ➢ Ecoding
    - Fixed, variable, Hybrid

Basic pipeline



What is Pipelining ?

- Pipelining:
  - is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements.

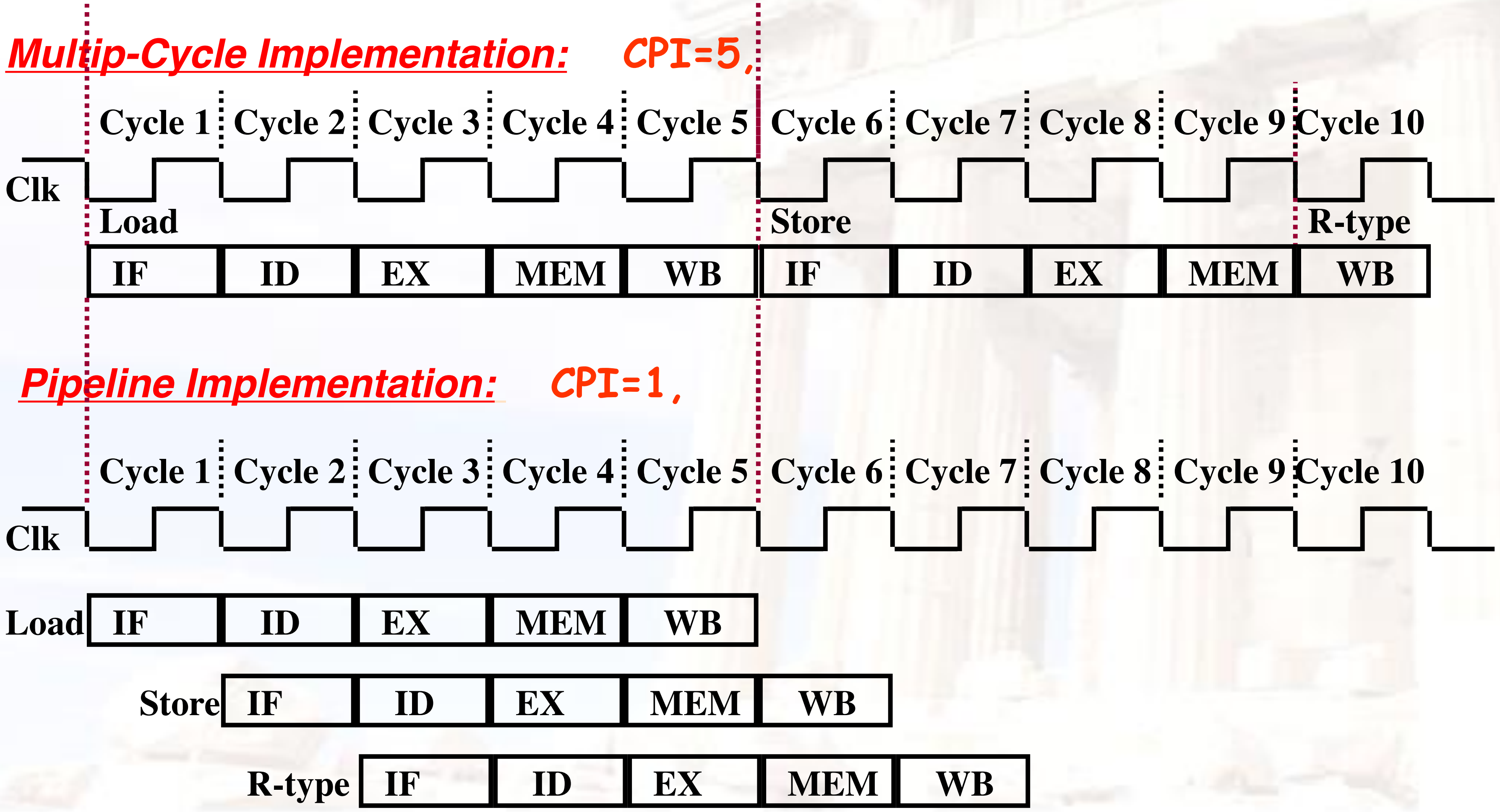
Instruction Pipeline

- An **instruction pipeline** is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput (the number of instructions that can be executed in a unit of time).
  - implementation technique whereby different instructions are overlapped in execution at the same time.
  - implementation technique to make fast CPUs

Why pipelining : conclusion

- The key implementation technique used to Make fast CPU: decrease CPUtime.
- Improving of Throughput (rather than individual execution time)
- Improving of efficiency for resources (functional unit)

Multi-cycle implementation vs. pipelining



Ideal Performance for Pipelining

- If the stages are perfectly balanced, The time per instruction on the pipelined processor equal to:

$$\frac{\textit{Time per instruction on unpipelined machine}}{\textit{Number of pipe stages}}$$

- So, **Ideal speedup equal to**

  Number of pipe stages.

Pipeline hazard: the major hurdle

- **A hazard** is a condition that prevents an instruction in the pipe from executing its next scheduled pipe stage
- Taxonomy of hazard
  - **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
  - **Control hazards**
    - branch condition and the branch PC are not available in time to fetch an instruction on the next clock

What we need to know ?

- What results in the various hazards ?
  - The causes
  - How to judge whether there is hazard ?
- The approaches to solve the hazards.
- How to evaluate the performance considering the hazards ?

5-cycle MIPS CPU

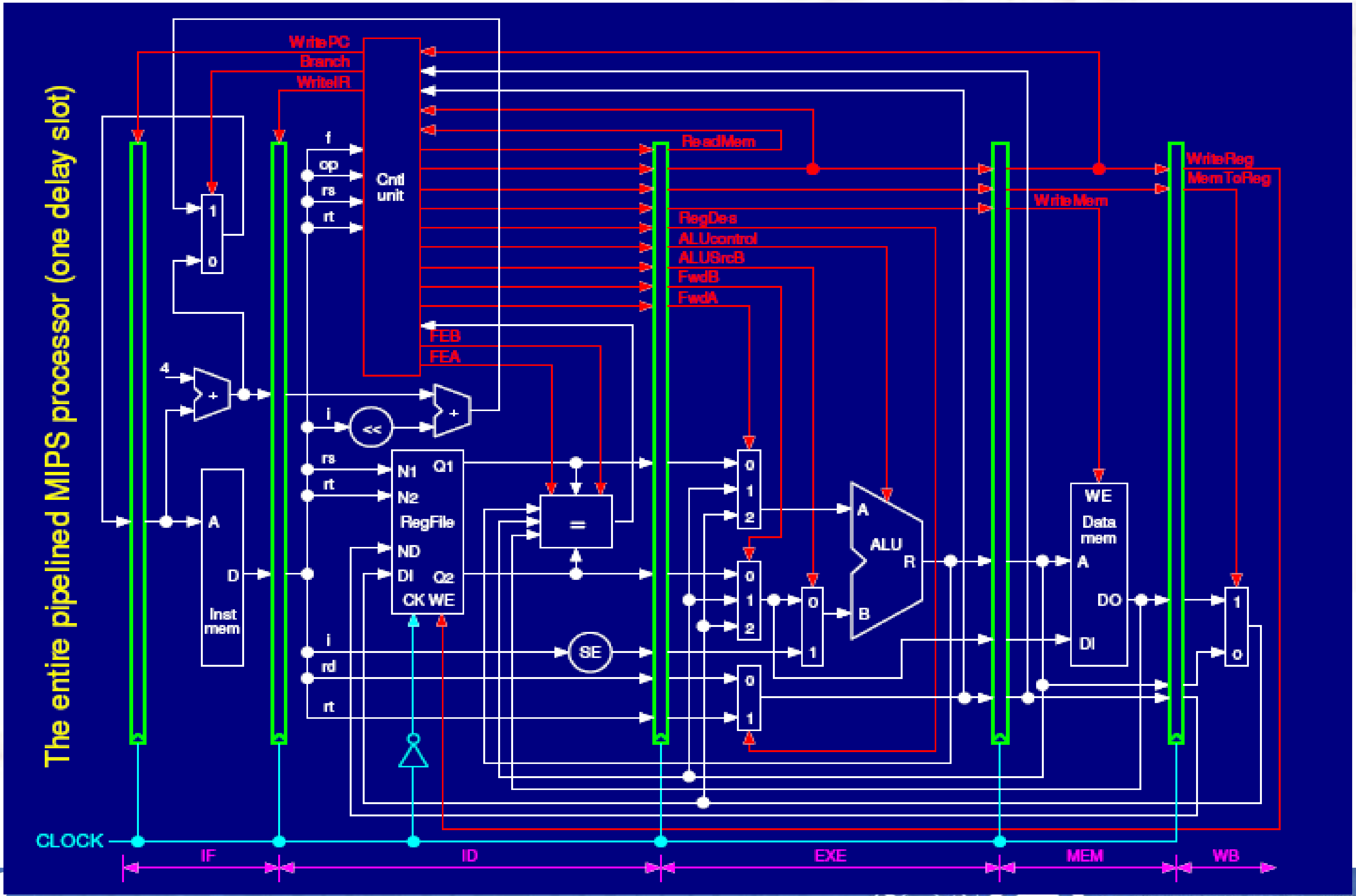
Steps	Rtype	lw/sw	beq	j

Instruction fetch (IF)	IR=Memory[PC]
	PC=PC+4			

Instruction decode and register fetch (ID)	A=Reg[rs]
	B=Reg[rt]
	C=PC+(sign-extend(imm)<<2)			

Execution (EXE)	C=A op B	C=A+ sign-extend(imm)	if (A—B)==0 then PC=C	PC=PC[31-28]|| (address<<2)
Memory access (MEM) or Rtype completion (WB)	Reg[rd]=C (WB)	lw: DR=Memory[C] sw: Memory[C]=B		
Memory access completion (WB)		lw: Reg[rt]=DR (WB)		

Pipeline with 1 delay slot and forwarding path and load stall



Could you write the logic expression ?

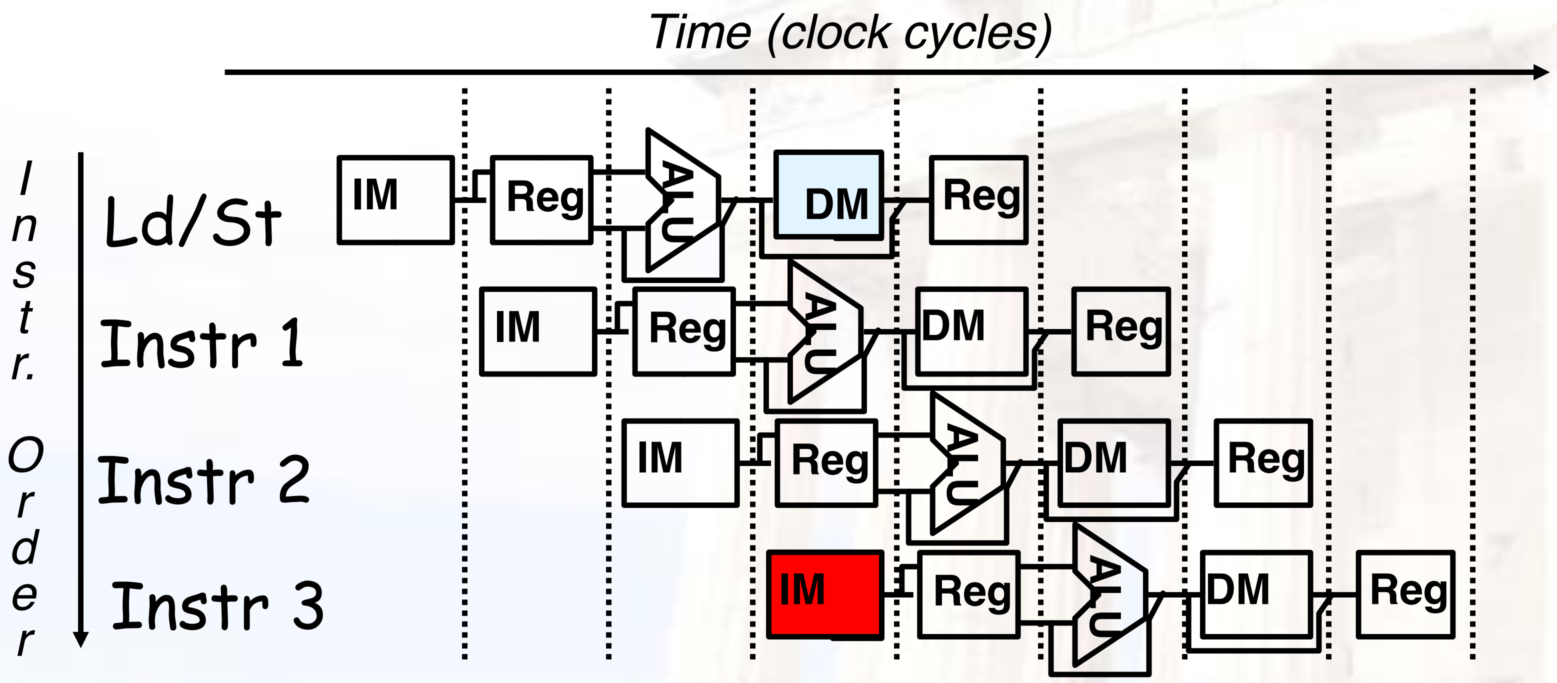
- LoadStall ?
- FowardA ?
- FowardB ?
- BranchStall ?
- When and How many stalls ? Examples ?
- Which Forwarding path will be used ?
- How to implement “stall” ?

- What if give an 7-stage pipeline ?

Structural hazard: Pipe Stage Contention

- **Structural hazards**
  - Occurs when two or more instructions want to use the same hardware resource in the same cycle
  - Causes bubble (stall) in pipelined machines
  - Overcome by replicating hardware resources
    - Multiple accesses to the register file
    - Multiple accesses to memory
    - some functional unit is not fully pipelined.
    - Not fully pipelined functional units

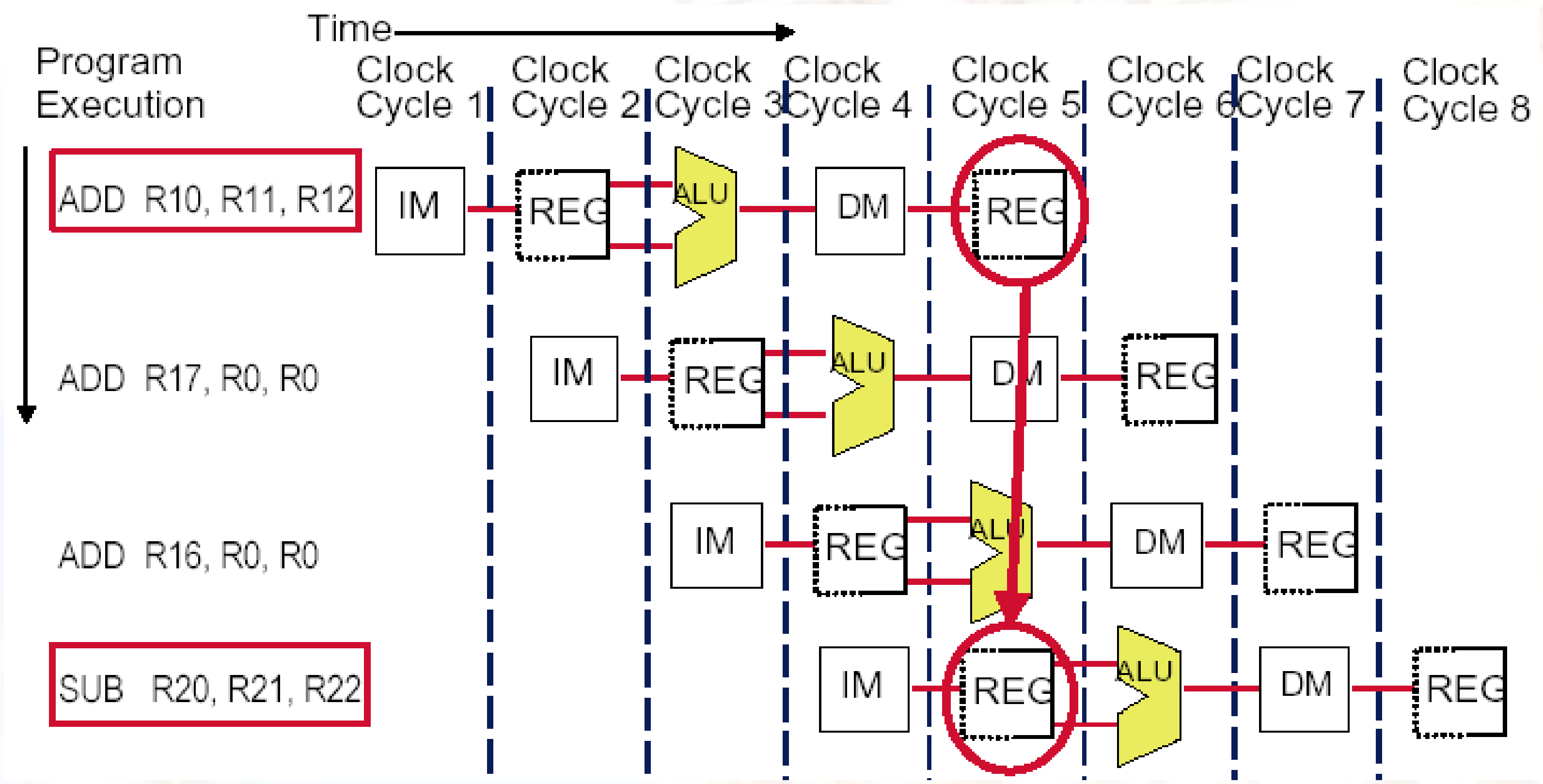
Split instruction and data memory



- <u>Split instruction and data memory</u> / <u>multiple memory port</u> / <u>instruction buffer</u> means:

  fetch the instruction and data inference using different hardware resources.

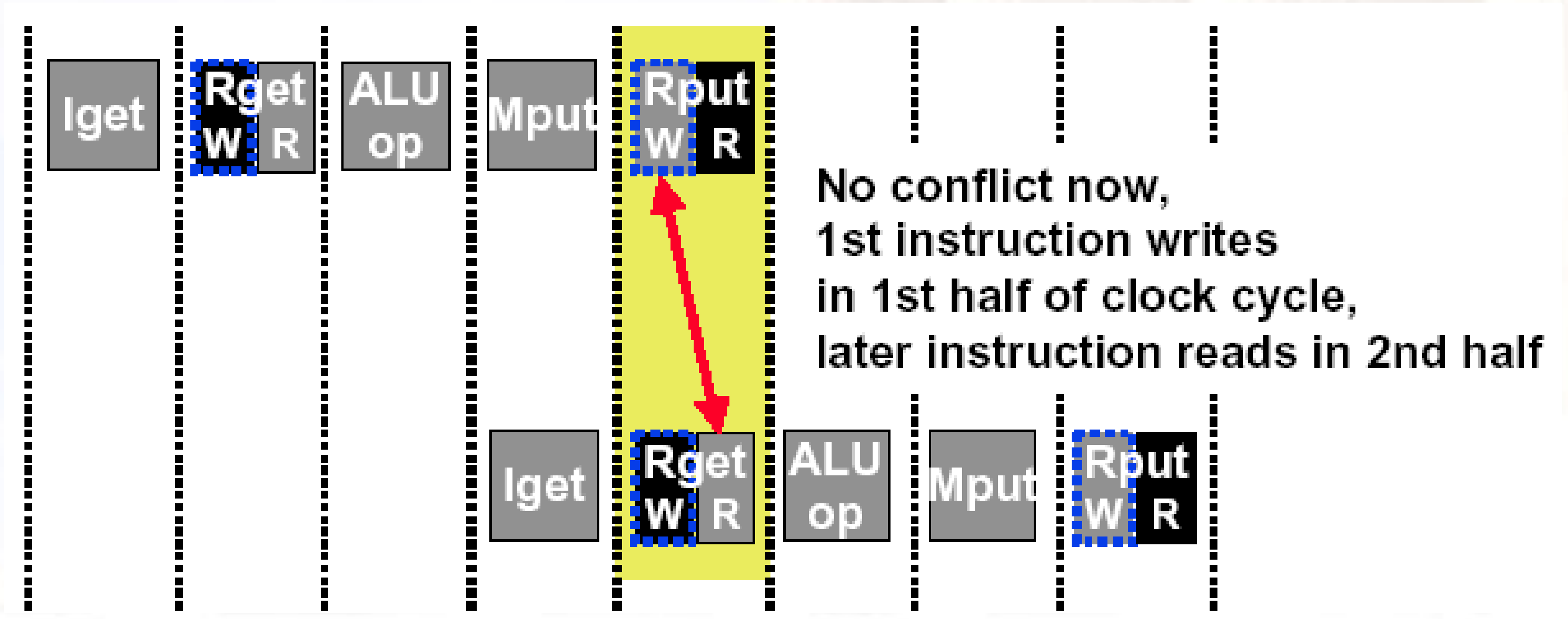
Multi access to the register file



- Simply insert a stall， speedup will be decreased.
- We can resolve it with “ double bump”

Double Bump Works !

- allow WRITE-then-READ in one clock cycle (double bump)



No conflict now,
1st instruction writes
in 1st half of clock cycle,
later instruction reads in 2nd half

Not fully pipelined function unit : may cause structural hazard

Unpipelined Float Adder

ADDD	IF	ID	ADDD						WB	
ADDD		IF	ID	stall	stall	stall	stall	stall	ADDD	

Not fully pipelined Adder

ADDD	IF	ID	A1		A2		A3		WB	
ADDD		IF	ID	stall	A1		A2		A3	

Resolved using fully pipelined Adder

ADDD	IF	ID	A1	A2	A3	A4	A5	A6	WB	
ADDD		IF	ID	A1	A2	A3	A4	A5	A6	WB

Or multiple unpipelined Float Adder

ADDD	IF	ID	ADDD1						WB	
ADDD		IF	ID	ADDD2						WB

Data hazard

- Data dependence = Data hazard ?
  - RAW, WAR, WAW

- How to resolve ?
  - Double bump
  - Forwarding path
  - Load stall
  - Compiler scheduling
  - Dynamic scheduling, Register renaming

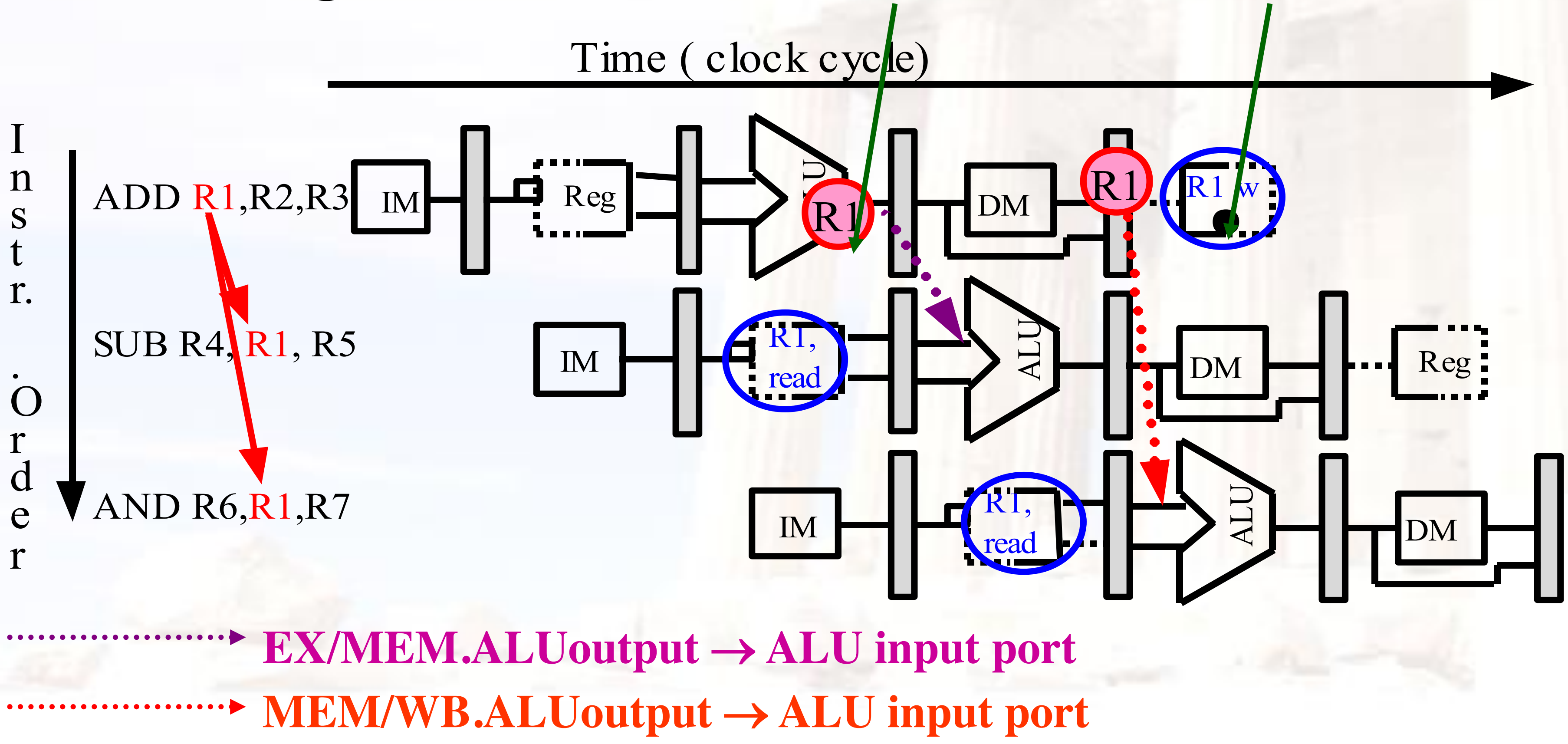
Data hazard

- Data hazards occur when the pipeline changes the order of read/write accesses to operands comparing with that in sequential executing .
- Let's see an Example

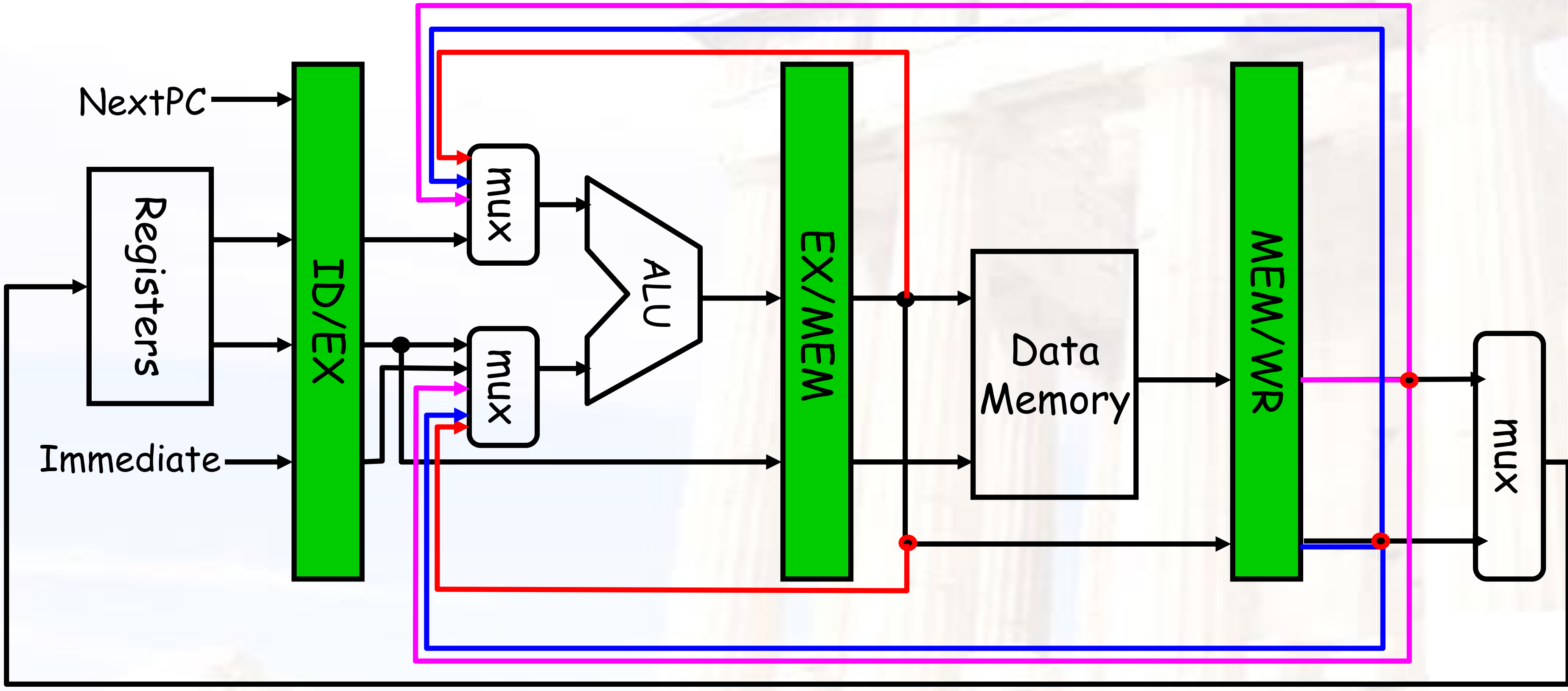
```
DADD R1,   R1, R3
DSUB R4,   R1, R5
AND  R6,   R1, R7
OR   R8,   R1, R9
XOR  R10,  R1, R11
```

Forwarding:reduce data hazard stalls

- Data may be already computed - just not in the Register File



Hardware Change for Forwarding

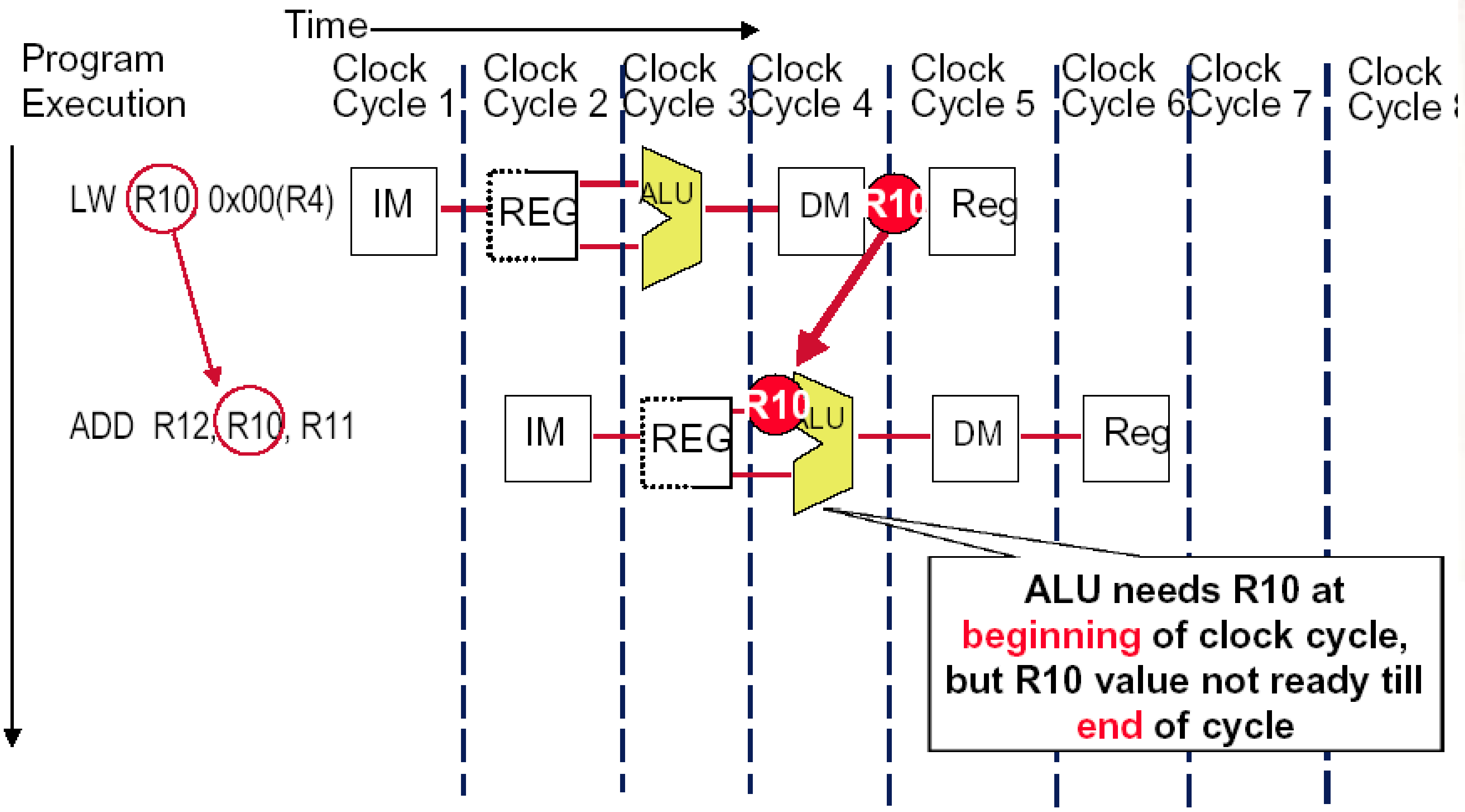


🟥 EX/Mem.ALUoutput → ALU input

🟥 MEM/WB.ALUoutput → ALU input

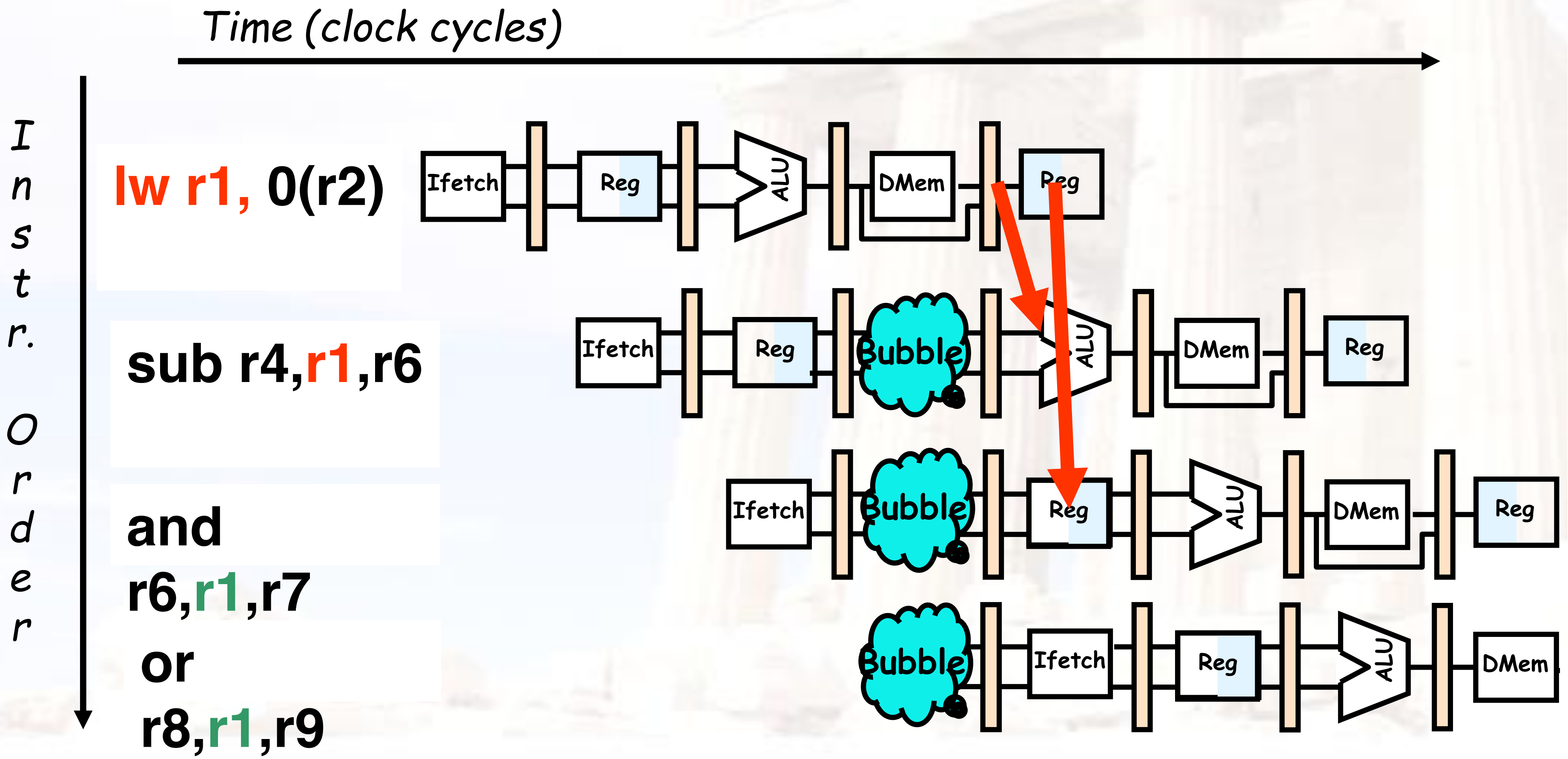
🟪 MEM/WB.LMD → ALU input

Forwarding Doesn't Always Work



ALU needs R10 at beginning of clock cycle, but R10 value not ready till end of cycle

So we have to insert stall: Load stall

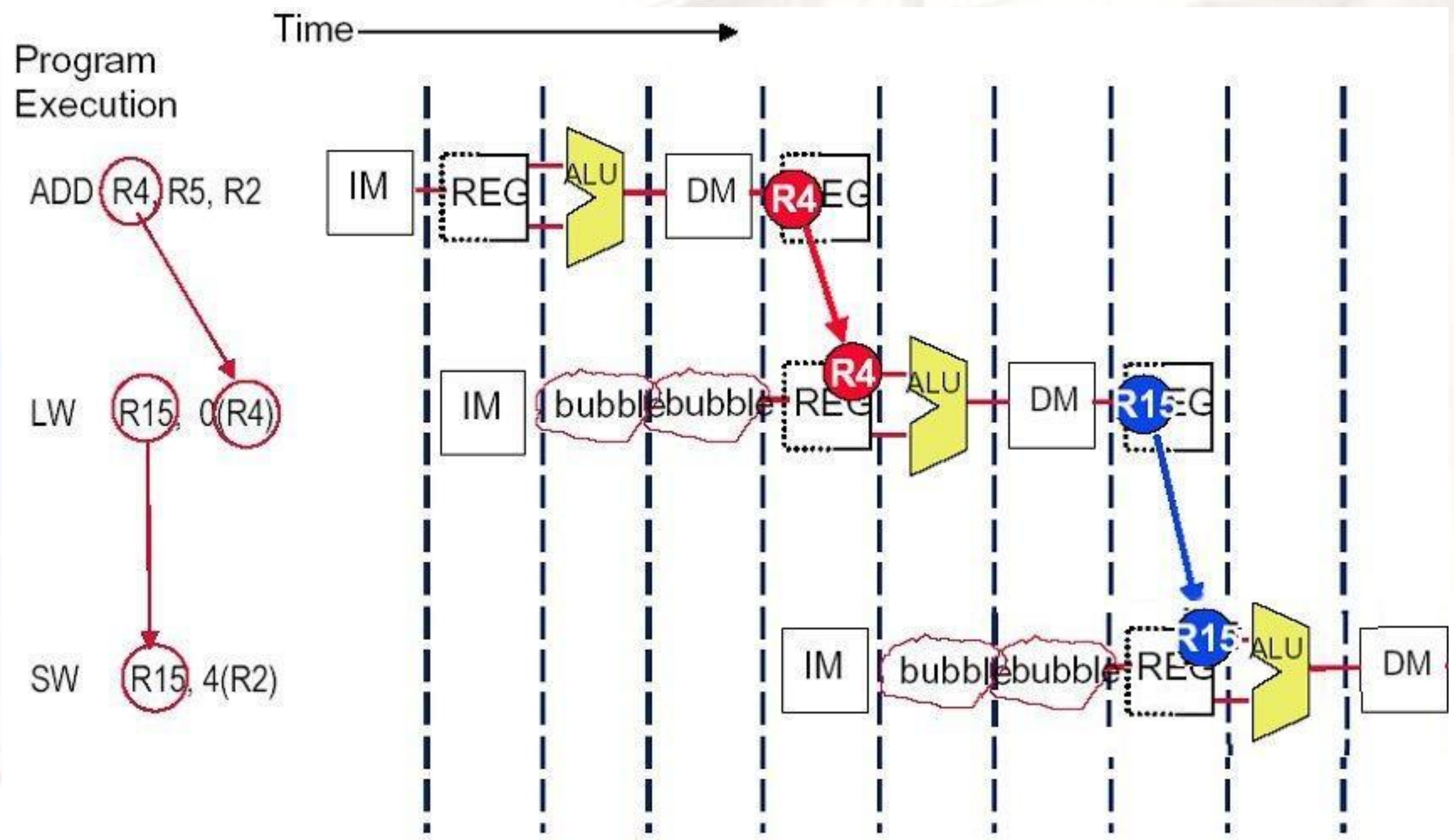


Example of Forwarding and Load Delay

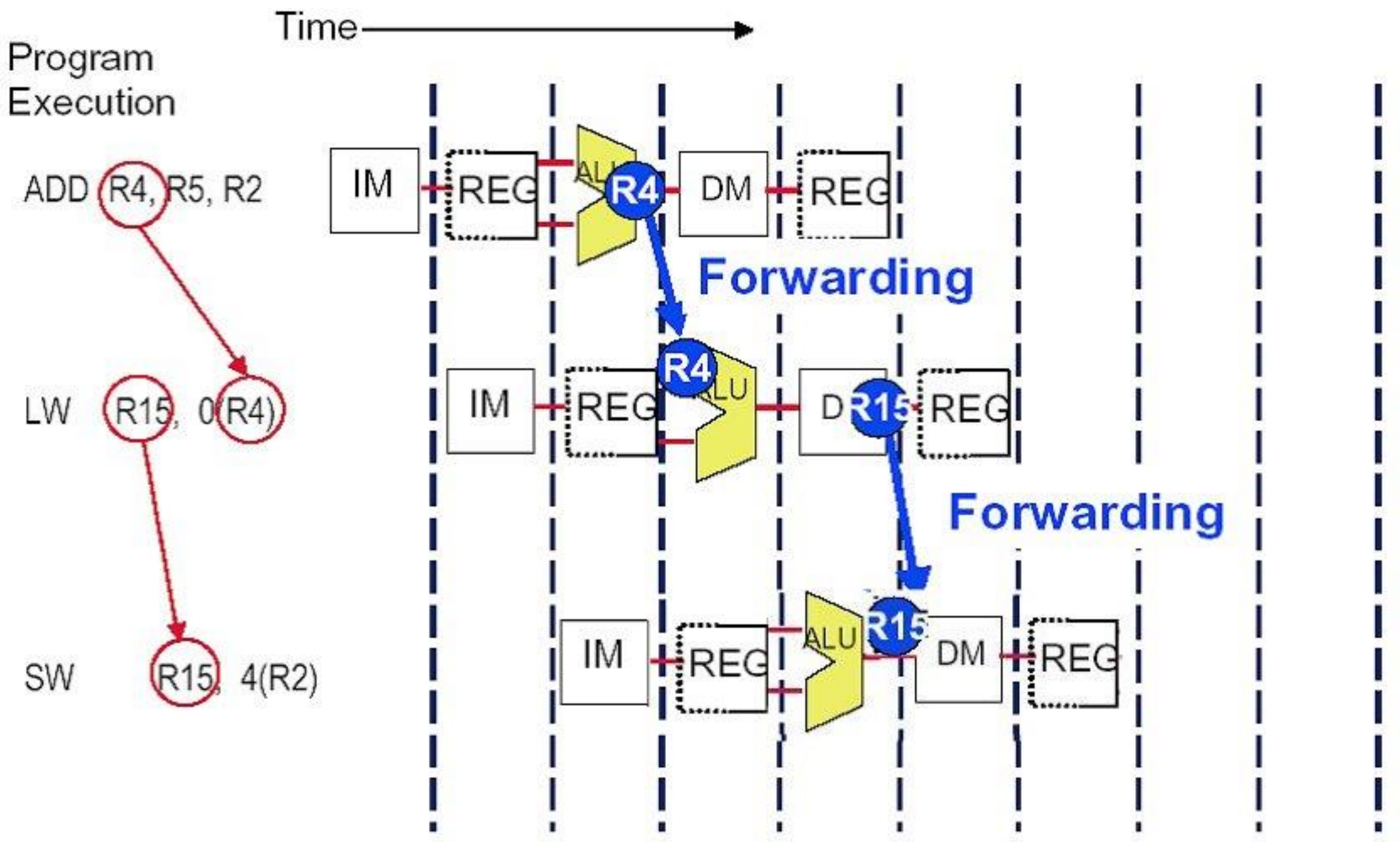
- Why forwarding?
  - ADD R4, R5, R2
  - LW R15, 0(R4)
  - SW R15, 4(R2)

- Why load delay?
  - ADD R4, R5, R2
  - LW R15, 0(R4)
  - SW R15, 4(R2)

Solution (without forwarding)



Solution (with forwarding)



EX/MEM.ALUoutput → ALU input port
MEM/WB.ALUoutput → DM data write port

Instruction reordering by compiler to avoid load stall

- Try producing fast code for

  a = b + c;

  d = e – f;

  assuming a, b, c, d ,e, and f in memory.
- **Slow code:**

```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

```
          LW    Rb,b
          LW    Rc,c
          LW    Re,e
          ADD
Ra,Rb,Rc
          LW    Rf,f
          SW    a,Ra
          SUB
Rd,Re,Rf
          SW    d,Rd
```

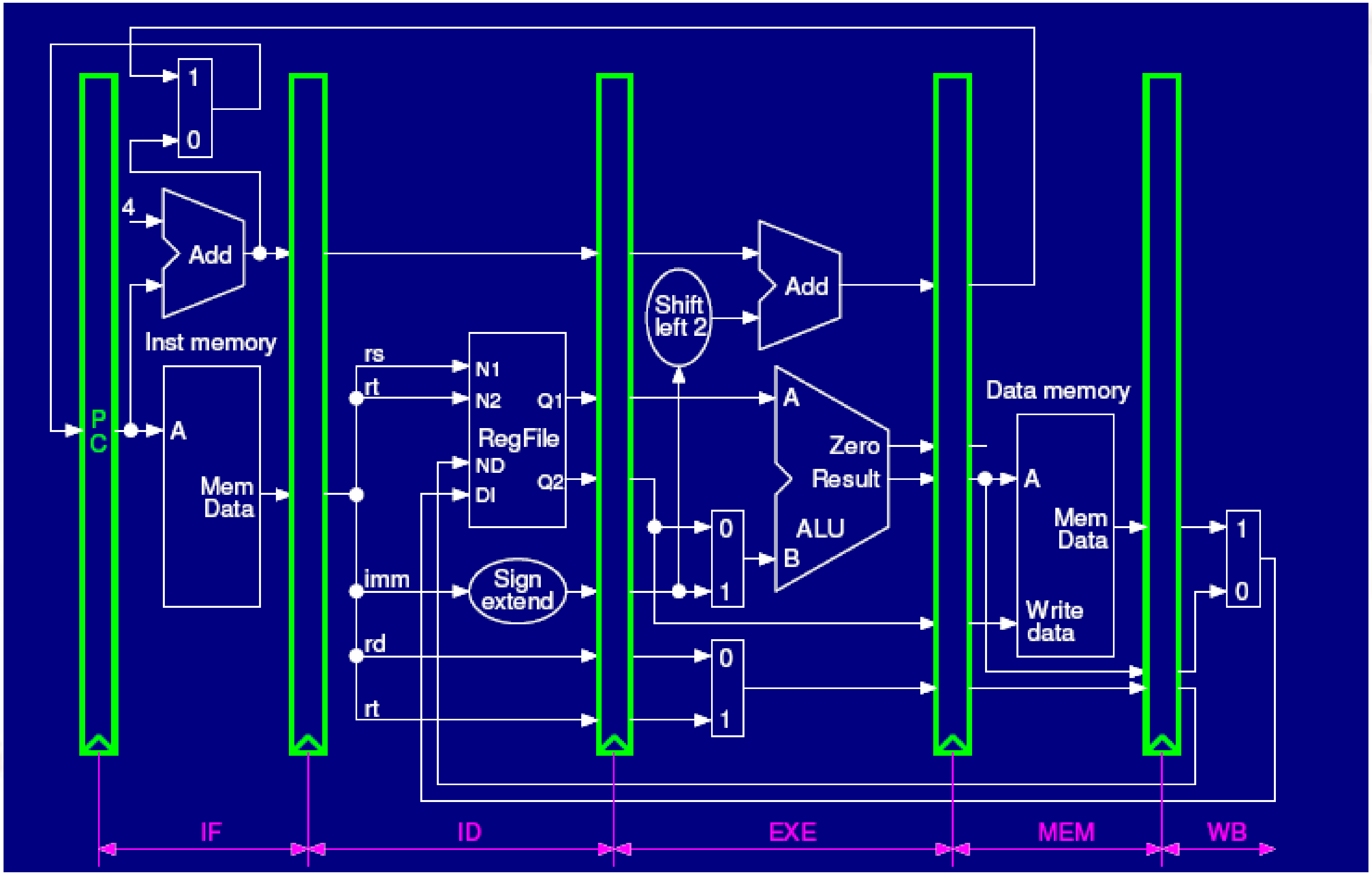
Deal with Control Hazards

- Flushing the pipeline
- Predict-taken
- Predict-not-taken
- Delayed branch

- Resolve the branch as earlier as possible

- Branch predict , Specluation

Deal with Control Hazards



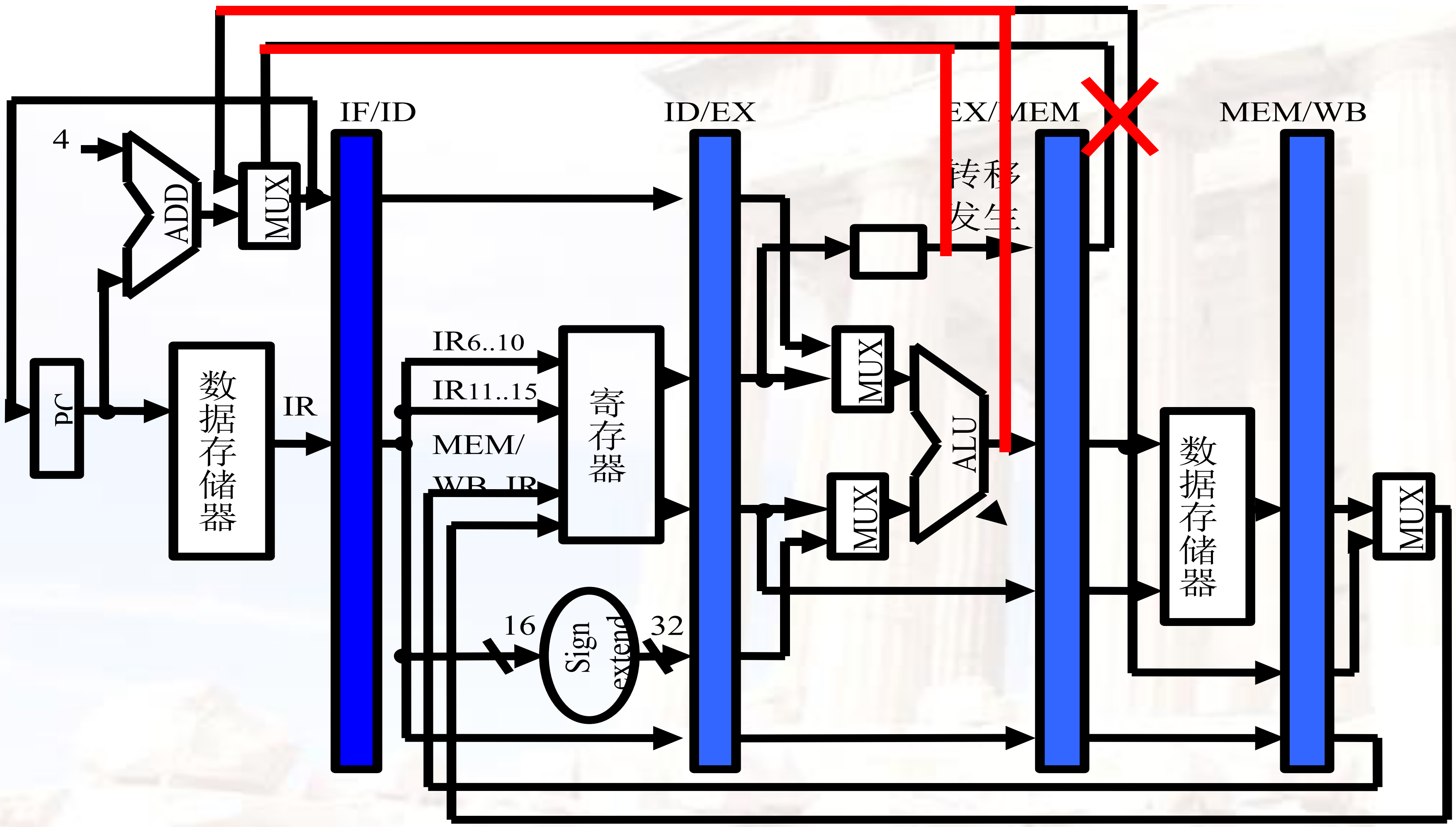
Flushing the pipeline

- Simplest hardware:
  - Holding or deleting any instruction after branch until the branch destination is know.
  - Penalty is fixed.
  - Can not be reduced by software.

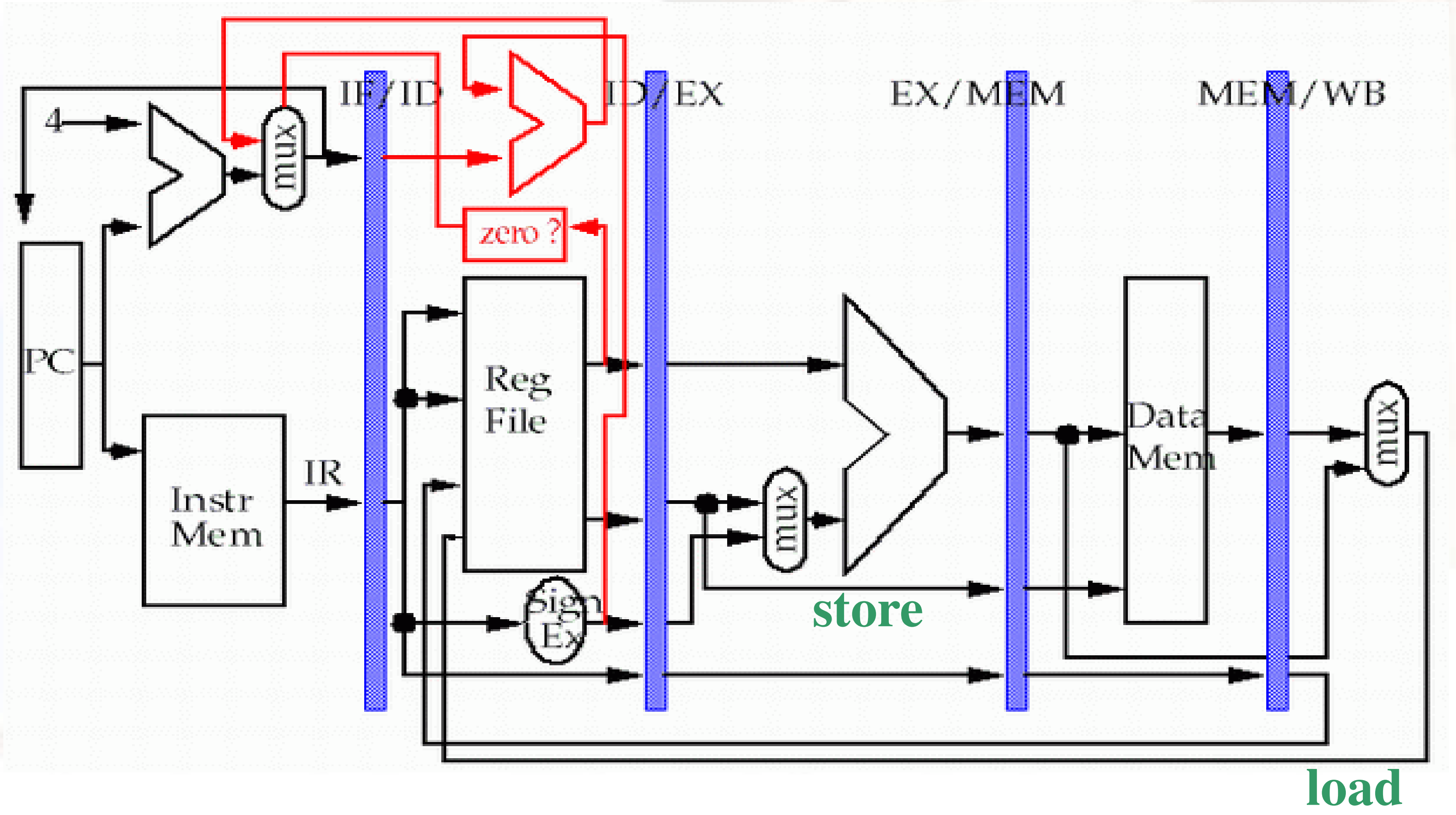
The pipeline status --flushing

Branch instruction	IF	ID	EX	MEM	WB		
Branch Successor		IF	stall	stall	idle	idle	
Branch successor+1					IF	ID	EX
Branch successor+2						IF	ID
Branch successor+3							IF

Move the Branch Computation Forward



Move the Branch Computation more Forward



Why “waste” the fetched instruction ?

Branch instruction	IF	ID	EX	MEM	WB		
Branch Successor		IF	idle	idle	idle	idle	
Branch successor+1			ID	EX	MEM	WB	ID
Branch successor+2				ID	EX	MEM	WB
Branch successor+3					ID	EX	MEM

- We have fetched the instruction 48, why should we waste the delay slot ?

Delayed branch

- Good news
  - 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

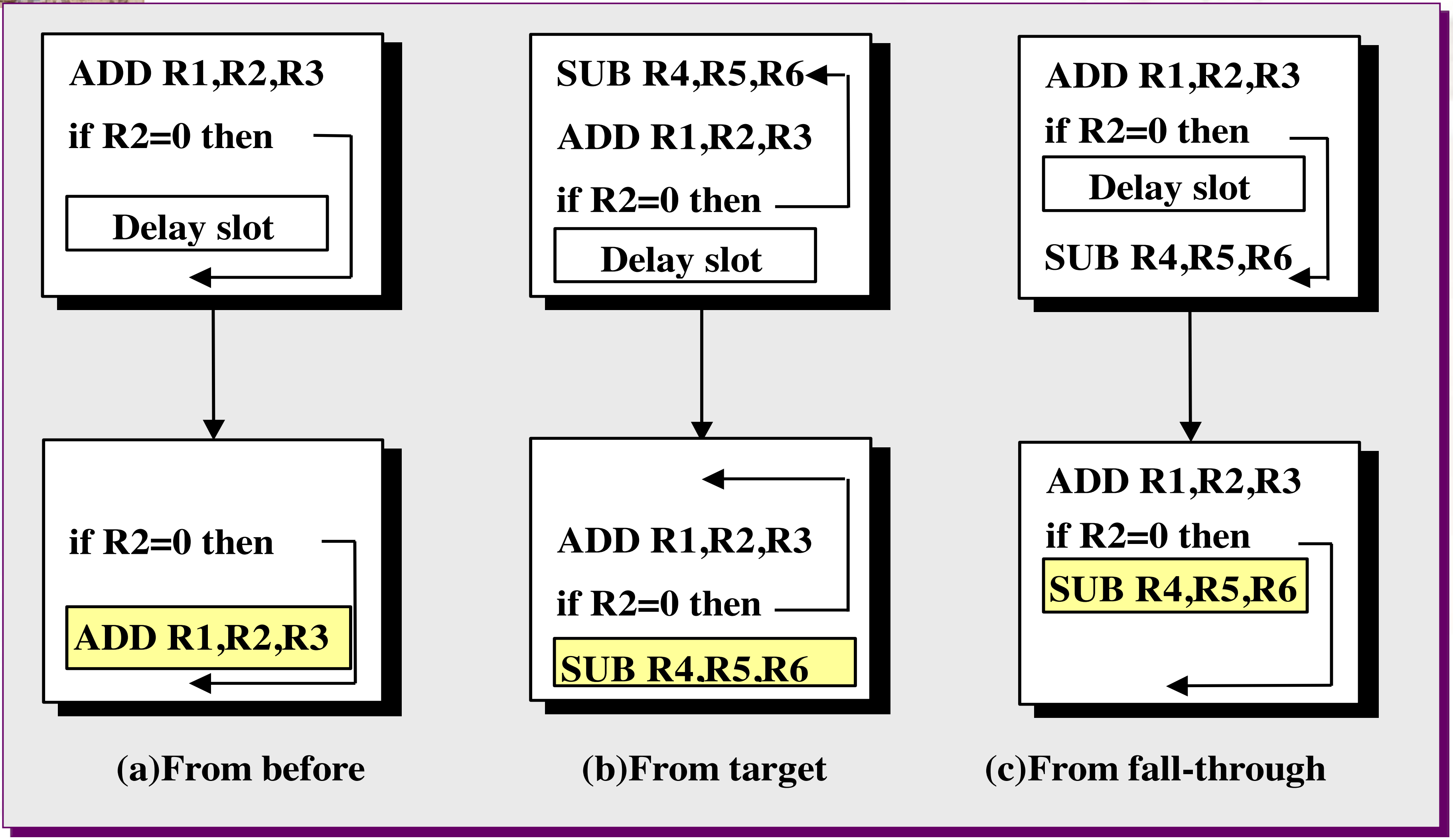
- Hence the name: branch delay slot

branch instruction
sequential successor$_1$
sequential successor$_2$
...
sequential successor$_n$
branch target if taken

Branch delay slots

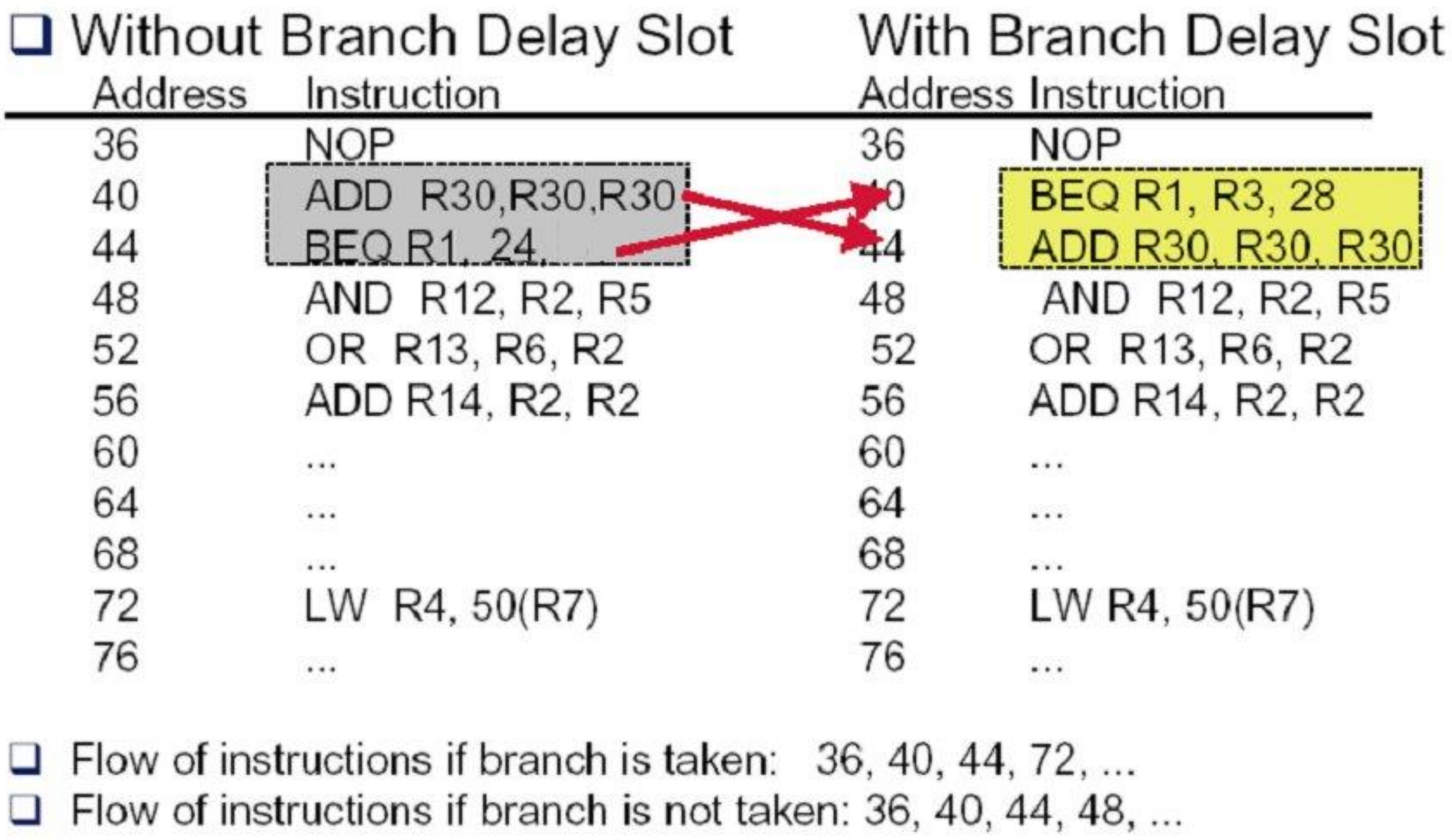
  - The instruction cycle after the branch is used for address calculation , 1 cycle delay necessary
  - SO…we regard this as a free instruction cycle, and we just DO IT
- Consequence
  - You (or your compiler) will need to adjust your code to put some useful work in that "slot", since just putting in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

How to adjust the codes?



(a)From before (b)From target (c)From fall-through

Example: rewrite the code (a)



❑ Without Branch Delay Slot		With Branch Delay Slot	
Address	Instruction	Address	Instruction
36	NOP	36	NOP
40	ADD R30,R30,R30	40	BEQ R1, R3, 28
44	BEQ R1, 24	44	ADD R30, R30, R30
48	AND R12, R2, R5	48	AND R12, R2, R5
52	OR R13, R6, R2	52	OR R13, R6, R2
56	ADD R14, R2, R2	56	ADD R14, R2, R2
60	...	60	...
64	...	64	...
68	...	68	...
72	LW R4, 50(R7)	72	LW R4, 50(R7)
76	...	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
```

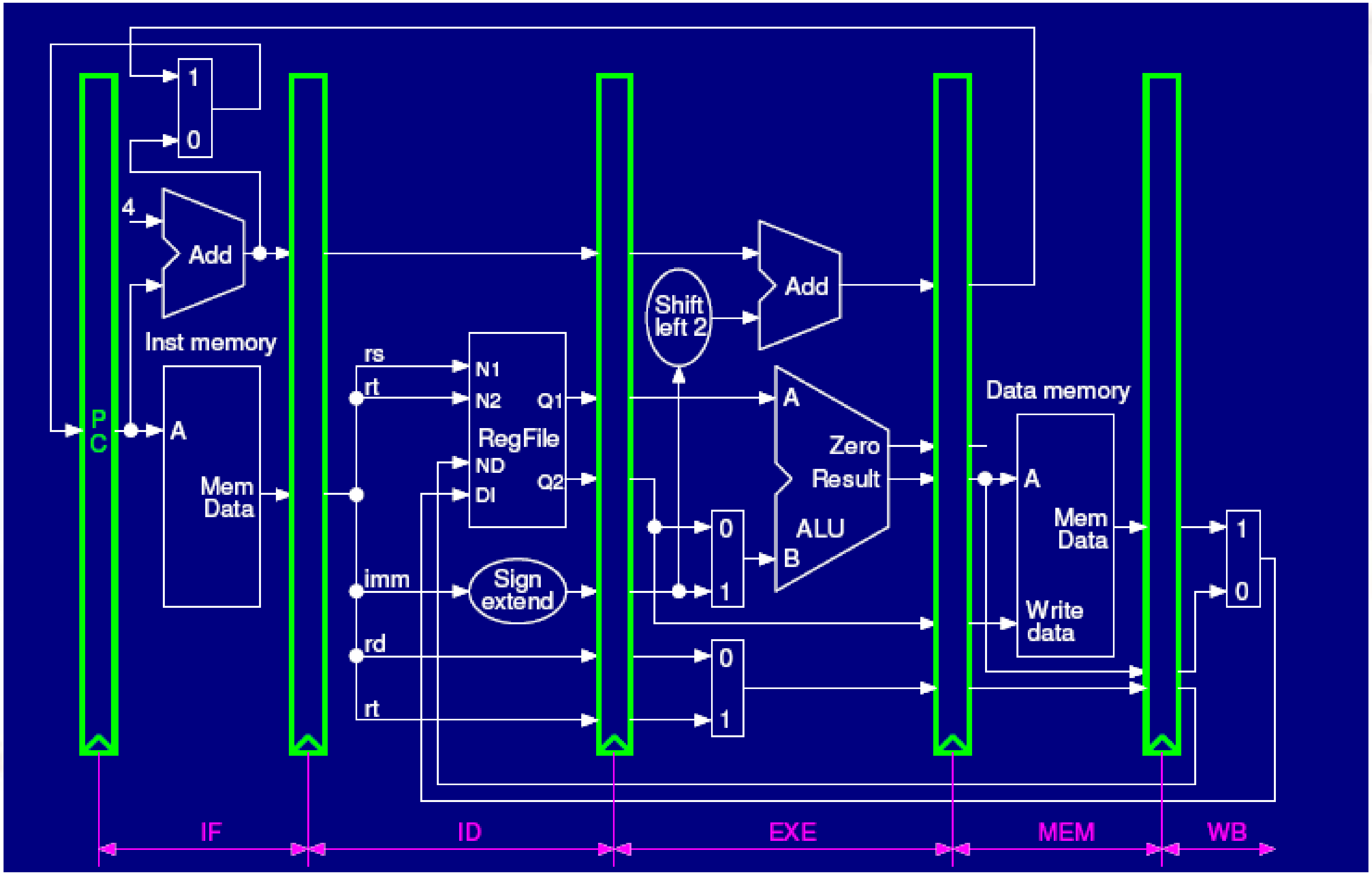
```
         LW      R2, 0(R1)
Loop:    ADD     R3, R2, R4
         SW      R3, 0(R1)
         ......
         SUB  R1,R1, #4
         BNEZ R1, Loop
         LW      R2, 0(R1)
```

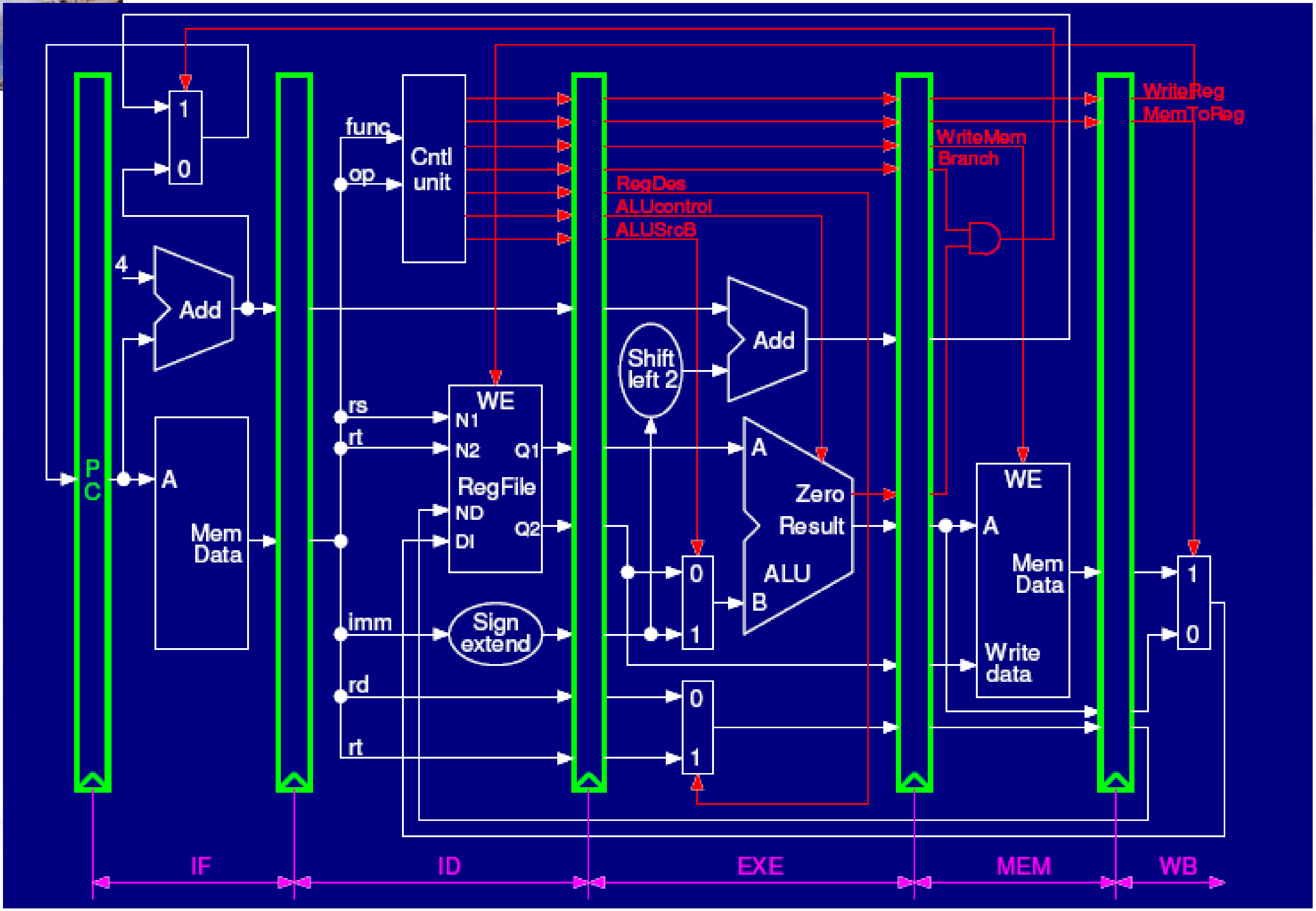
Example: rewrite the code (b-2)

```
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)
```

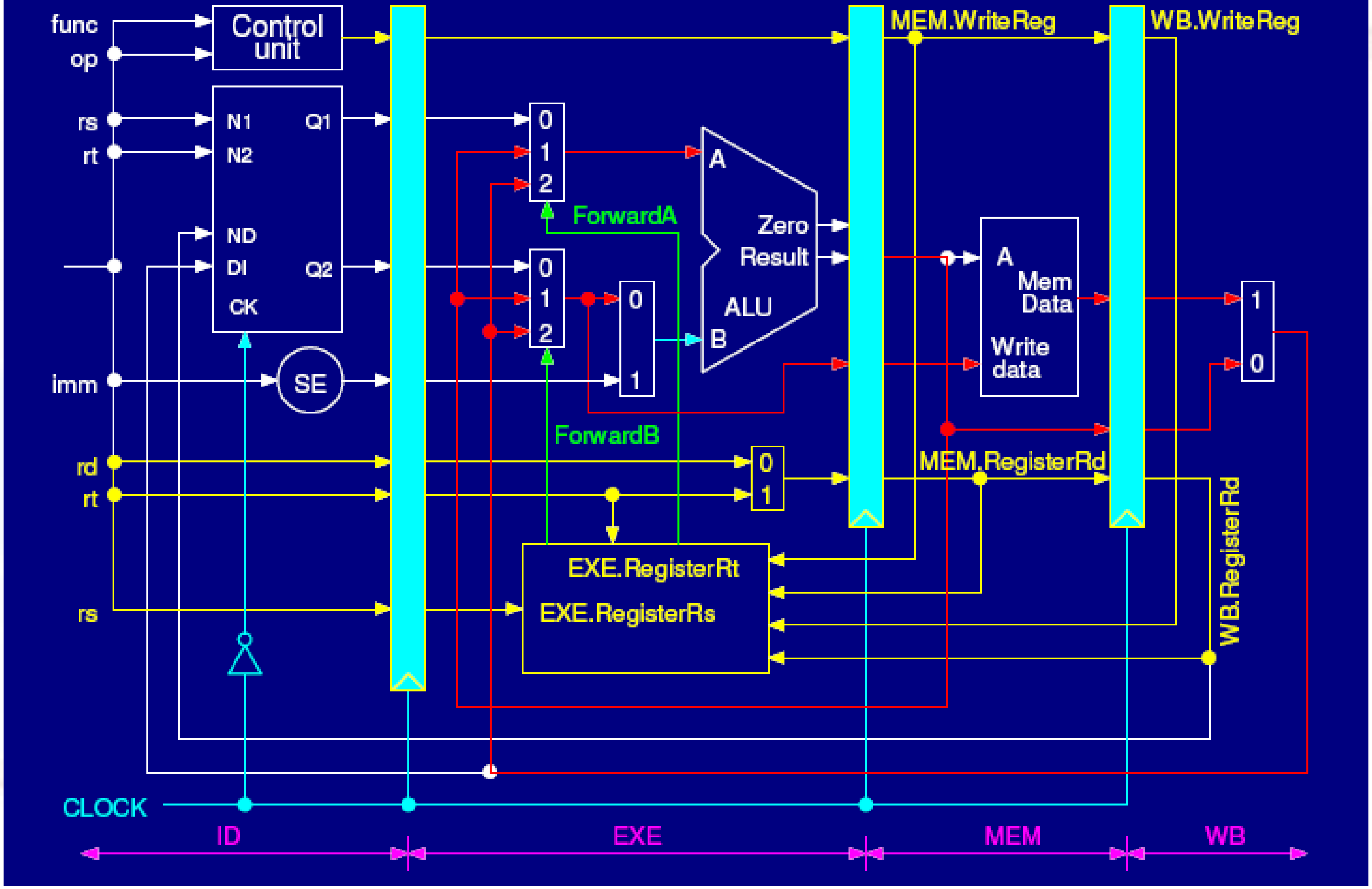
MIPS Pipeline



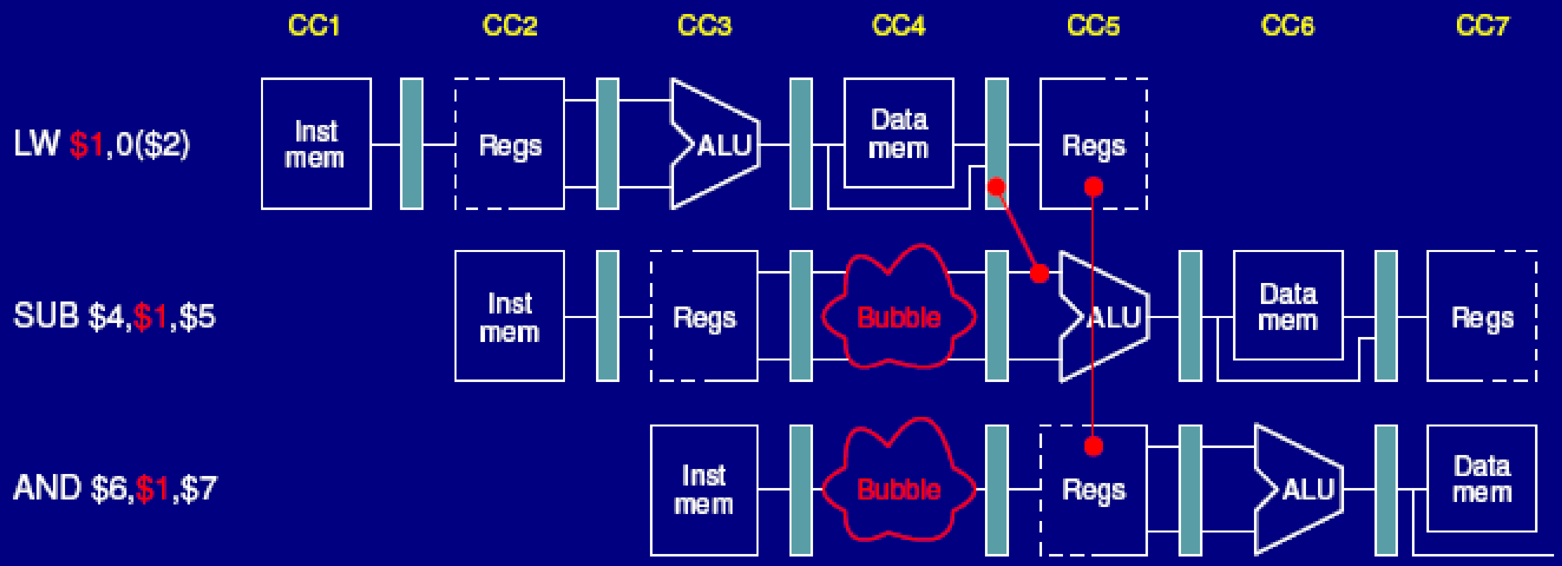


WriteReg
MemToReg
WriteMem
Branch
RegDes
ALUcontrol
ALUSrcB
func
op
Cntl unit
1
0
4
Add
P C
A
Mem Data
rs
rt
WE
N1
N2
Q1
RegFile
ND
Q2
DI
imm
Sign extend
rd
rt
Shift left 2
Add
A
Zero
Result
ALU
B
0
1
0
1
WE
A
Mem Data
Write data
1
0
IF
ID
EXE
MEM
WB

Internal Forwarding

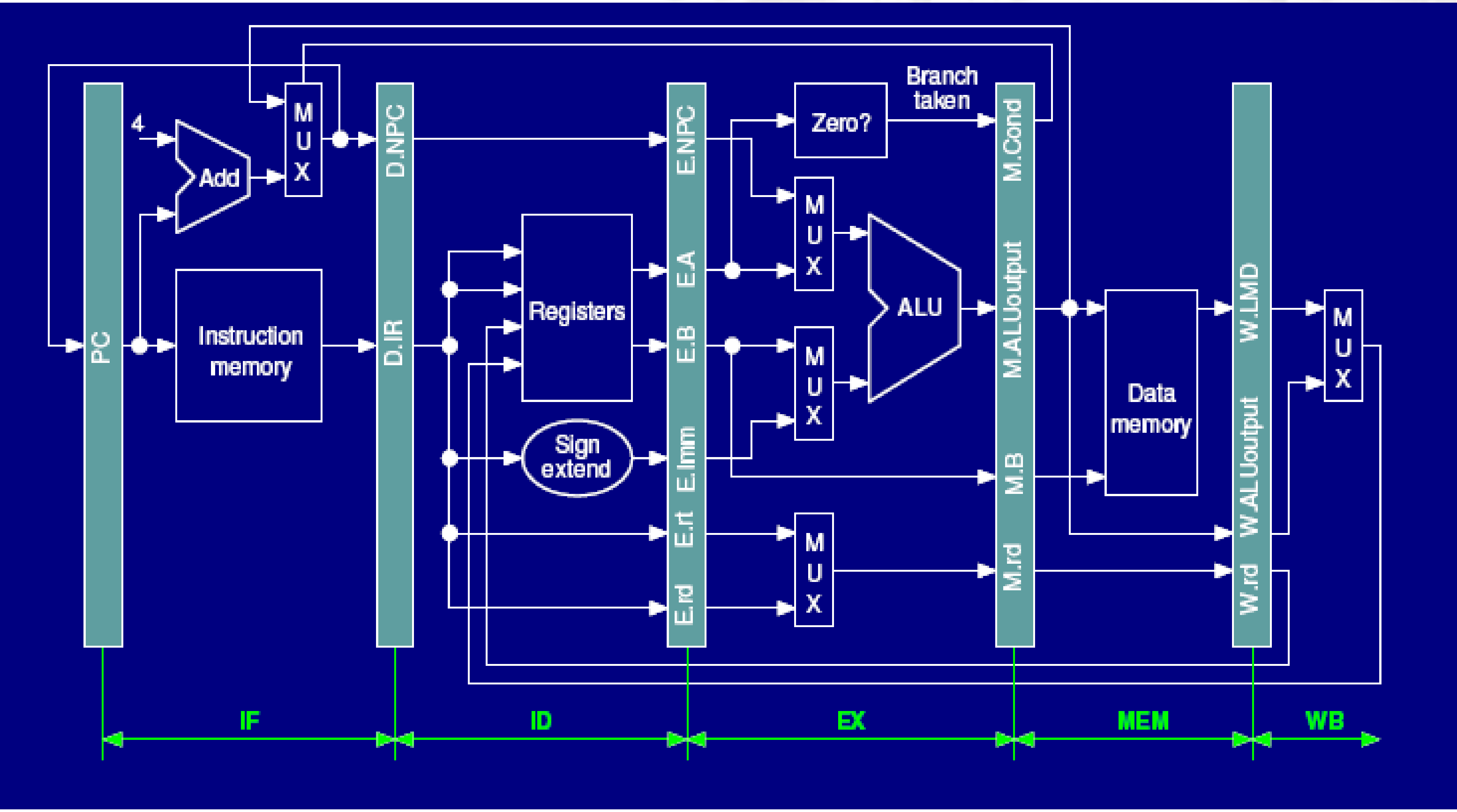


Load stall even with forwarding path

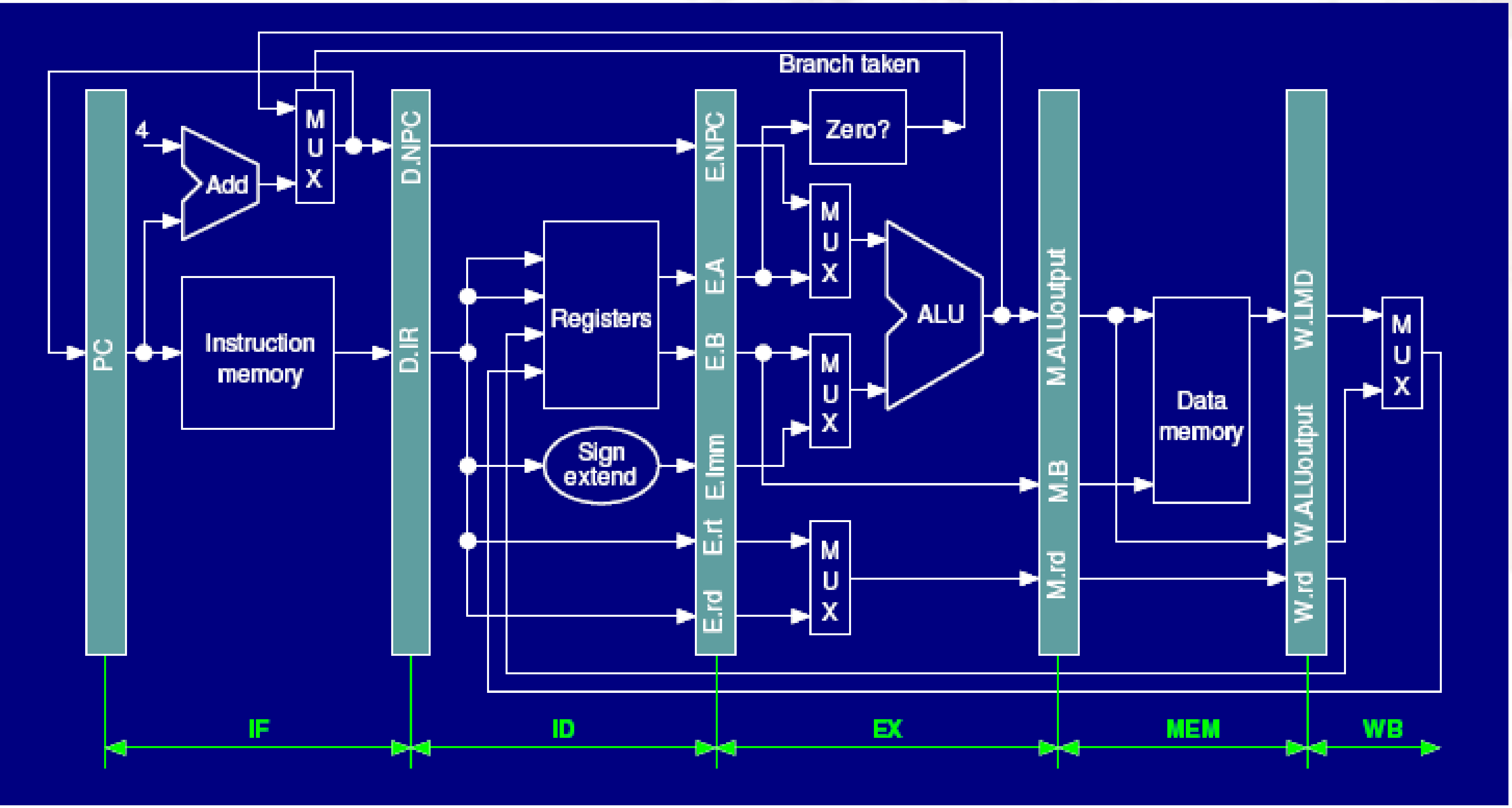


To stall the pipeline, we can generate two signals (WritePC and WriteIR) to disable the writing to PC and the ID pipeline register (IR and PC+4).

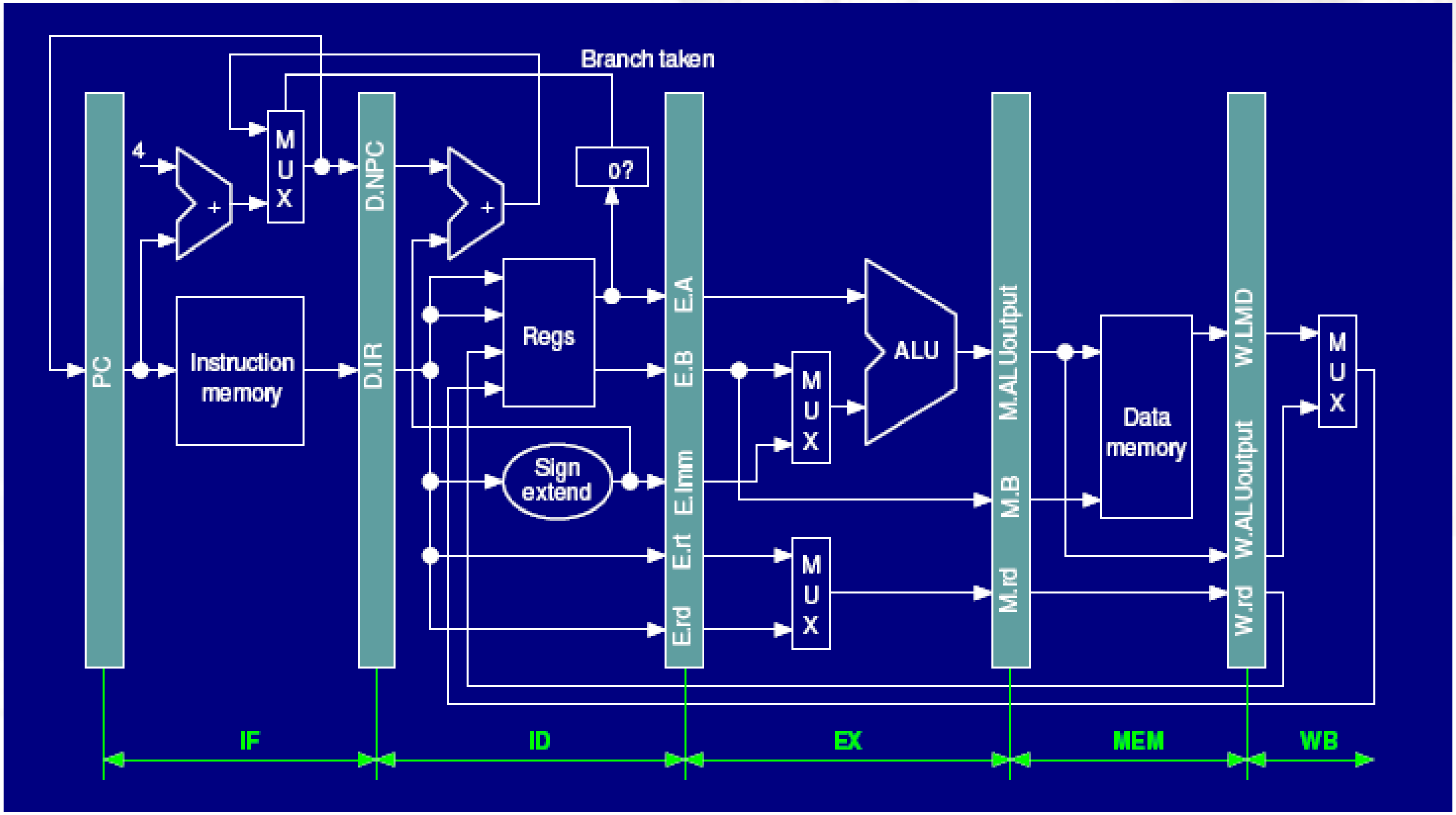
Pipeline with 3 branch delay slot



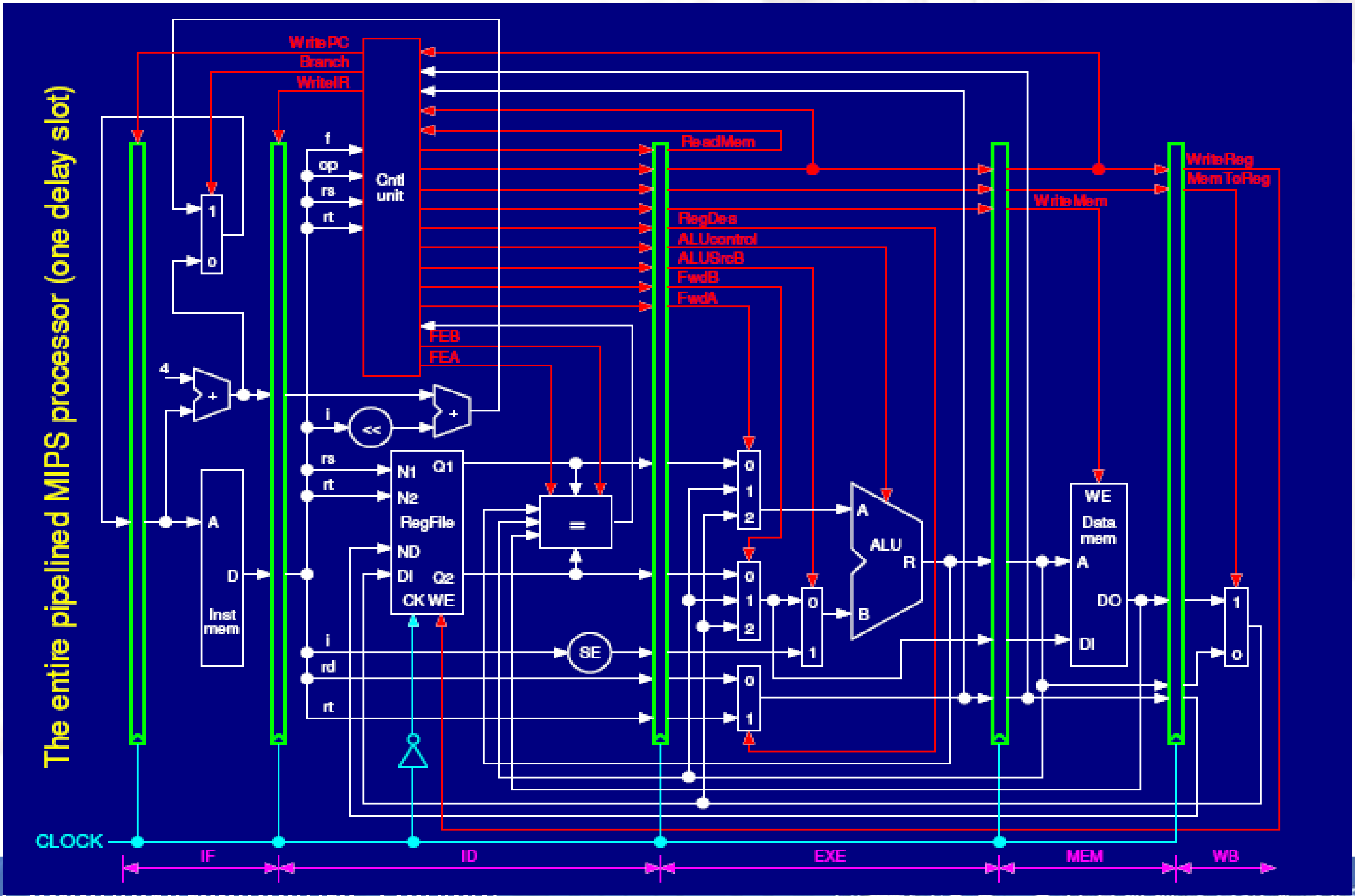
Pipeline with 2 branch delay slot



Pipeline with 1 branch delay slot



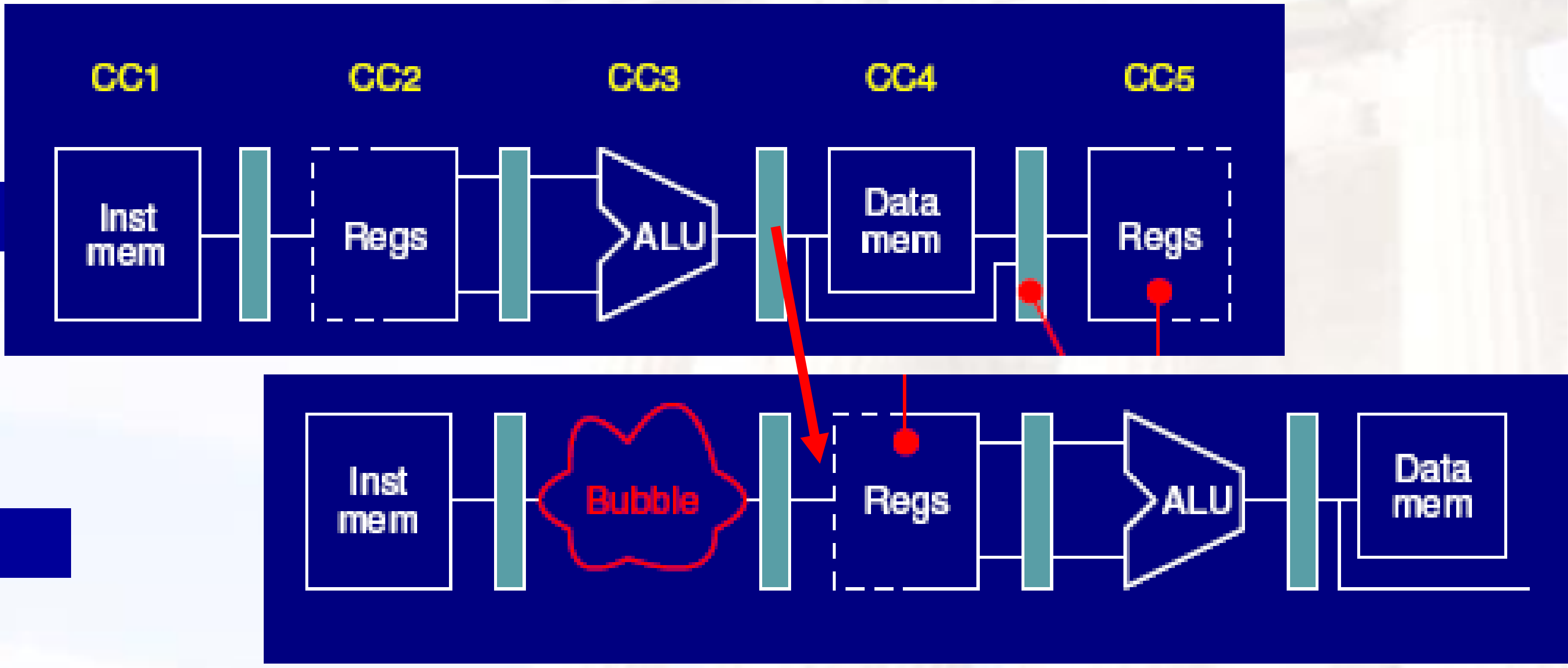
Pipeline with 1 delay slot and forwarding path and load stall



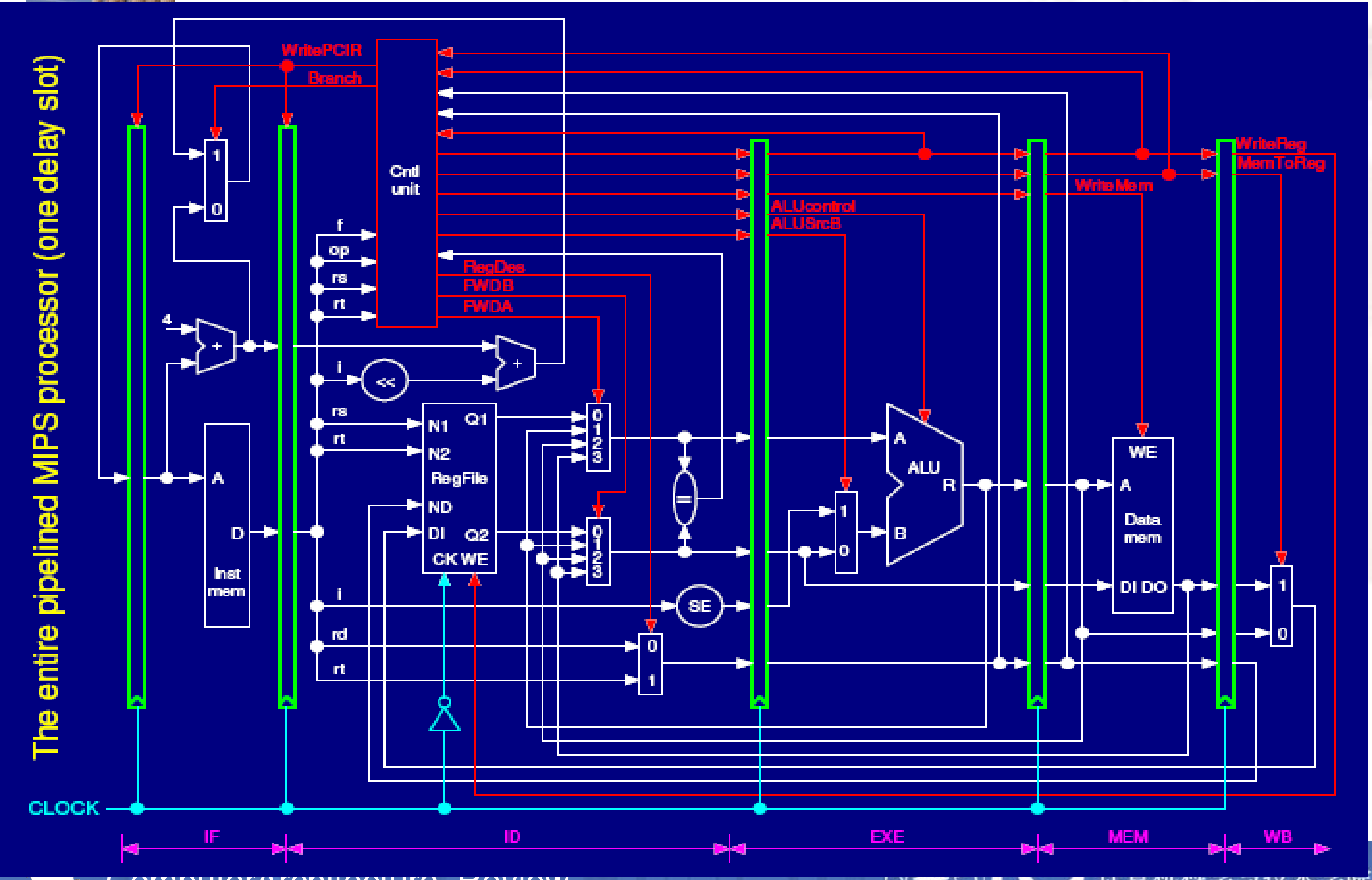
Still need to stall sometime

SUB $4, $1, $5

BEQ $4, $2, L1



Move forwarding to ID



Extending 5-MIPS pipeline to deal with multi-cycle operations

- Is there WAR hazard or WAW hazard ?
- Could you judge when and what hazard ?

Are you clear about these terms ?

36 terms of Cache

Cache	Virtual memory	
data cache	Instruction cache	unified cache
block	page	tag field
Block address	index field	block offset
full associative	set associative	**direct mapped**
n-way set associative	set	address trace
misses per instruction	Memory stall cycles	**miss penalty**
Valid bit	**dirty bit**	**locality**
cache hit	hit time	
cache miss	miss rate	page fault
Write through	write back	**write allocate**
random replacememt	least-recently used	**no-write allocate**
Average memory access time	write buffer	write stall

Memory Hierarchy (C.1)

- What's Memory hierarchy ? Why ?
- Basic four questions for Memory Hierarchy
  - Block placement
    - Relations among direct mapped, set associate, full associate
  - Block identification
    - Structure description $\leftarrow\rightarrow$ graph ?
    - Could you break the Physical Address down to Tag/ Index / block offset ?
  - Replacement
    - RAND, LRU, FIFO
  - Write strategy
    - Write through / write back
    - Write allocate / No write allocate

Memory Hierarchy (cont.)

What's the difference between cache and Virtual memory ?

- Cache ← L2 cache, memory
- Line(block)
- Direct mapped/set/full associate
- Pseudo RAND/ LRU / FIFO
- Write through / write back
- Hardware
- CPU Stall waiting for miss
- Aim at Memory Speed
- ……

- Physical Memory ←VM
- Page / segment
- Full associate
- LRU
- Write back
- TLB & software (page table)
- Interrupt while page fault
- Aim at Memory Size
- ……

What's TLB ? Can we call it a cache ? What are included in a TLB entry ?

Cache Performance

- What metric is used to evaluate cache performance ?
  - Miss Rate? AMAT ? CPU time ?
- How to calculate AMAT ? (examples)
- How to calculate CPUtime taking into account of memory system ?

What's the difference between Miss Rate and Misses per instruction ?

How to Improve Cache Performance?

- 3C model (C.3)
  - Compulsory
  - Capacity
  - Conflict
- 4th C –Coherency (4.3)
  - Truth sharing miss
  - False sharing miss

How to Improve Cache Performance? (C.3, 5.2)

1. Reduce the time to hit in the cache.--4
   ——small and simple caches, avoiding address translation, way prediction , and trace caches
2. Increase cache bandwidth .--3
   —— pipelined cache access, multibanked caches, non-blocking caches,
3. Reduce the miss penalty--4
   ——multilevel caches, critical word first, read miss prior to writes, merging write buffers, and victim caches
4. Reduce the miss rate--4
   ——larger block size, large cache size, higher associativity,and compiler optimizations
5. Reduce the miss penalty and miss rate via parallelism--2
   ——hardware prefetching, and compiler prefetching

Memory Technology

- What’s difference between DRAM & SRAM ?
- Memory organization
  - Wider memory
  - Simple interleaved memory
  - Independent memory banks
  - Avoiding Memory Bank Conflicts （complier / Hardware ）
- Memory chip
  - FPM (Fast Page Mode DRAM)
  - Synchronize DRAM
  - DDR (Double Date Rate)
  - RDRAM

Multiprocessor

- Catalogue the Parallel (MIMD) processors
  - Centralized shared memory / Distributed memory multi-processor
  - Pros vs. Cons
- concepts
  - SMP / DSM / CSM / Multiple computer
  - UMA / NUMA
  - SPP / MPP
- Communication Model
  - Share memory
  - Message passing

Cache coherence

- What's cache coherence ?
  - Write serialization
- Snooping protocol
  - Write Invalidate / Write update
  - What's the difference between them ?
  - State machine
- Directory-based protocol
  - Status machine (cache block, directory) (case 4.16)