Pipelined MIPS CPU

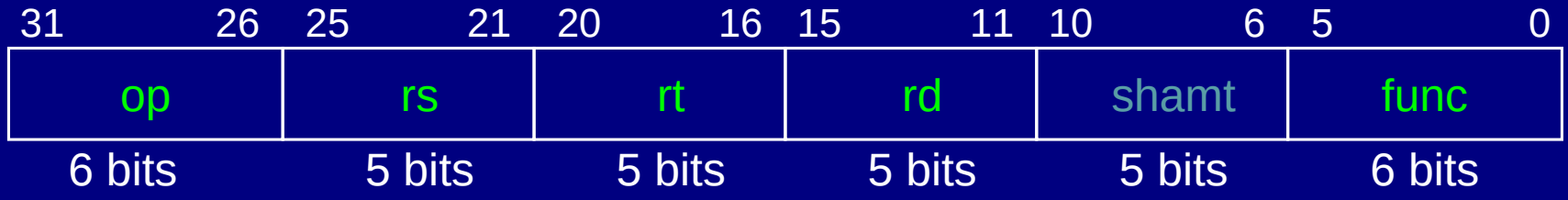
Yamin Li

Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University, Tokyo 184-8584 Japan

http://cis.k.hosei.ac.jp/~yamin/

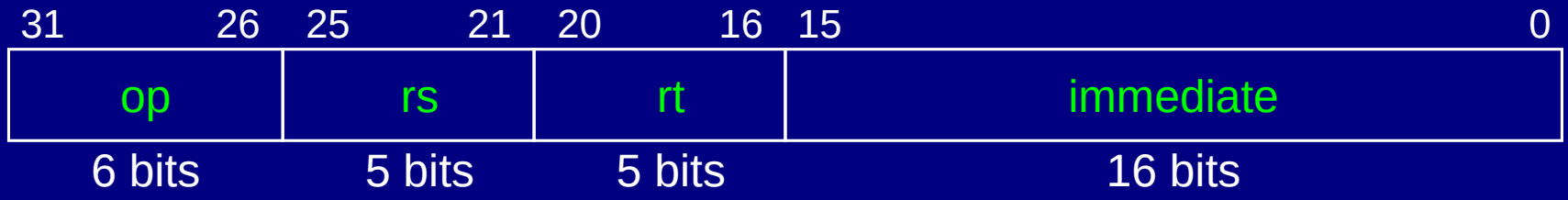
Instructions We Will Use

R-format instruction (add, sub, and, or, slt)

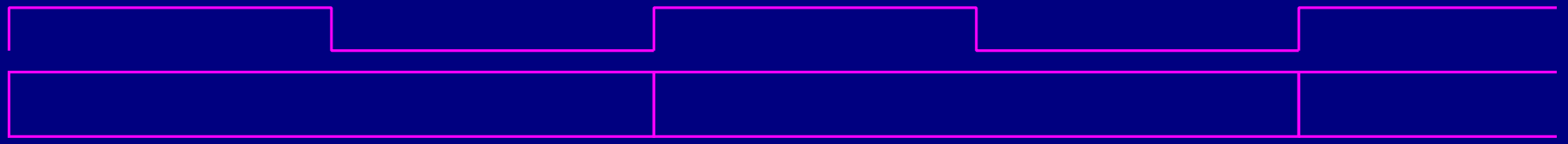


Example: add rd, rs, rt ; rd <= rs + rt

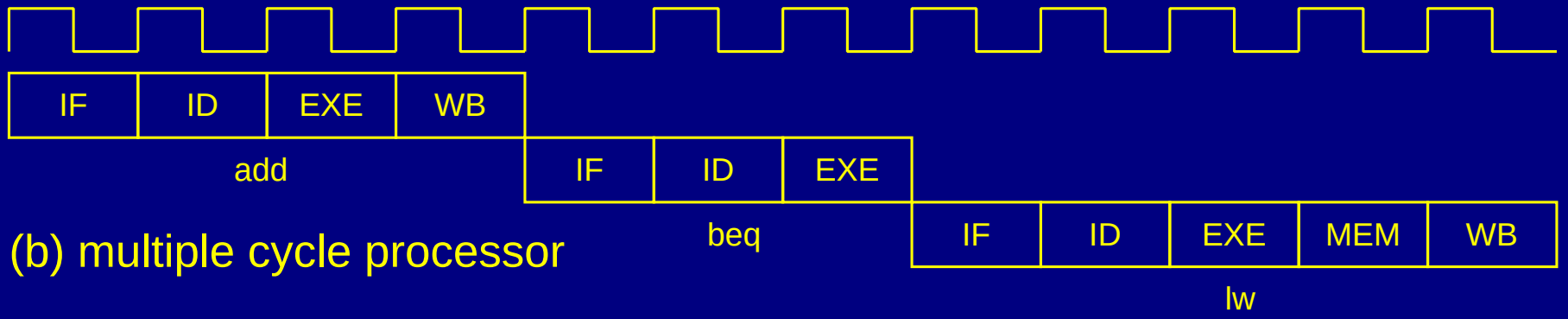
I-format instruction (lw, sw, beq, addi, andi, ori, bne)



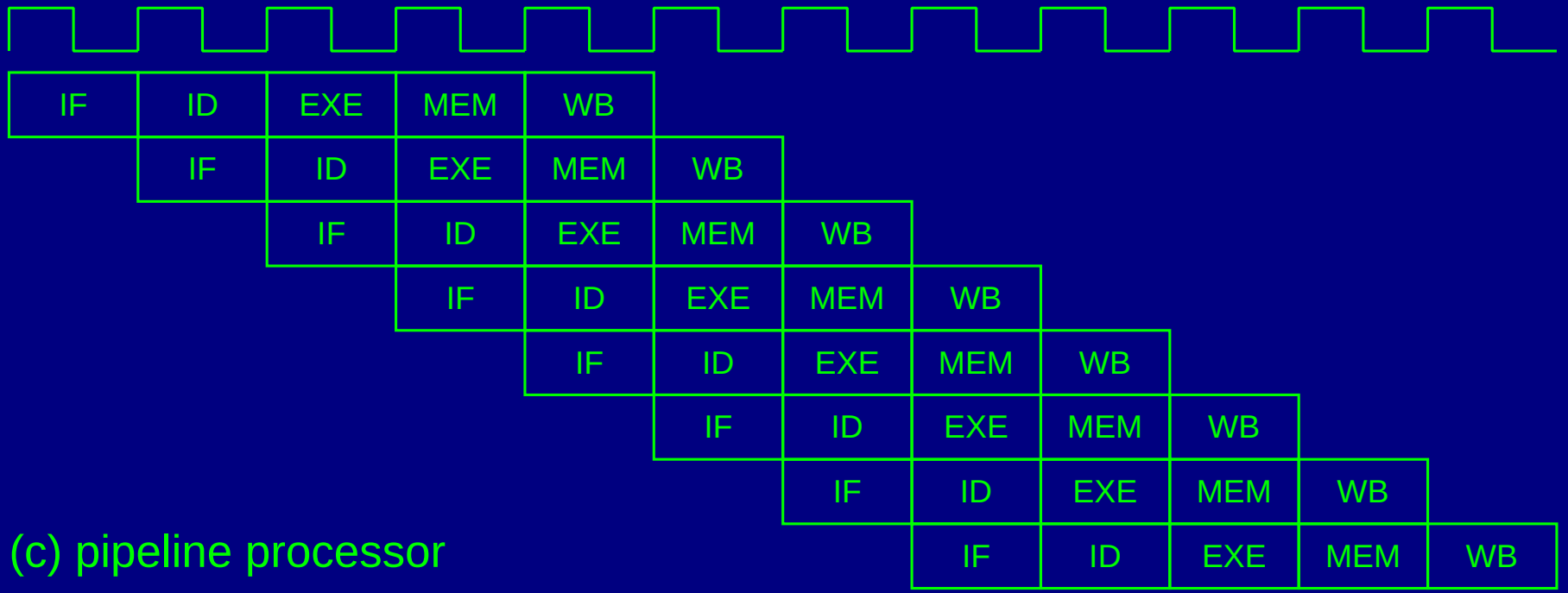

```
lw   rt,  imm(rs)      ; rt <= memory[rs+imm]
sw   rt,  imm(rs)      ; memory[rs + imm] <= rt
beq  rs,  rt, target   ; if (rs == rt) PC <= PC + 4 + signExtend(imm << 4)
```



(a) single cycle processor

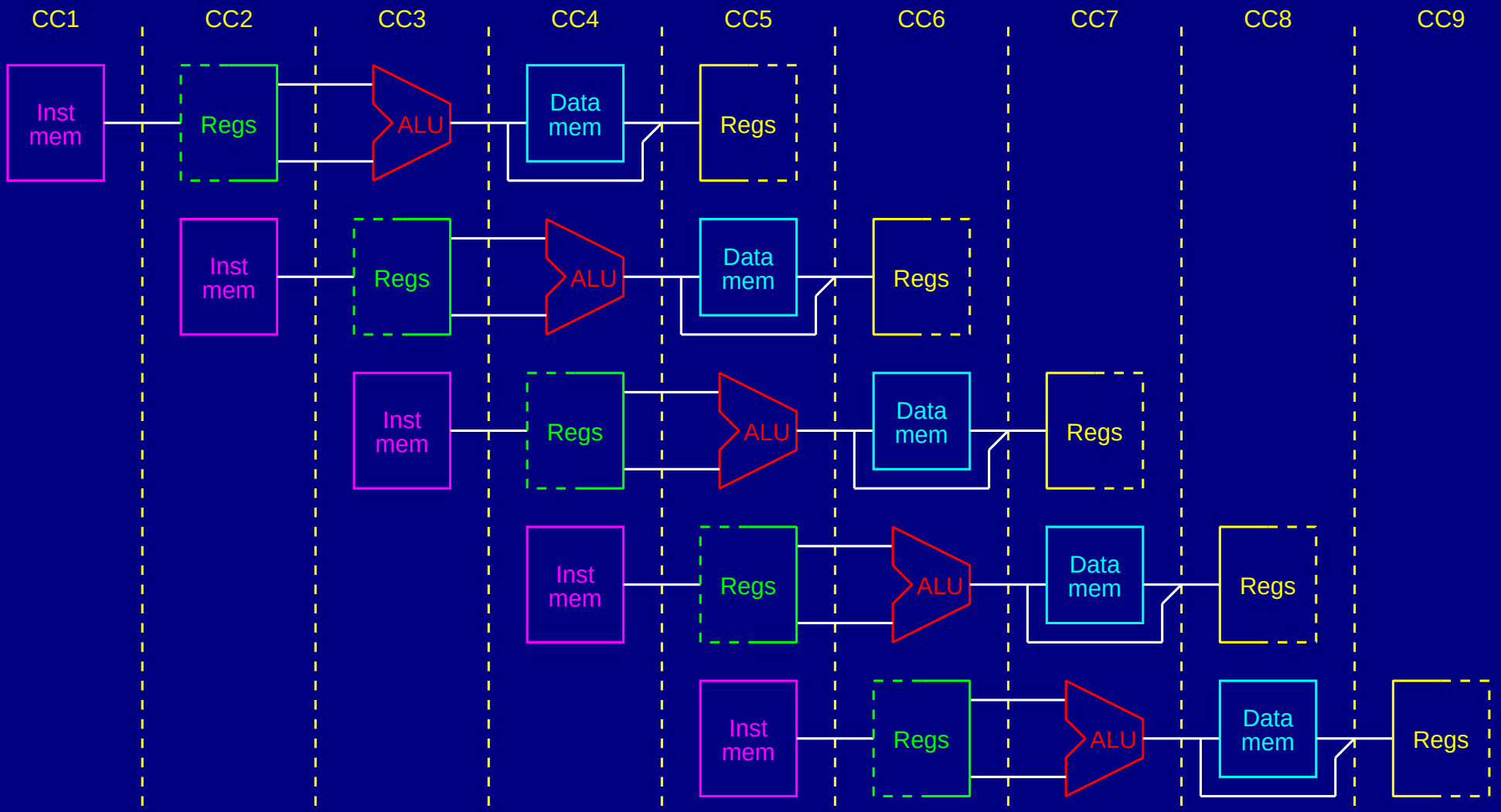


(b) multiple cycle processor

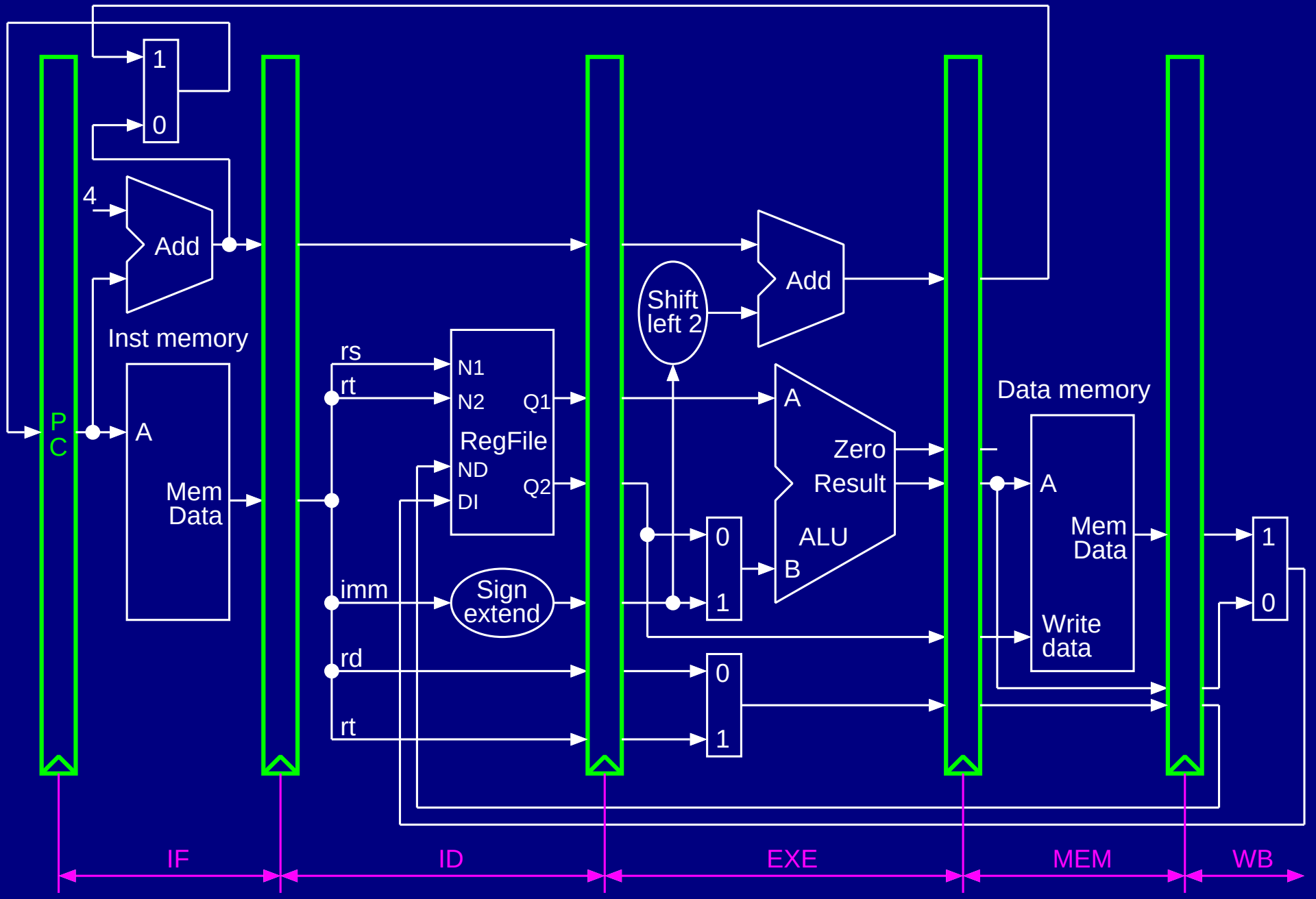


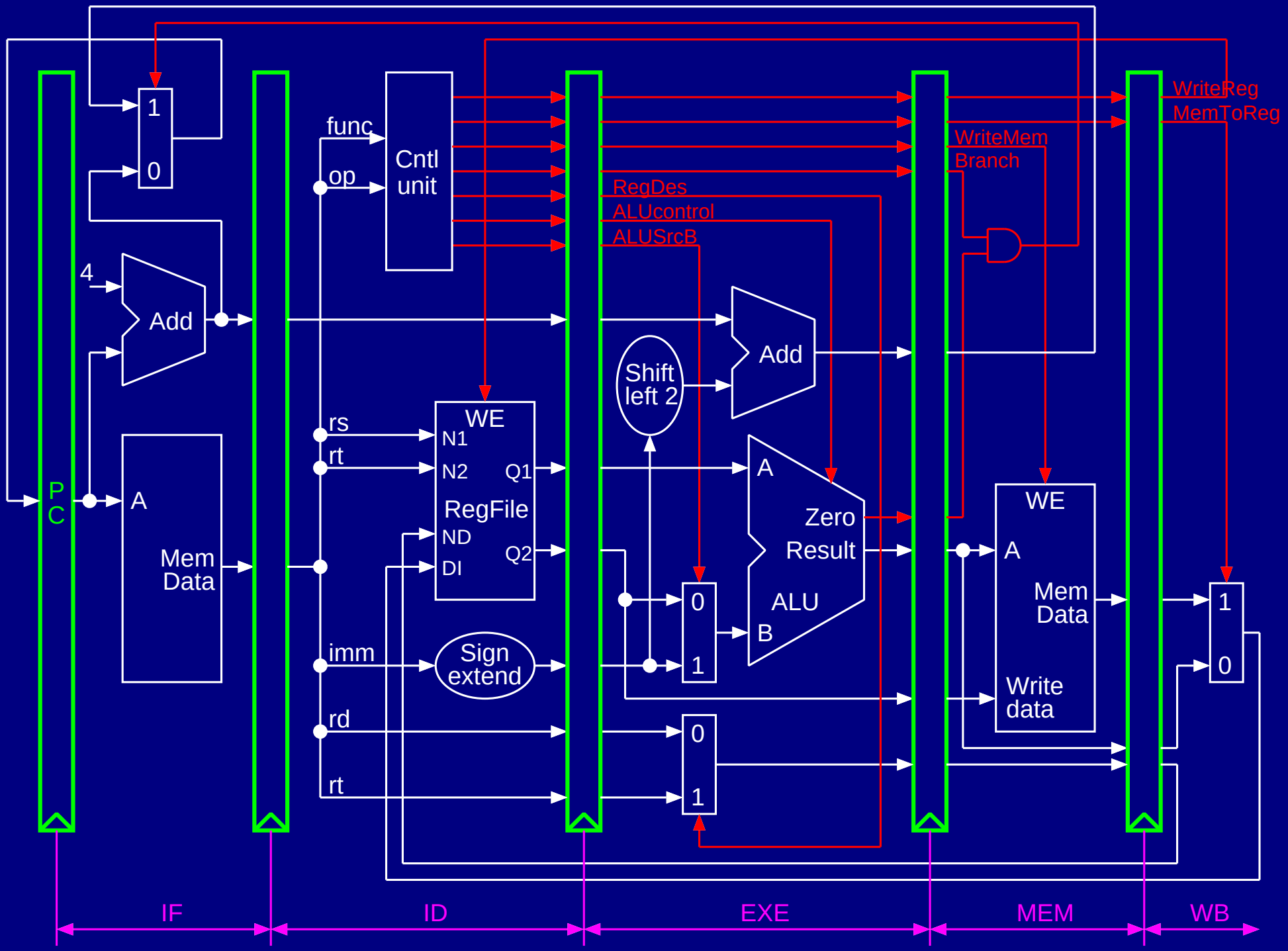
(c) pipeline processor

Concept of Pipelining

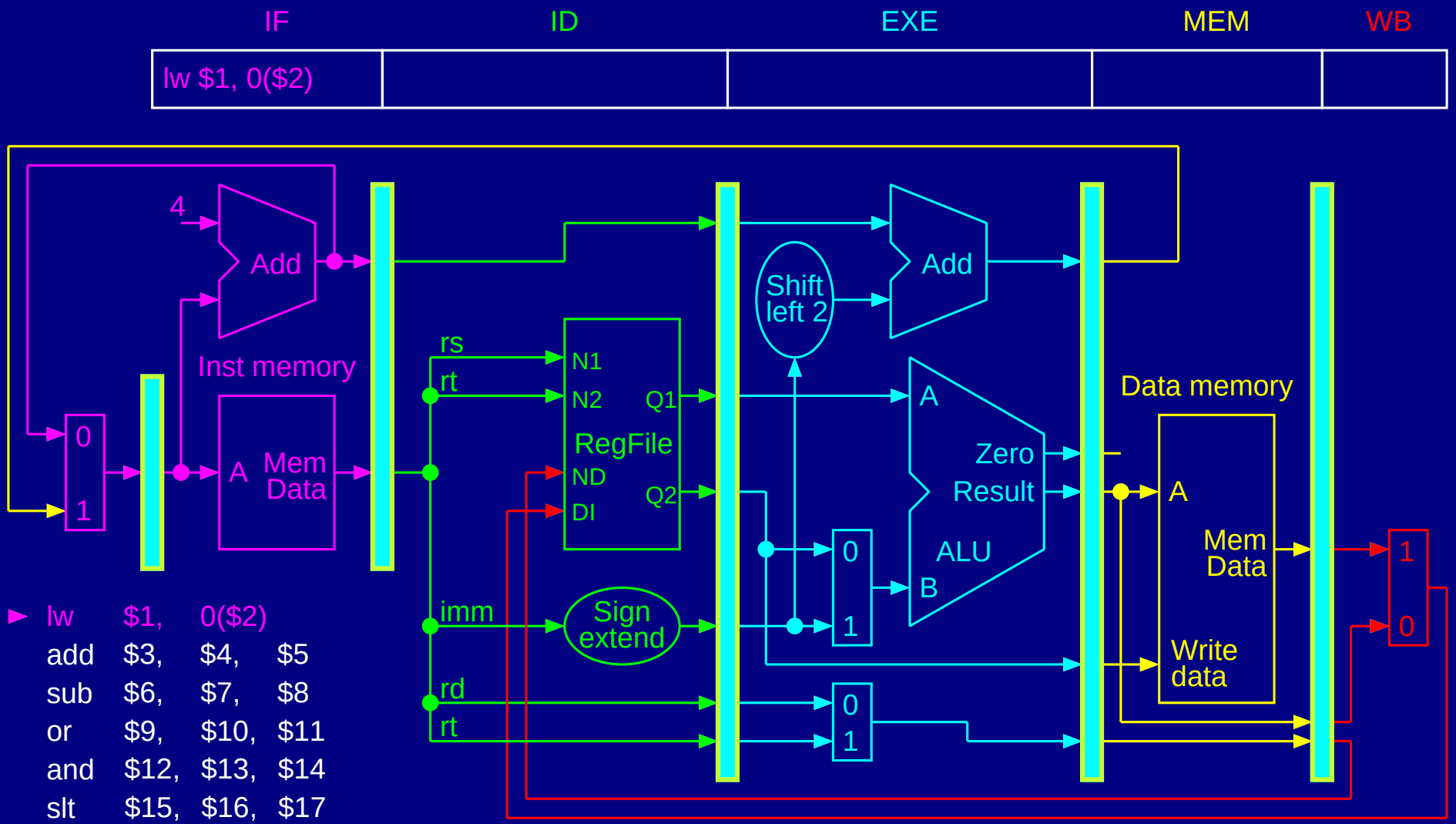


Pipelined MIPS CPU - Datapath

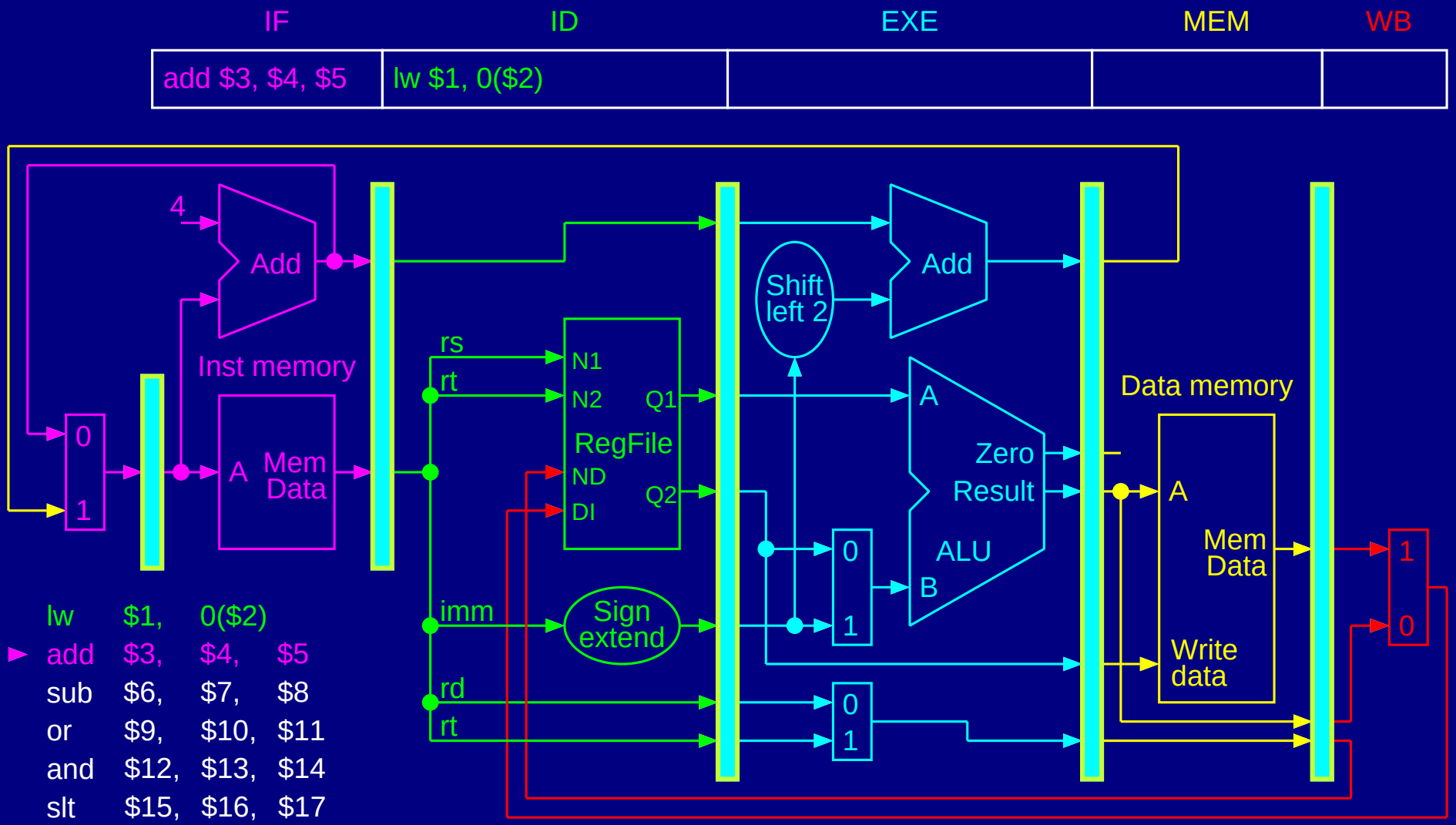



1
0
func
op
Cntl
unit
WriteReg
MemToReg
WriteMem
Branch
RegDes
ALUcontrol
ALUSrcB
4
Add
Shift
left 2
Add
WE
rs
rt
N1
N2
Q1
RegFile
ND
DI
Q2
A
Zero
Result
ALU
B
P
C
A
Mem
Data
WE
A
Mem
Data
Write
data
imm
Sign
extend
rd
rt
0
1
IF
ID
EXE
MEM
WB

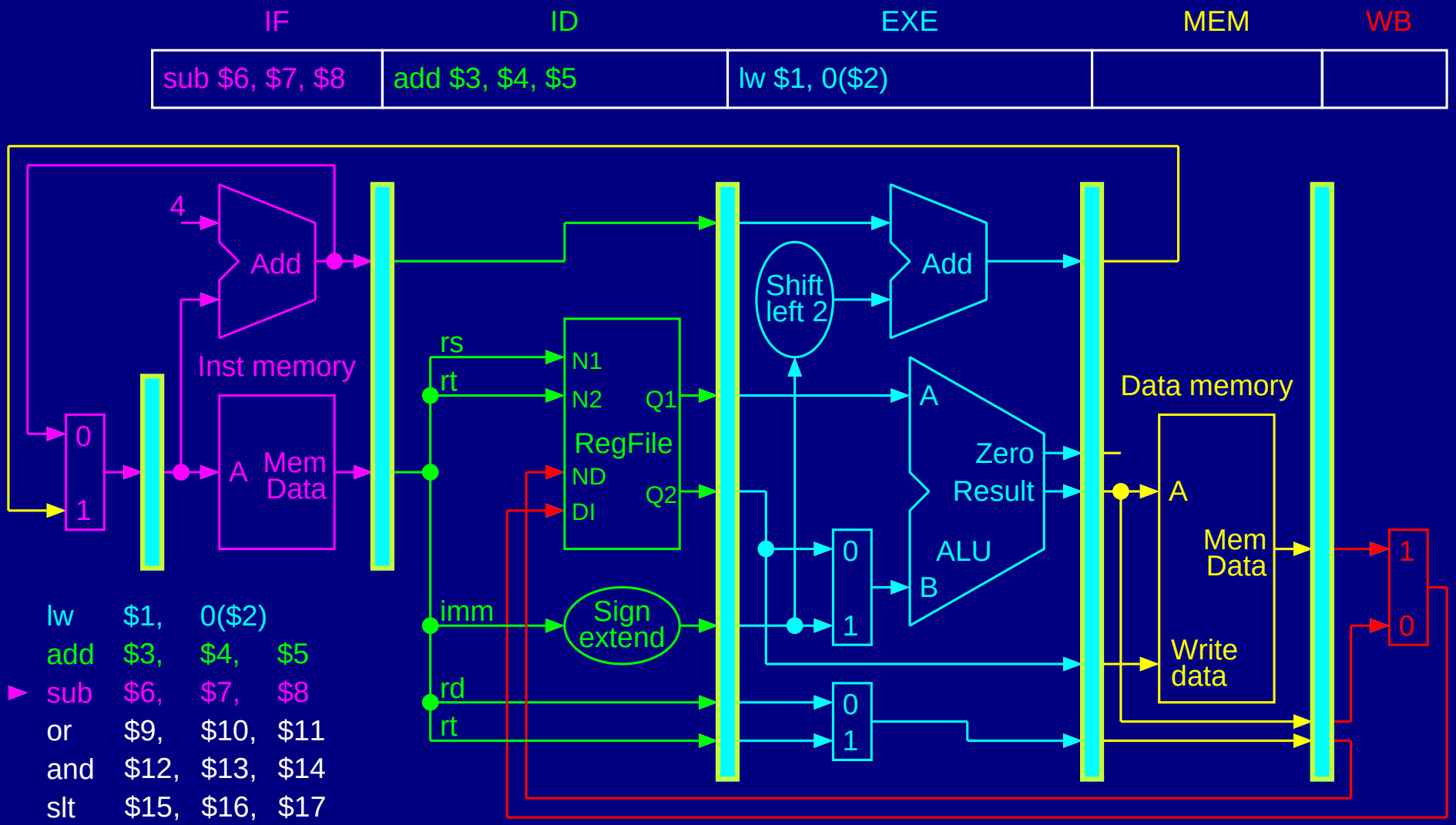
Pipelined MIPS CPU – Clock Cycle 1



Pipelined MIPS CPU – Clock Cycle 2

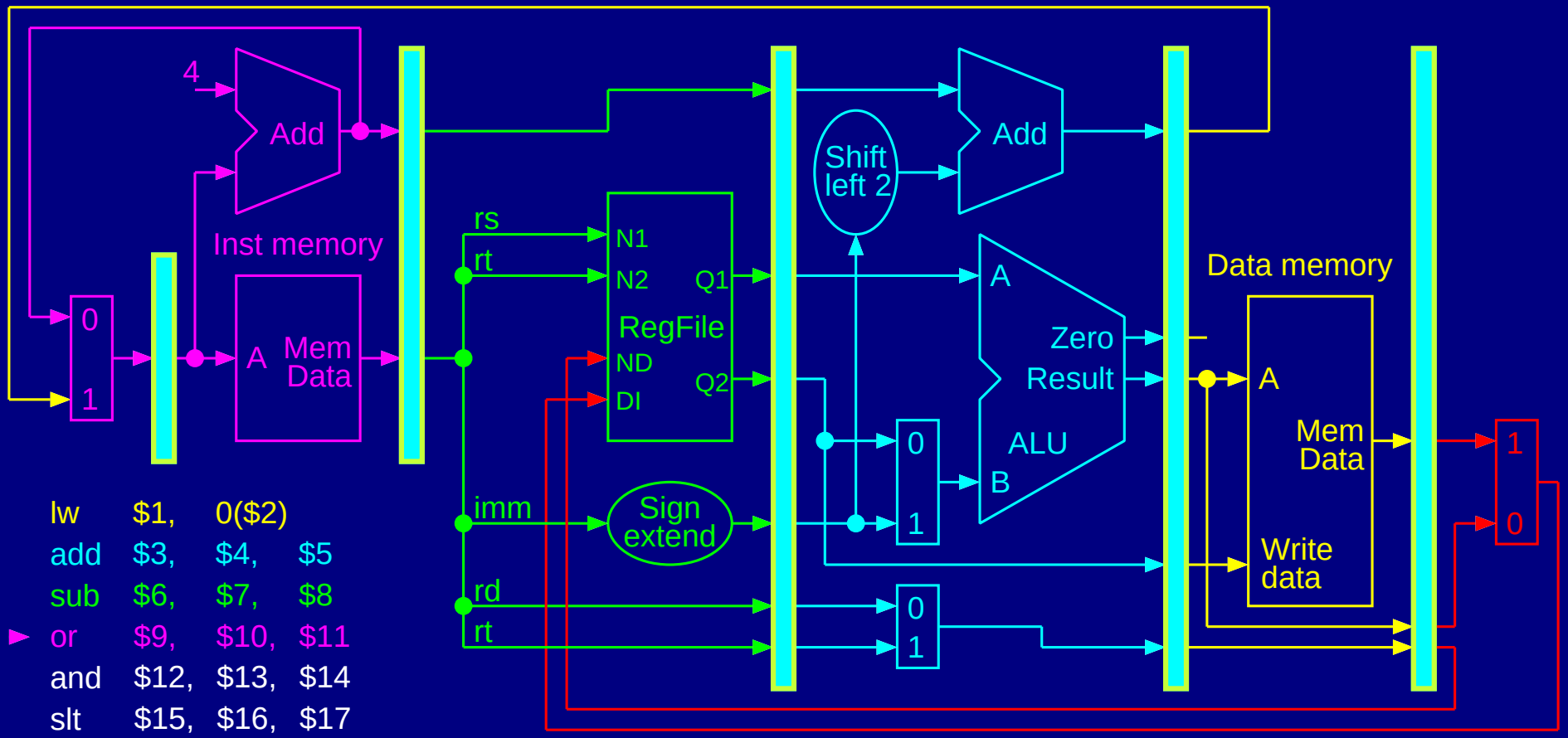


Pipelined MIPS CPU – Clock Cycle 3

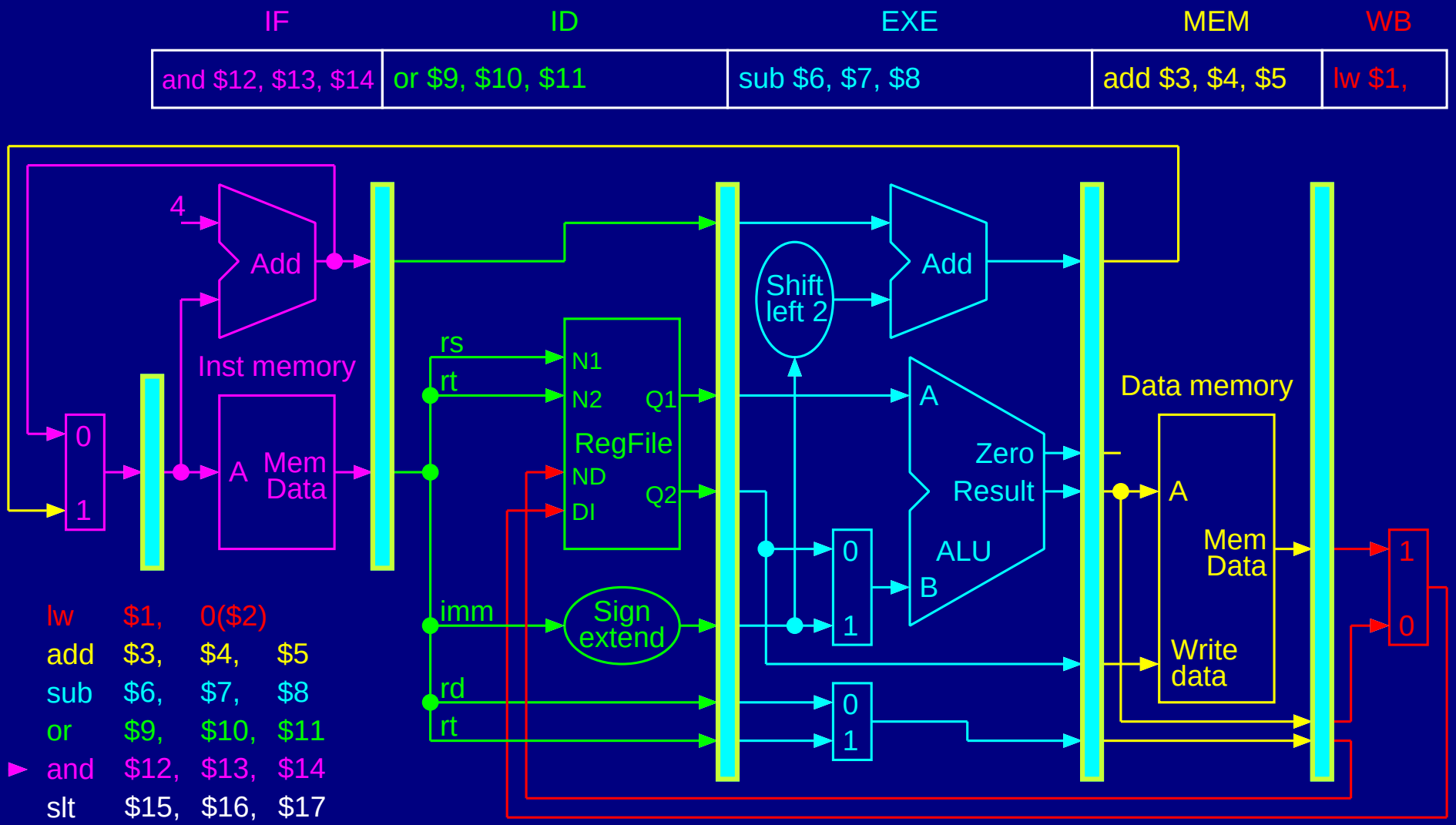


Pipelined MIPS CPU – Clock Cycle 4

IF	ID	EXE	MEM	WB
or $9, $10, $11	sub $6, $7, $8	add $3, $4, $5	lw $1, 0($2)	

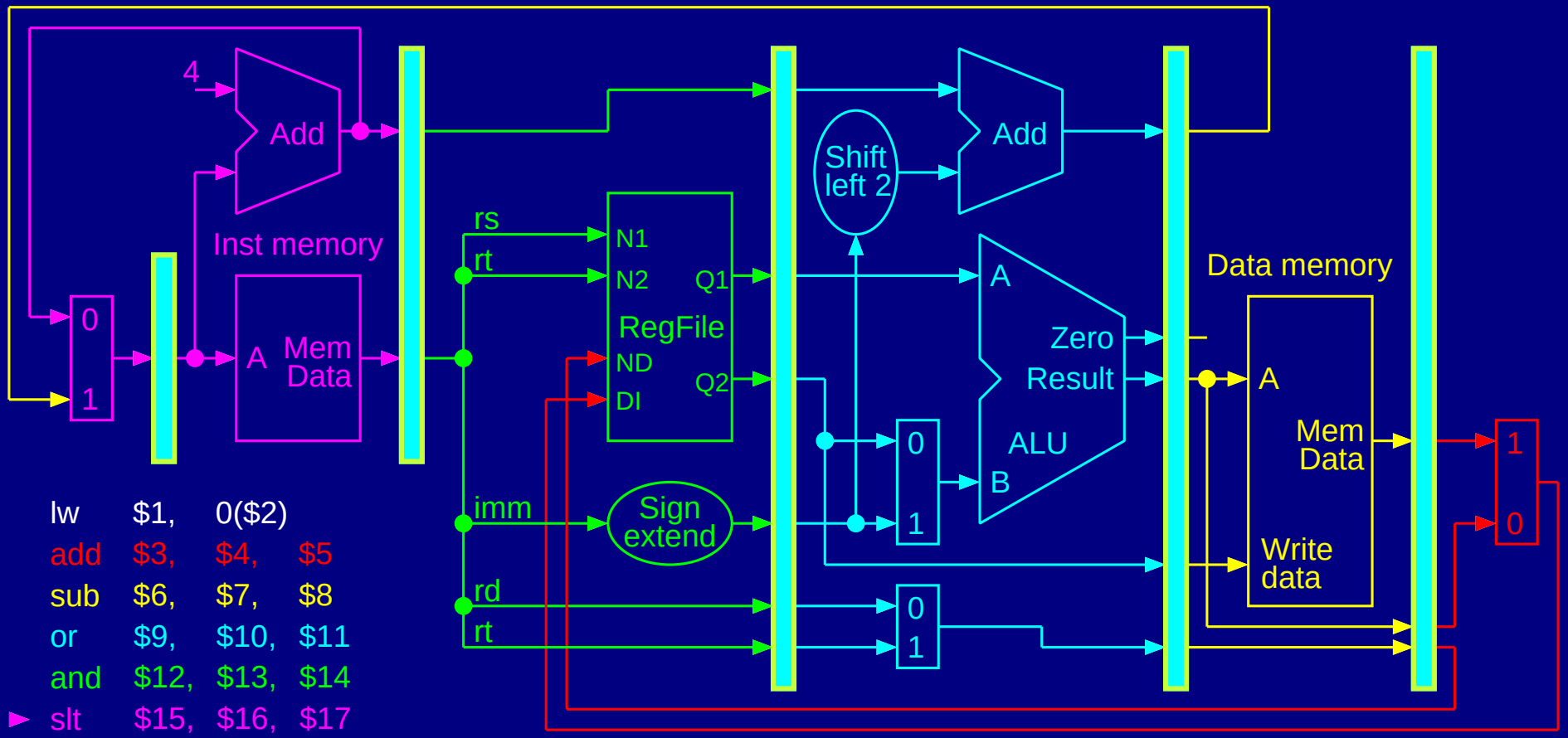


Pipelined MIPS CPU – Clock Cycle 5



Pipelined MIPS CPU – Clock Cycle 6

IF	ID	EXE	MEM	WB
slt $15,$16,$17	and $12,$13,$14	or $9, $10, $11	sub $6, $7, $8	add $3,



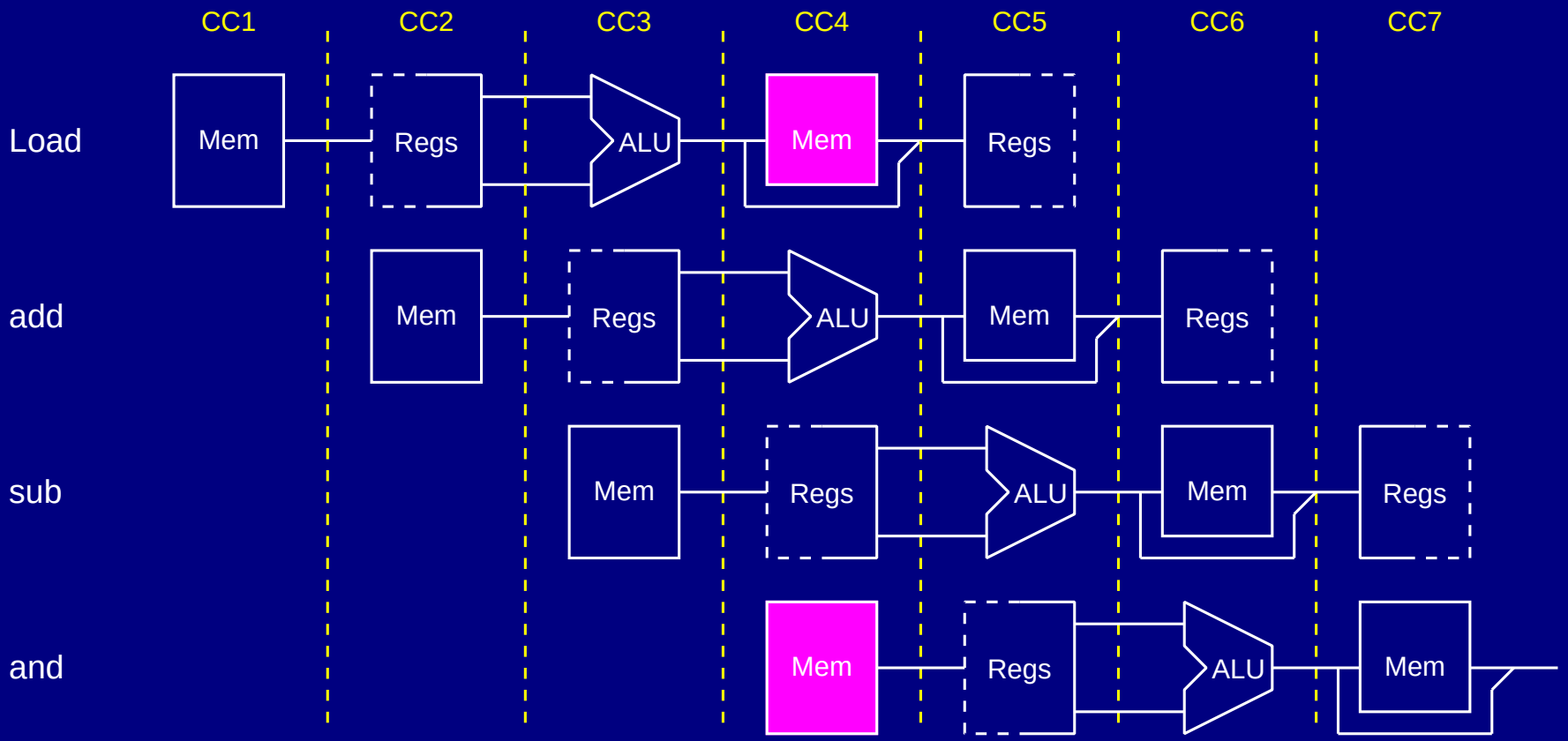
Three Classes of Pipeline Hazards

There are situations, called *hazards*, that prevent the next instruction in the instruction stream from executing during its designated clock cycle.

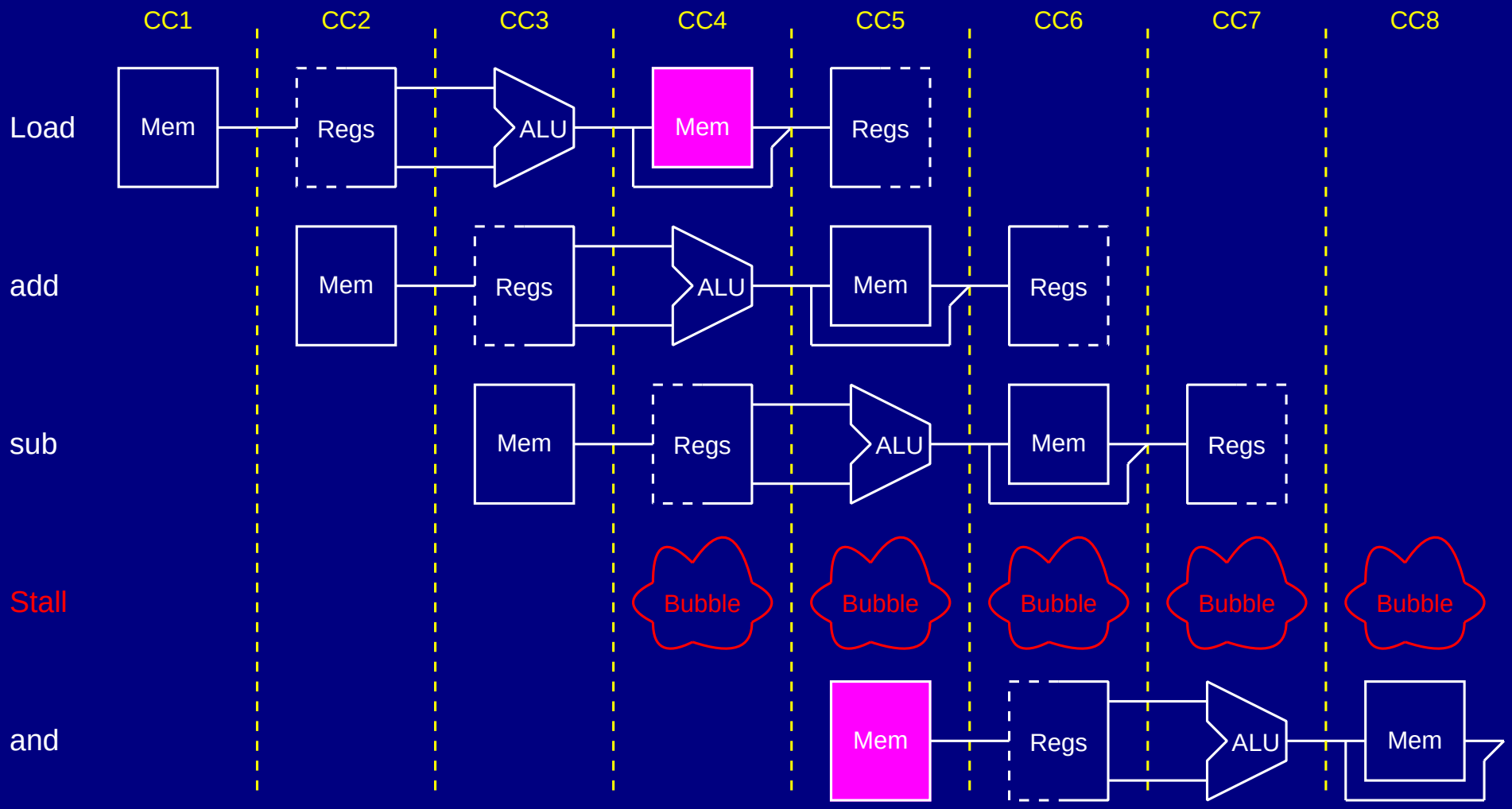
1. *Structural hazards* arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.
2. *Data hazards* arise when an instruction depends on the results of a previous instruction.
3. *Control hazards* arise from the pipelining of branches and other instructions that change the PC.

Structural Hazard — An Example

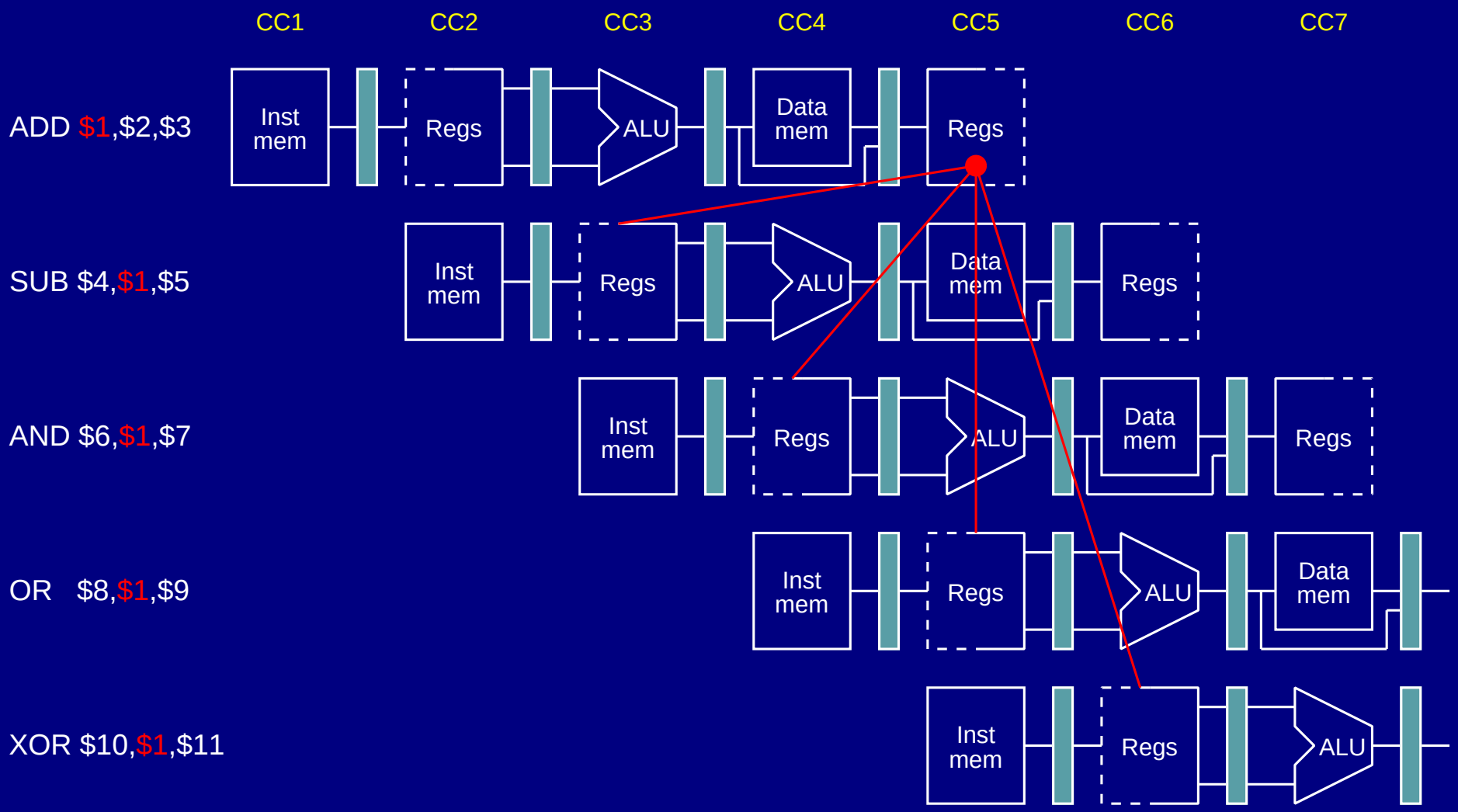
A machine with only one memory port will generate a conflict whenever a memory reference occurs.



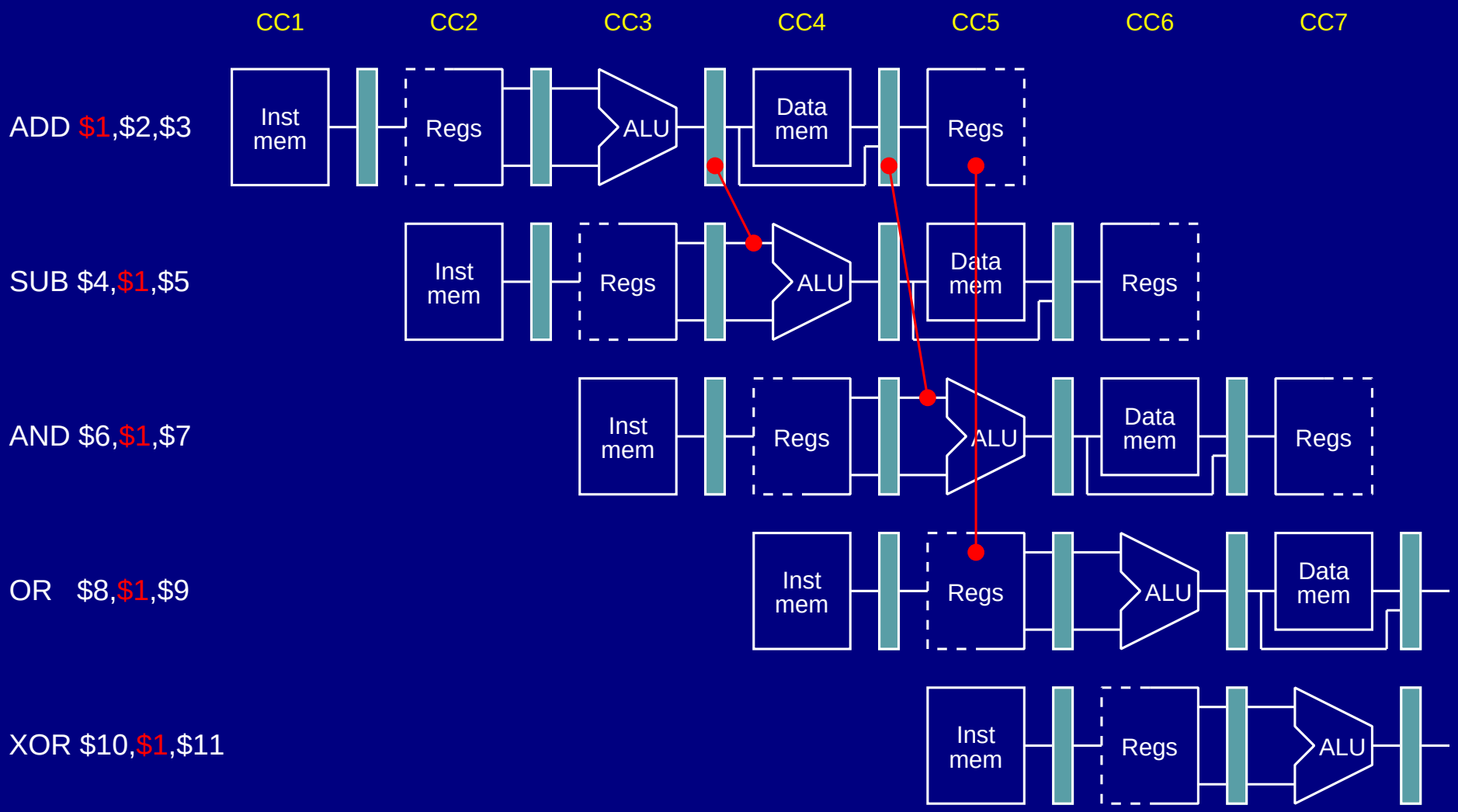
Structural Hazard Causes Bubbles to be Inserted



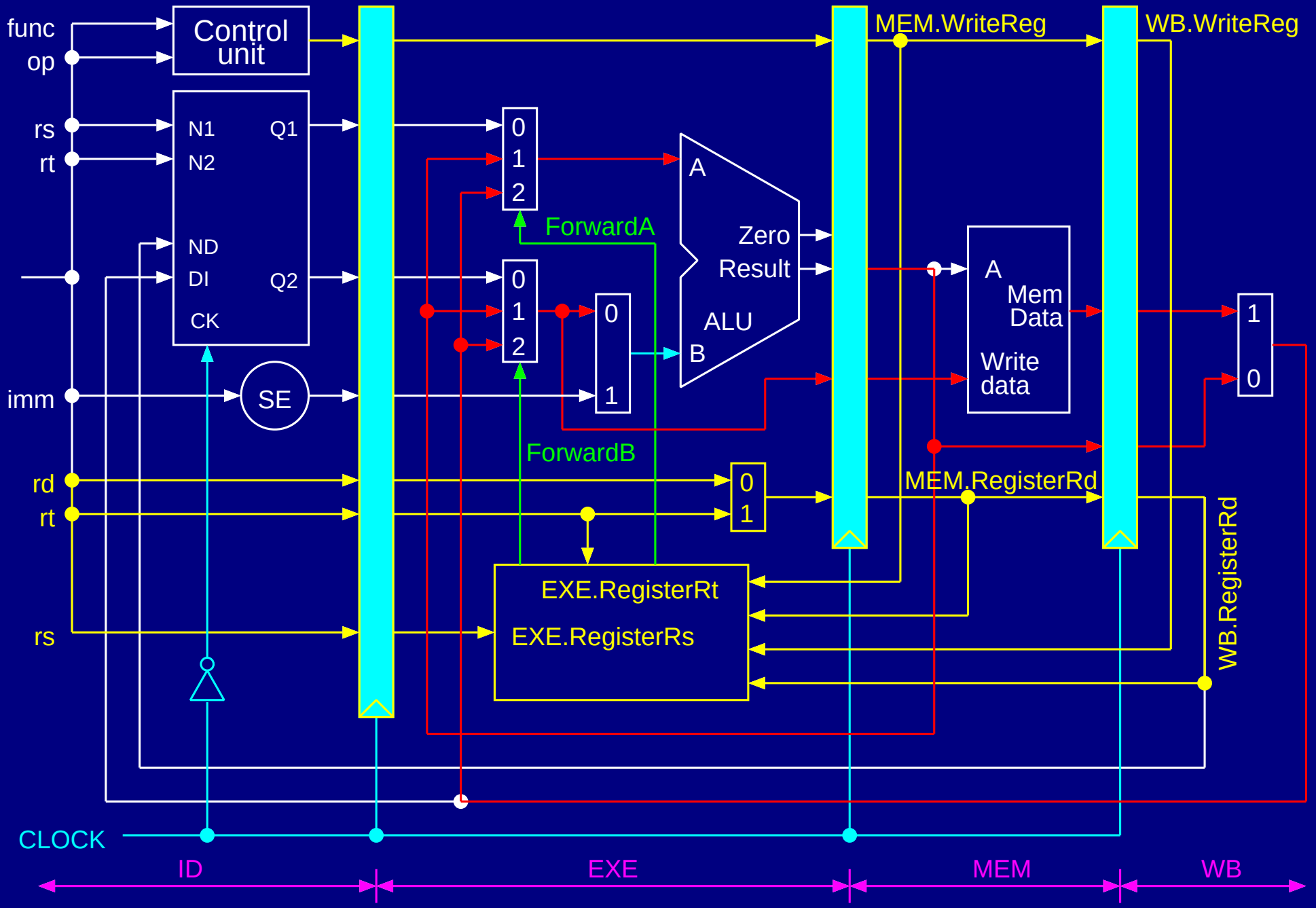
Data Hazard — An Example



Avoiding the Data Hazard with Forwarding Paths



Internal Forwarding



Internal Forwarding

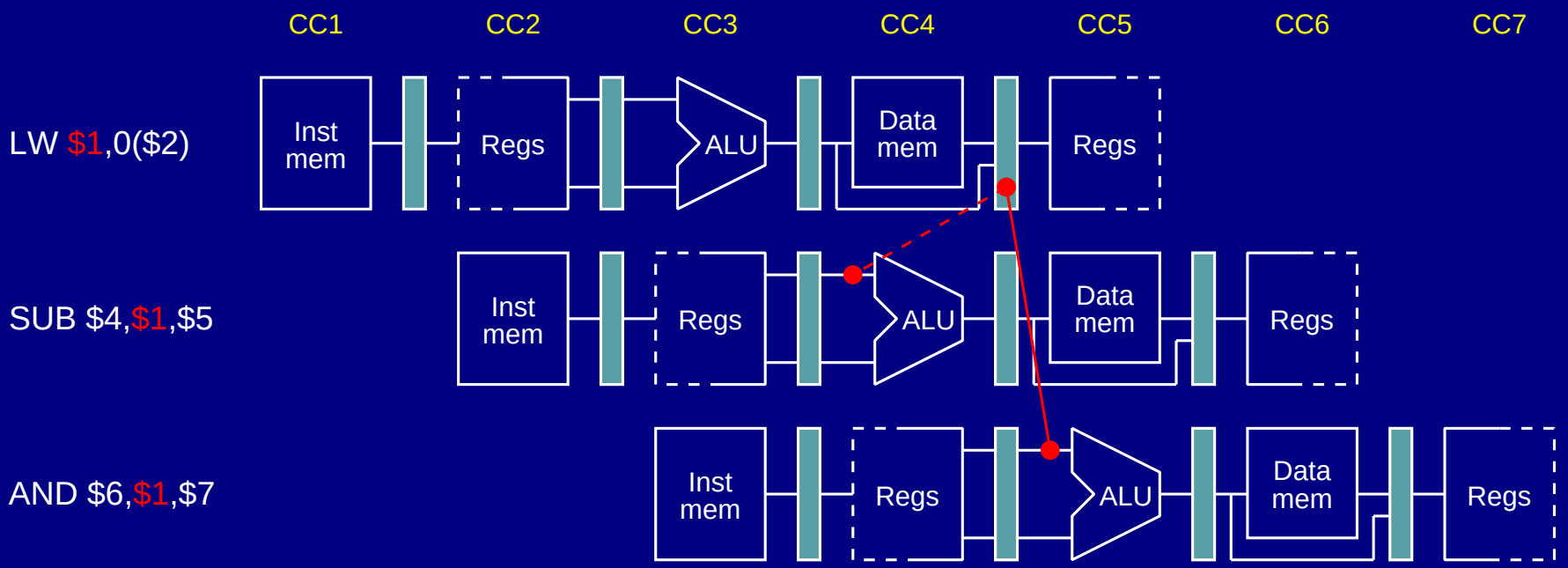
if (MEM.WriteReg
and (MEM.RegisterRd≠0)
and (MEM.RegisterRd==EXE.RegisterRs)) ForwardA=01

if (MEM.WriteReg
and (MEM.RegisterRd≠0)
and (MEM.RegisterRd==EXE.RegisterRt)) ForwardB=01

if (WB.WriteReg
and (WB.RegisterRd≠0)
and (MEM.RegisterRd≠EXE.RegisterRs))
and (WB.RegisterRd==EXE.RegisterRs)) ForwardA=10

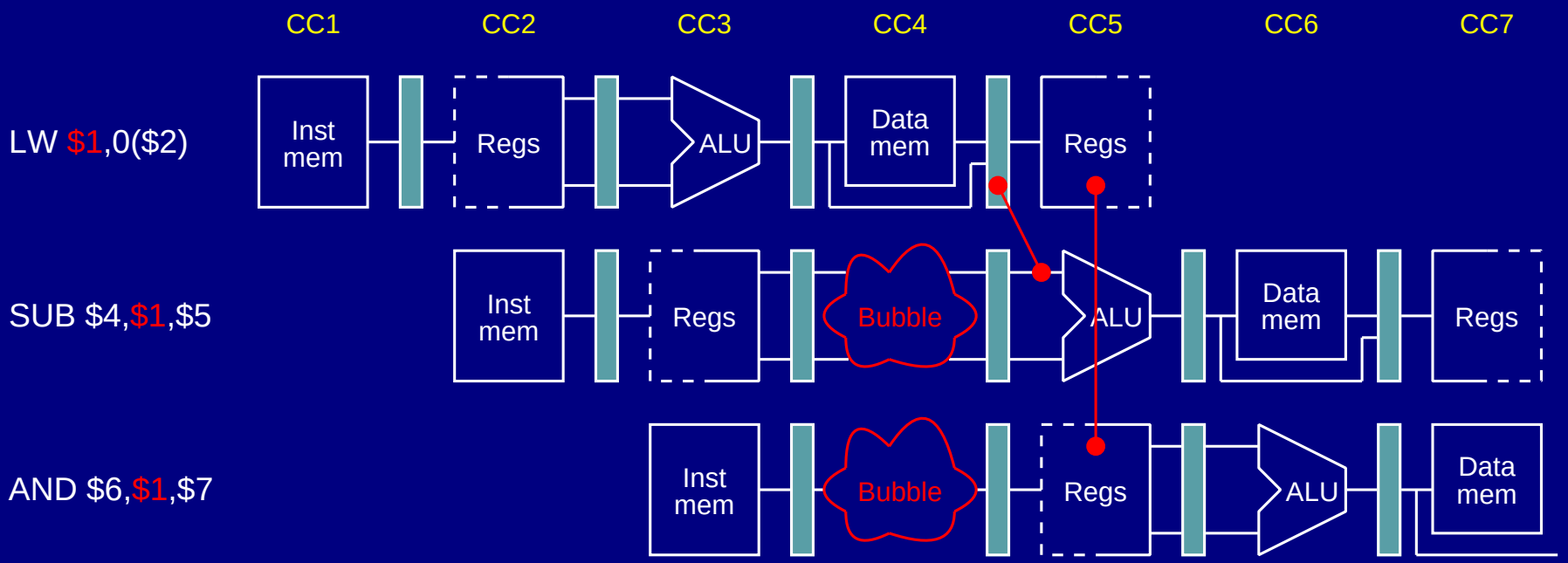
if (WB.WriteReg
and (WB.RegisterRd≠0)
and (MEM.RegisterRd≠EXE.RegisterRt))
and (WB.RegisterRd==EXE.RegisterRt)) ForwardB=10

Load Cannot Bypass Data to the Next Instruction



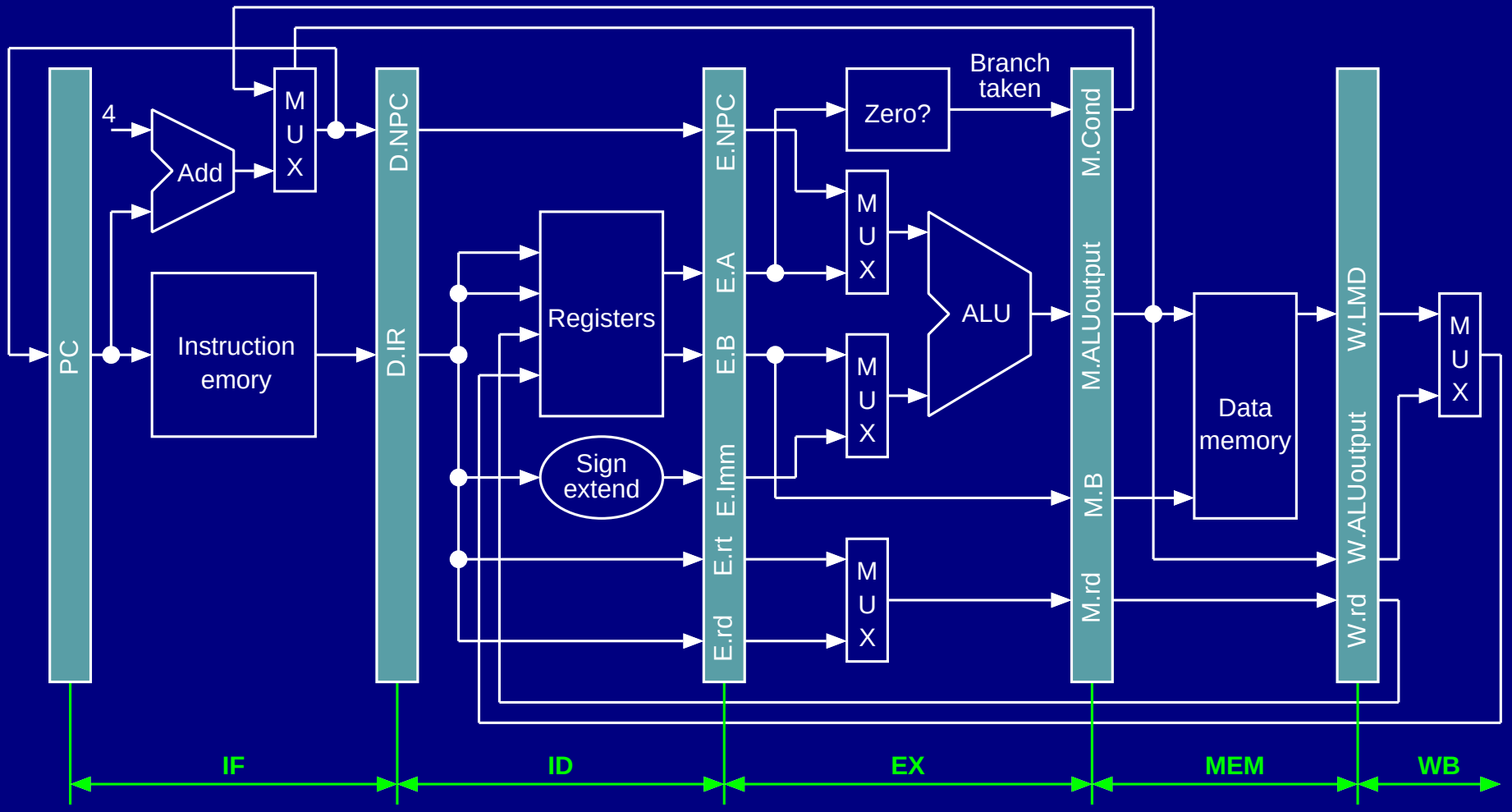
The load instruction can bypass its result to the AND instruction, but not to the SUB, since that would mean forwarding the result in “negative time”.

Load Hazard Requires Pipeline Stalls



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 Datapath — Three-Cycle Stall Branch

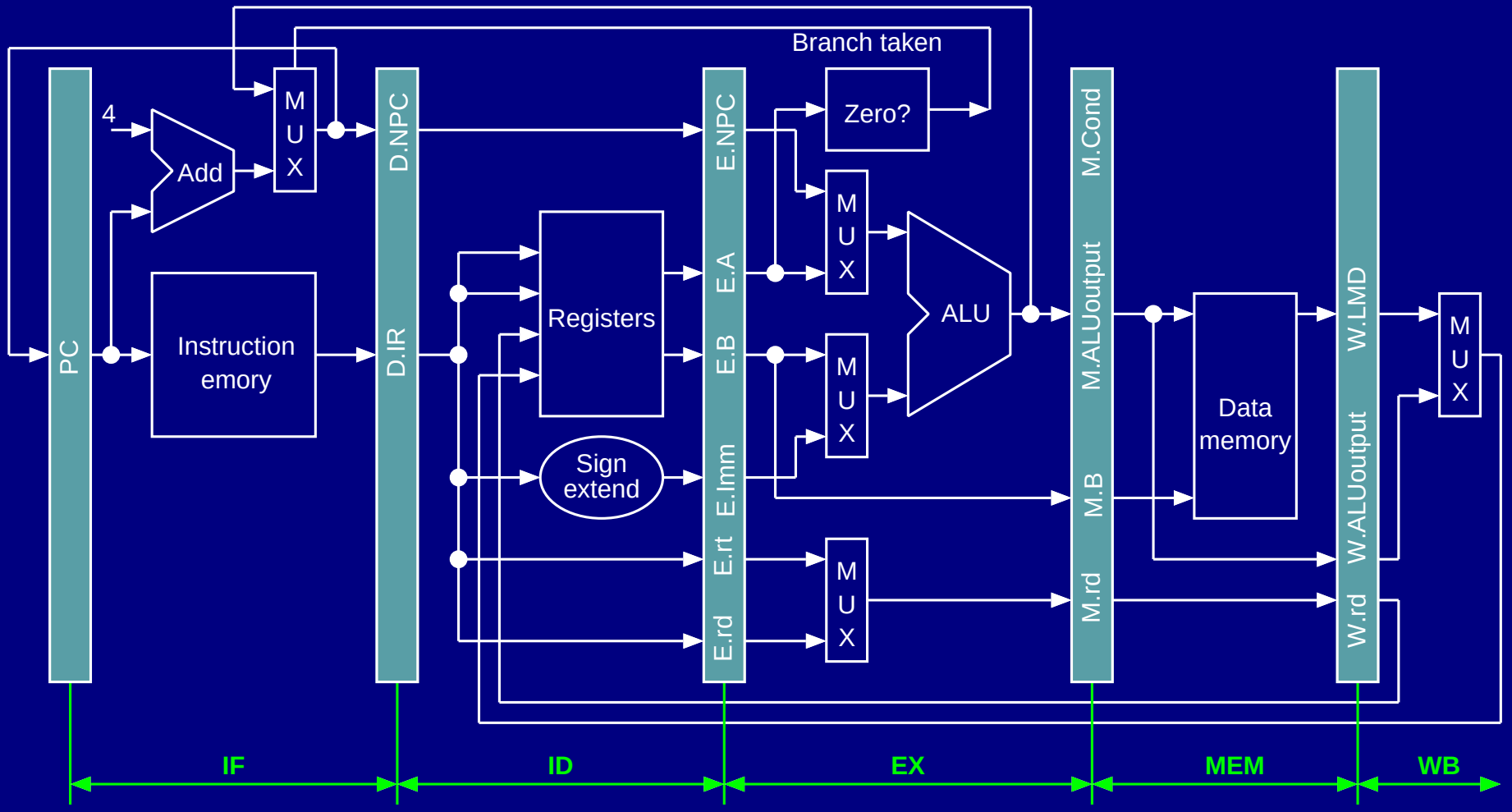


Branch Causes Pipeline Three-Cycle Stalls

Clock	1	2	3	4	5	6	7	8	9
Branch	IF	ID	EX	MEM	WB				
Successor 1	IF	*stall*	*stall*	IF	ID	EX	MEM	WB	
Successor 2					IF	ID	EX	MEM	
Successor 3						IF	ID	EX	
Successor 4							IF	ID	

A branch instruction (beqz in our figures) goes to the branch target address after finishing the MEM stage in the MIPS pipeline, so it causes a three-cycle stall: One cycle is a repeated IF and two cycles are idle.

Pipeline Datapath — Two-Cycle Stall Branch

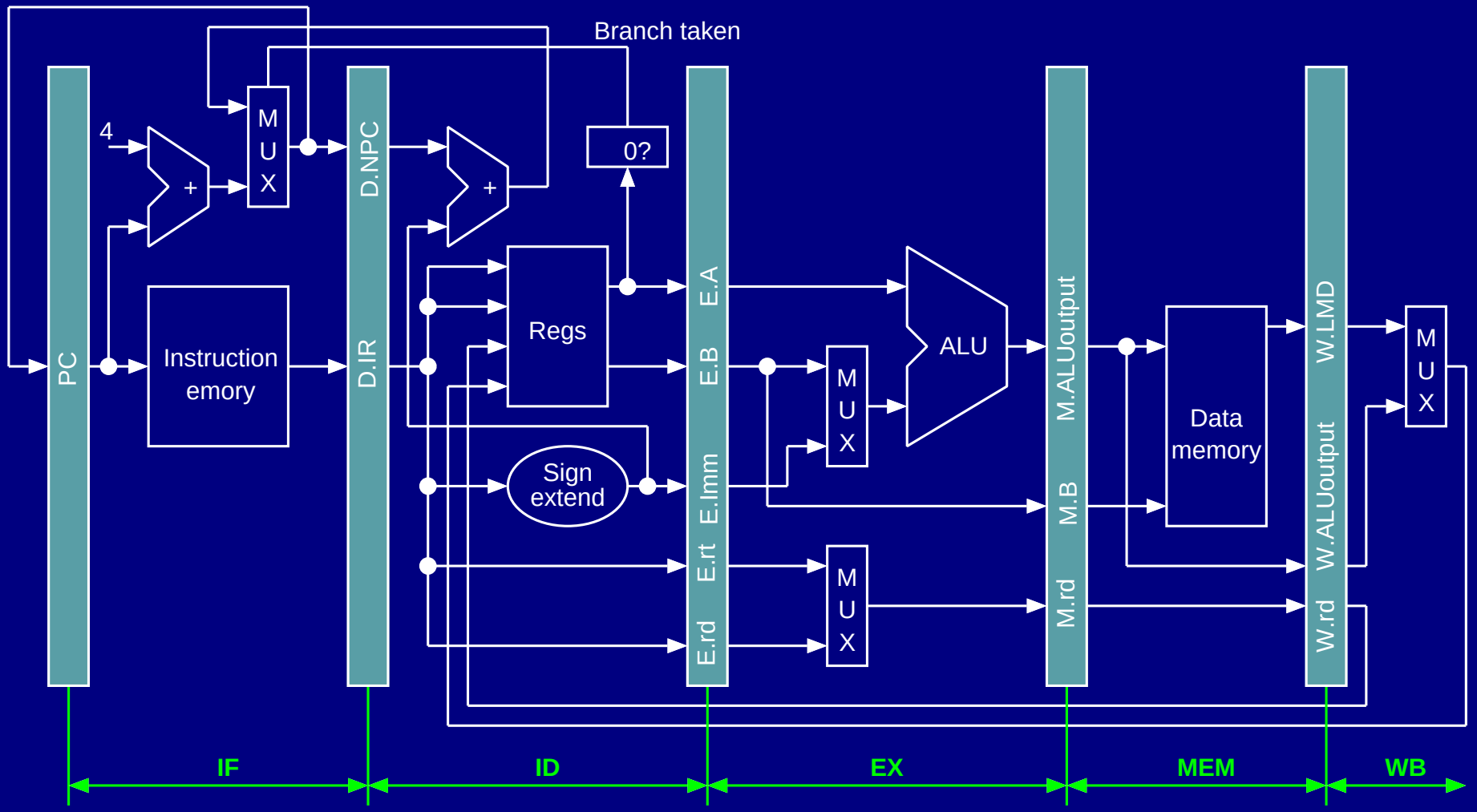


Branch Causes Pipeline Two-Cycle Stalls

Clock	1	2	3	4	5	6	7	8	9
Branch	IF	ID	EX	MEM	WB				
Successor 1	IF	*stall*	IF	ID	EX	MEM	WB		
Successor 2				IF	ID	EX	MEM	WB	
Successor 3					IF	ID	EX	MEM	
Successor 4						IF	ID	EX	

A branch instruction goes to the branch target address after finishing the EXE stage in the modified MIPS pipeline, so it causes a two-cycle stall: One cycle is a repeated IF and one cycle is idle.

Pipeline Datapath — One-Cycle Stall Branch



Branch Causes Pipeline One-Cycle Stalls

Clock	1	2	3	4	5	6	7	8	9
Branch	IF	ID	EX	MEM	WB				
Successor 1	IF	IF	ID	EX	MEM	WB			
Successor 2			IF	ID	EX	MEM	WB		
Successor 3				IF	ID	EX	MEM	WB	
Successor 4					IF	ID	EX	MEM	

A branch instruction goes to the branch target address after finishing the ID stage in the modified MIPS pipeline, so it causes a one-cycle stall: One cycle is a repeated IF.

Delayed Branch

Branch is not taken

Branch	IF	ID	EX	MEM	WB	
Inst i+1		IF	ID	EX	MEM	WB *(delay slot)*
Inst i+2			IF	ID	EX	MEM WB
Inst i+3				IF	ID	EX MEM WB

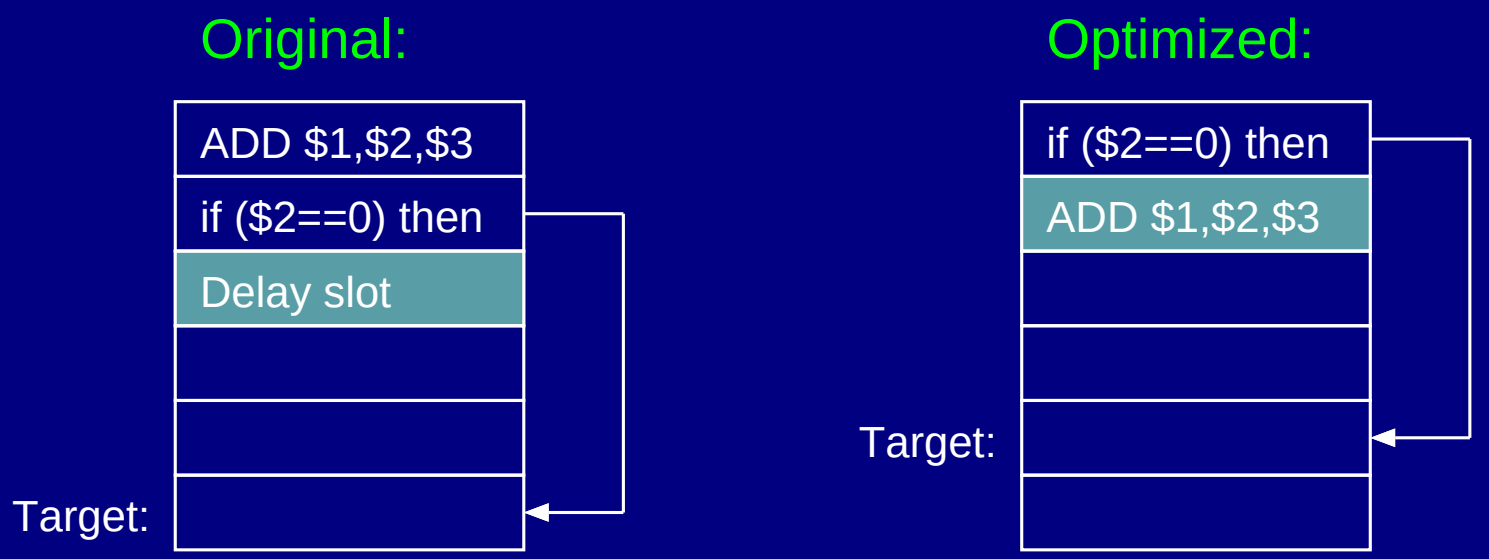
Branch is taken

Branch	IF	ID	EX	MEM	WB	
Inst i+1		IF	ID	EX	MEM	WB *(delay slot)*
Target			IF	ID	EX	MEM WB
Target+1				IF	ID	EX MEM WB

Delayed Branch — Optimization Method 1

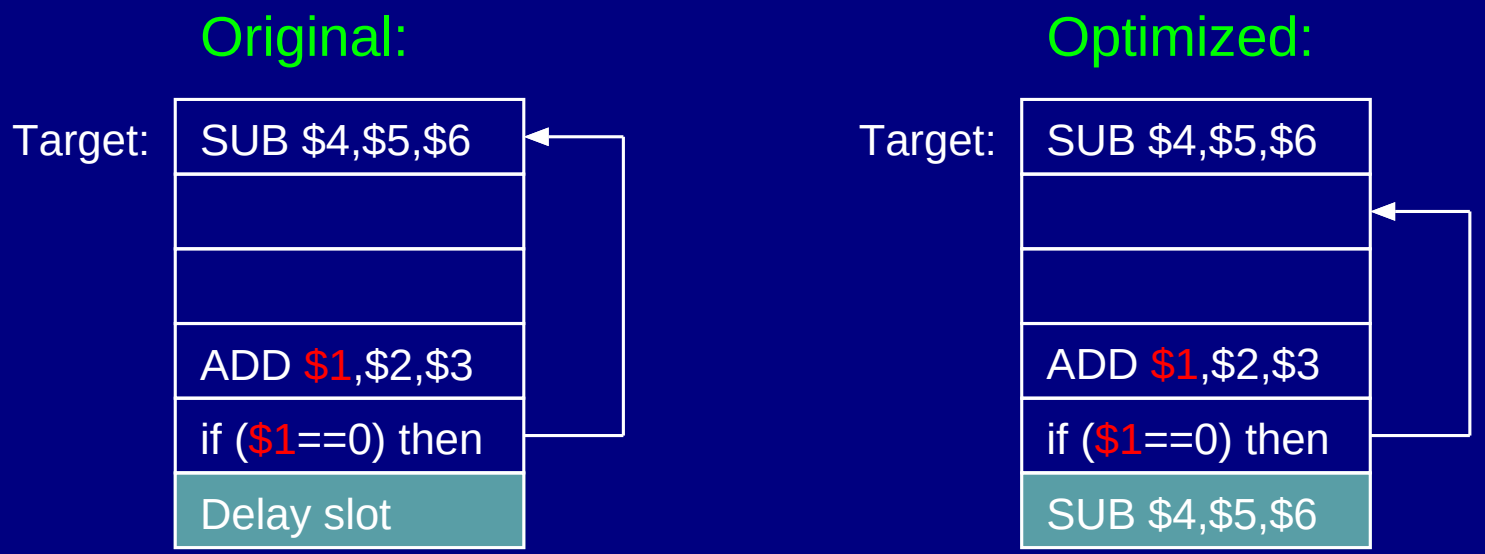
The job of the compiler is to make the instruction in the delay slot valid and useful.

1. Find a useful instruction from *before* — best choice:



Delayed Branch — Optimization Method 2

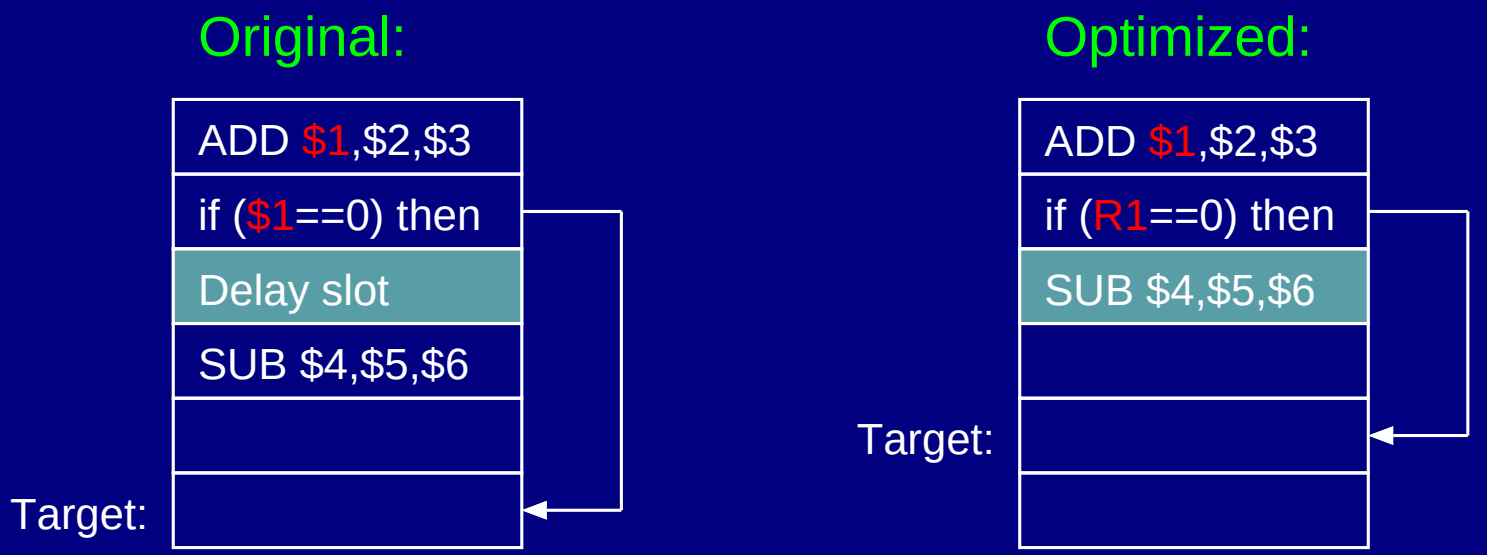
2. Find a useful instruction from *target*:



Used in loop branch. The delay slot is scheduled from the target of the branch. The content of $4 should not be used in the branch-untaken path.

Delayed Branch — Optimization Method 3

3. Find a useful instruction from *fall through*:

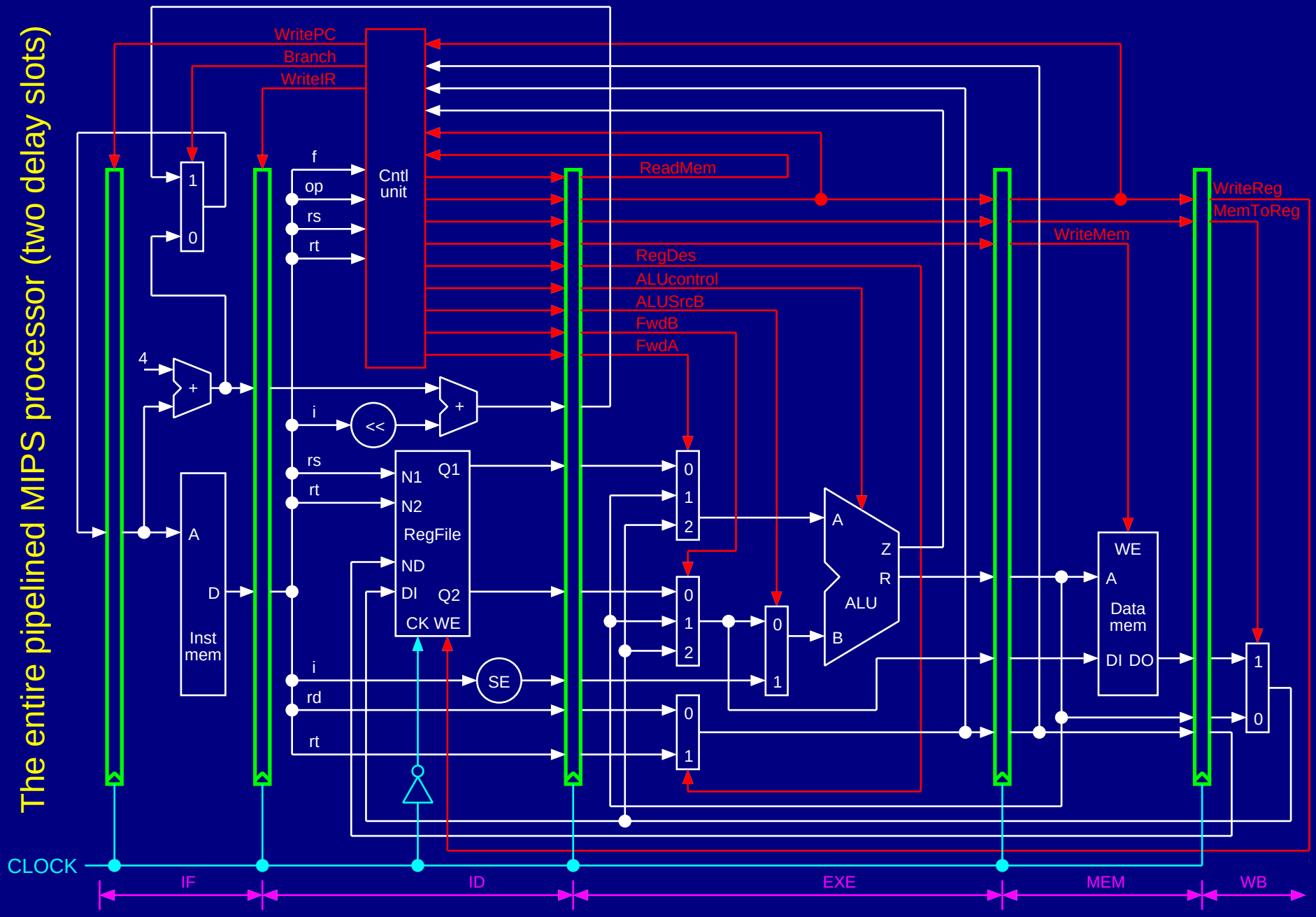


Useful when branch is not taken. The delay slot is scheduled from the branch-untaken path. The content of $4 should not be used in the branch-taken path.

Pipelined MIPS Processor

Pipelined MIPS processor with two delay slots

Data hazard checking was moved to ID stage.


The entire pipelined MIPS processor (two delay slots)
WritePC
Branch
WriteIR
Cntl unit
f
op
rs
rt
ReadMem
WriteReg
MemToReg
WriteMem
RegDes
ALUcontrol
ALUSrcB
FwdB
FwdA
4
+
<<
i
rs
rt
N1
Q1
N2
RegFile
ND
DI
Q2
CK WE
A
D
Inst mem
SE
rd
A
Z
R
ALU
B
WE
Data mem
DI DO
CLOCK
IF
ID
EXE
MEM
WB

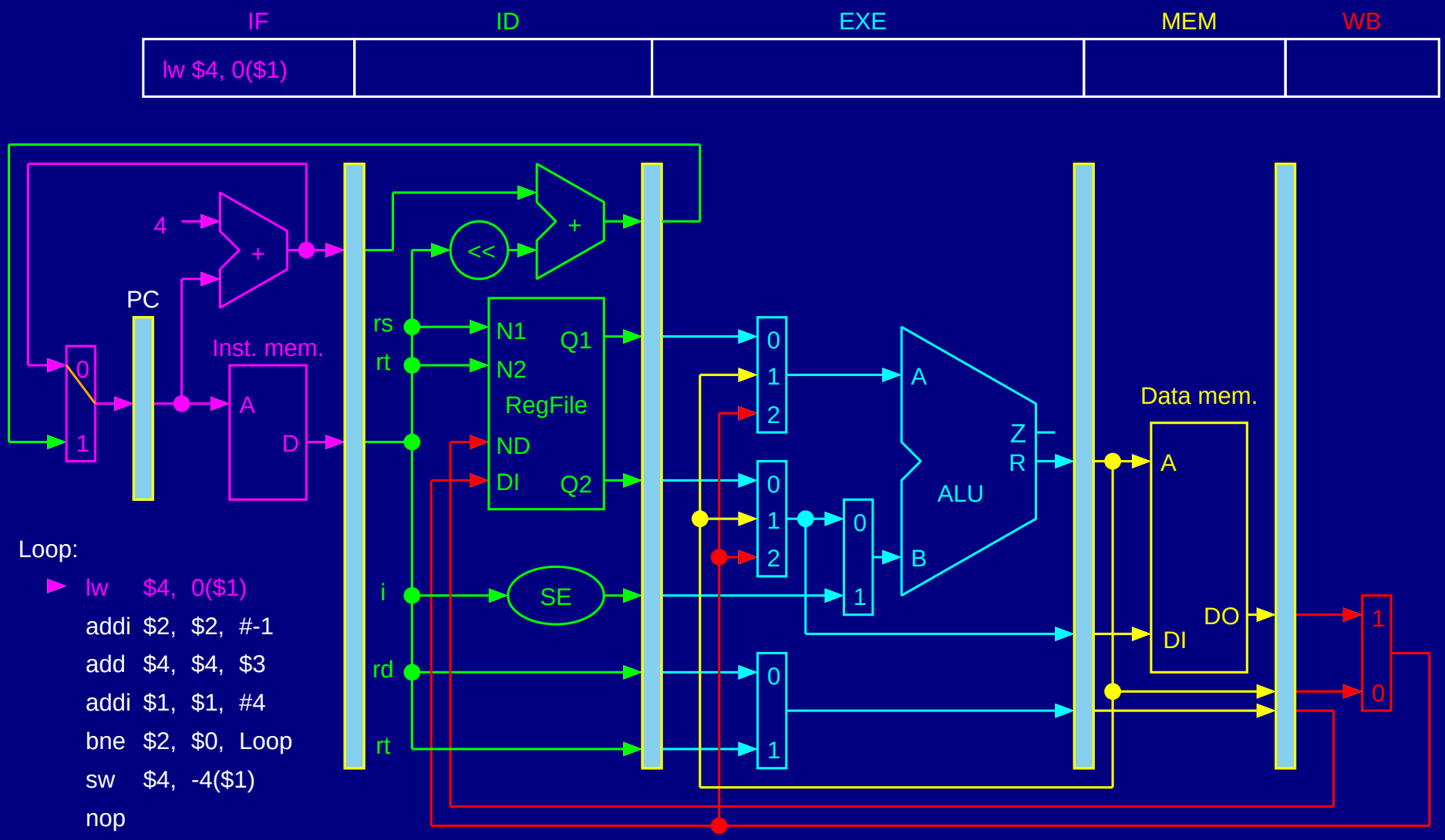
Program Execution on Pipelined MIPS CPU

```
for (i = 0; i < n; i++) {
    x[i] = x[i] + s;
}
```

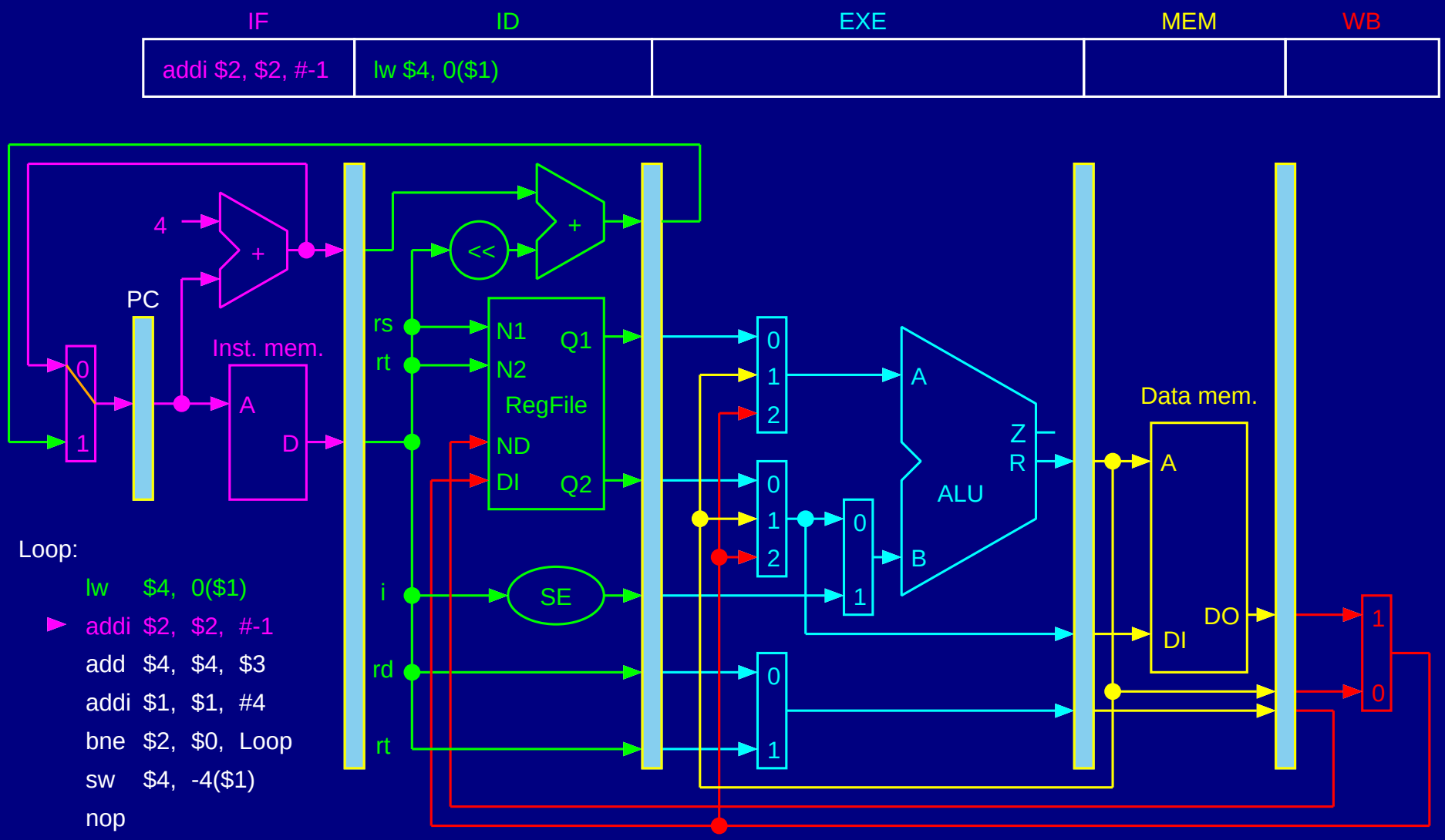
```
Loop:   lw    $4,   0($1)          #   $1: start address of x
        addi  $2,   $2,   #–1      #   $2: counter
        add   $4,   $4,   $3       #   $3: s
        addi  $1,   $1,   #4       #   $1: address + 4
        bne   $2,   $0,   Loop     #   $2: if counter ≠ 0, goto Loop
        sw    $4,   –4($1)         #   $4: x[i] + s, delay slot 1
        nop                        #   delay slot 2
```

Two-slot delayed branch was used.

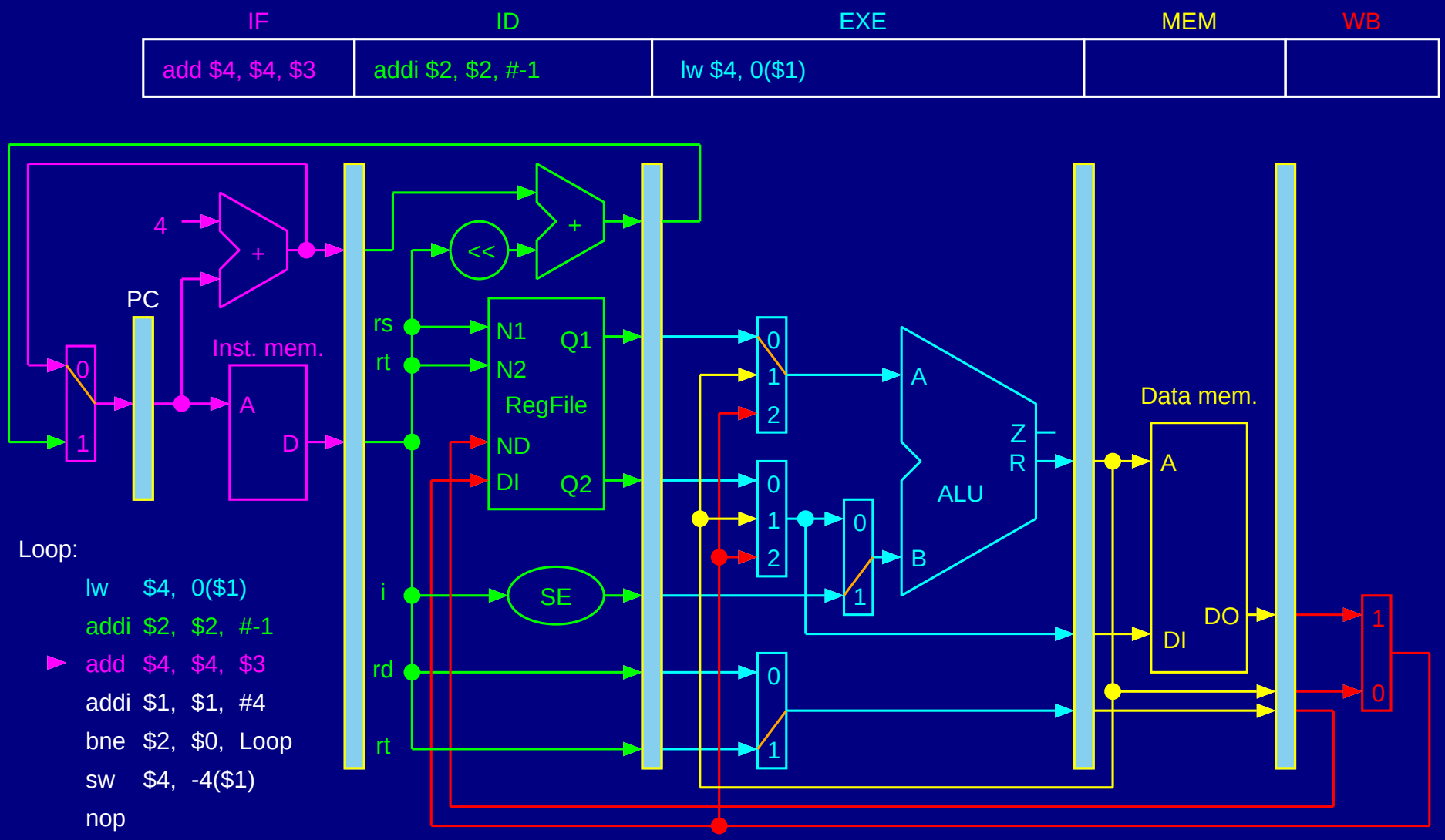
Pipelined MIPS CPU — Cycle 1



Pipelined MIPS CPU — Cycle 2

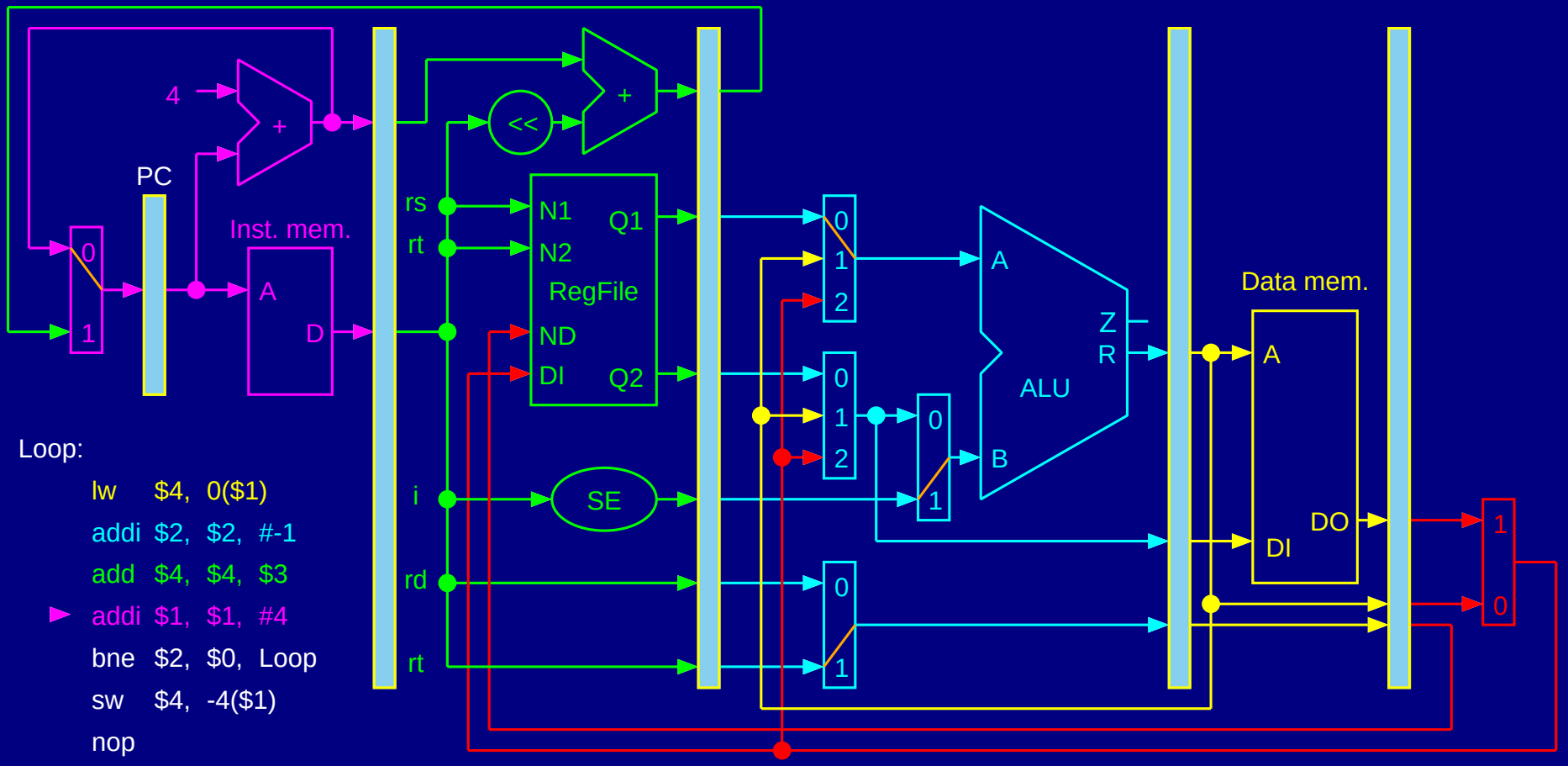


Pipelined MIPS CPU — Cycle 3

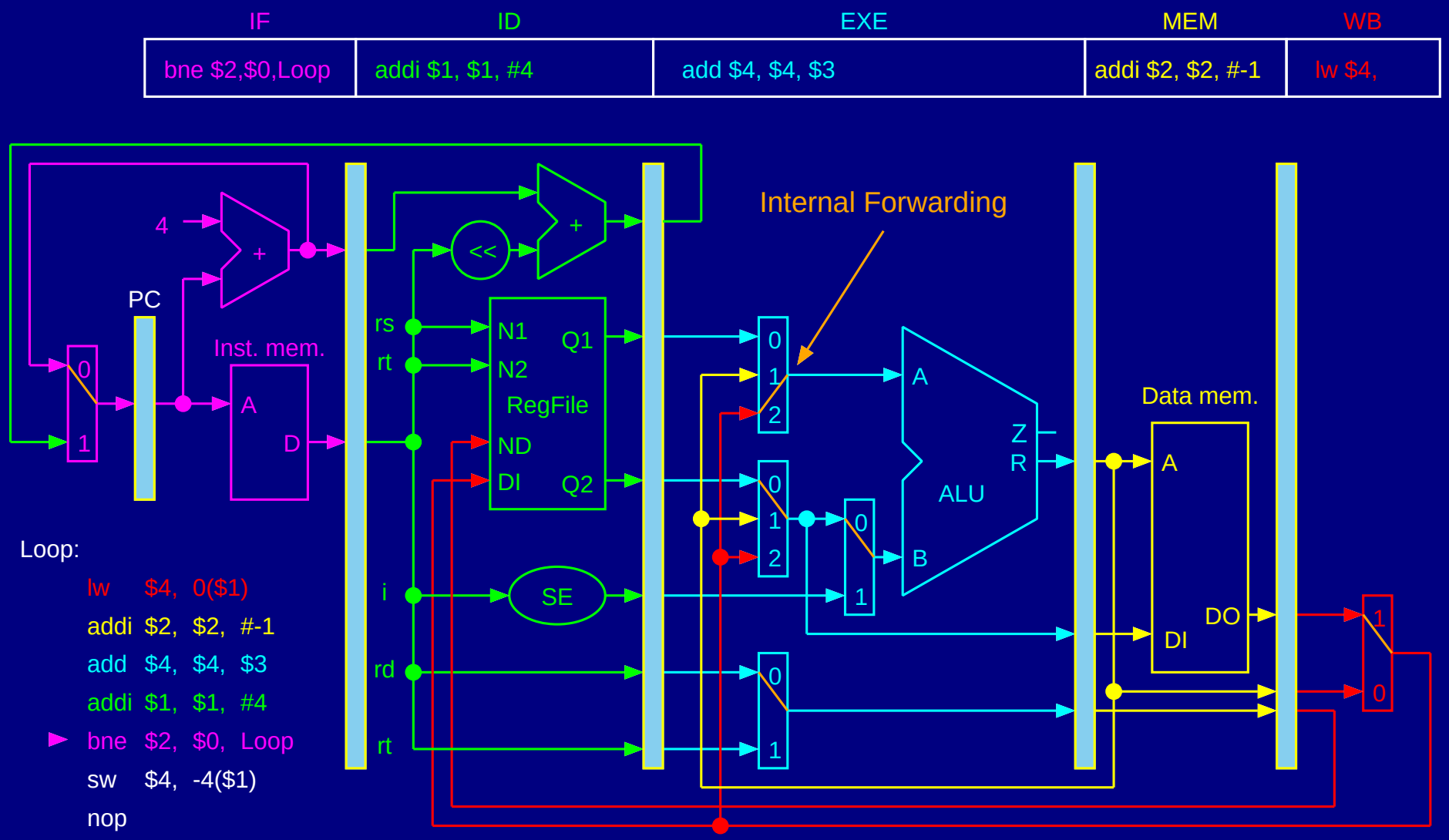


Pipelined MIPS CPU — Cycle 4

IF	ID	EXE	MEM	WB
addi $1, $1, #4	add $4, $4, $3	addi $2, $2, #-1	lw $4, 0($1)	

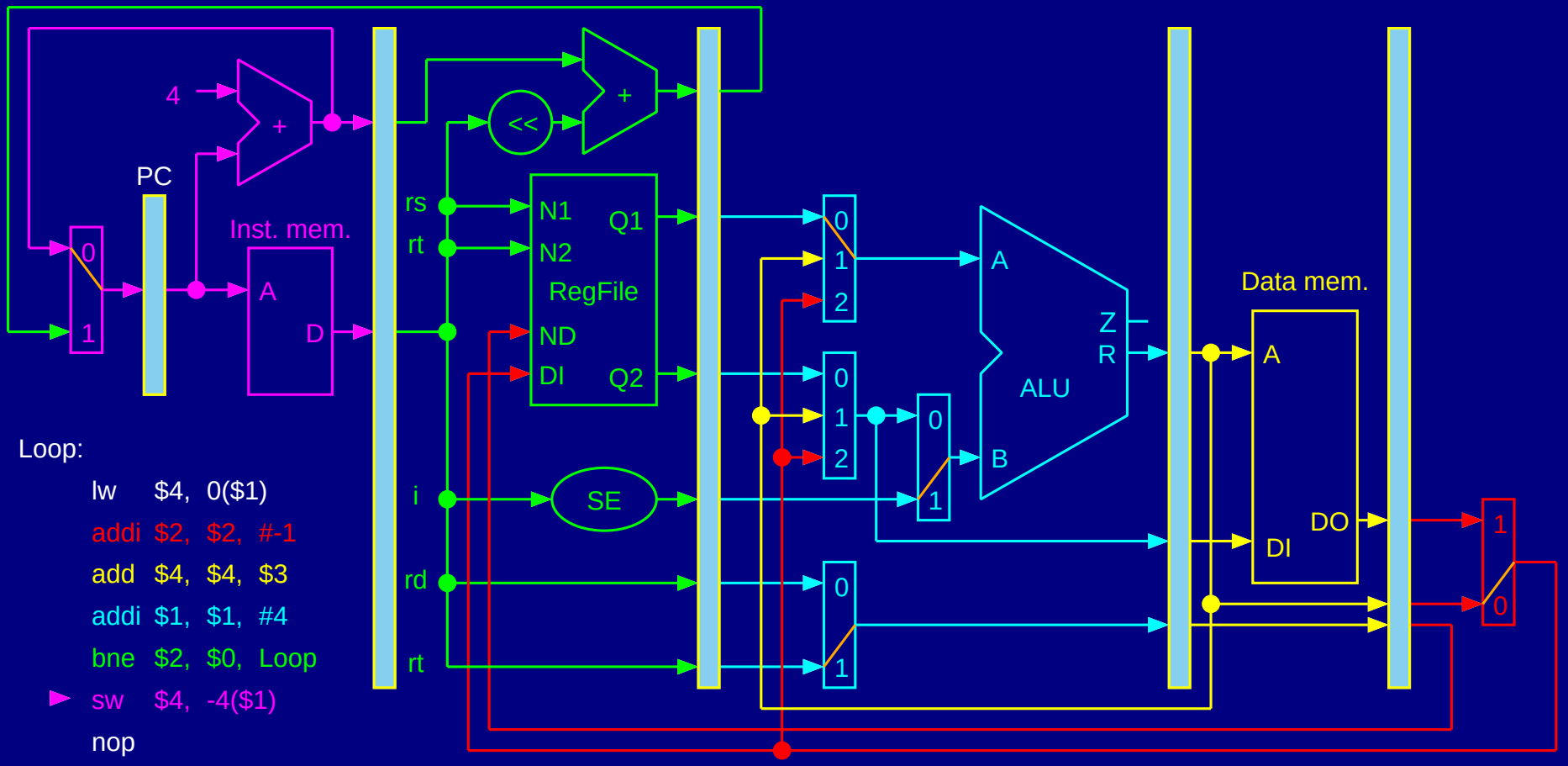


Pipelined MIPS CPU — Cycle 5



Pipelined MIPS CPU — Cycle 6

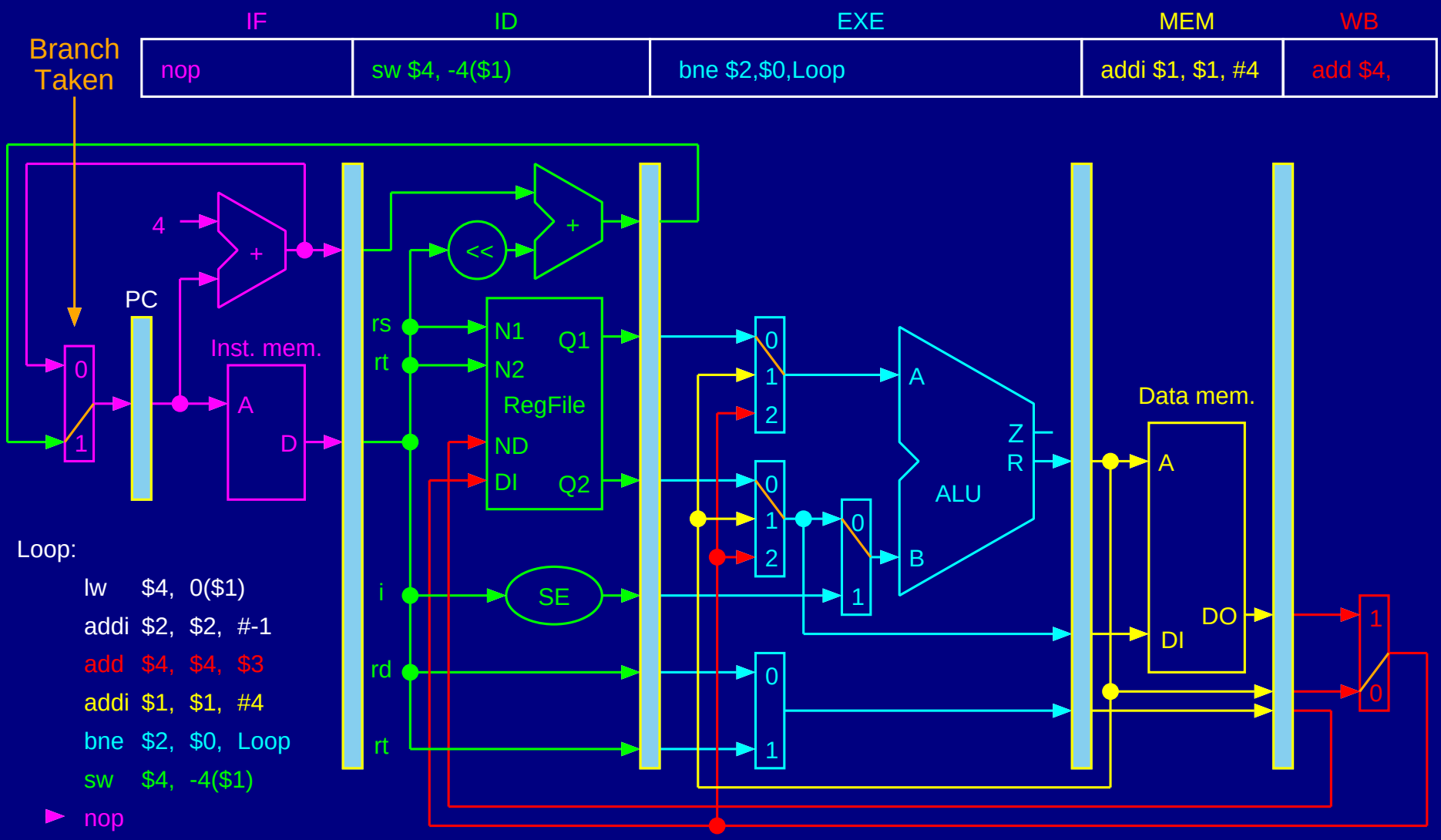
IF	ID	EXE	MEM	WB
sw $4, -4($1)	bne $2,$0,Loop	addi $1, $1, #4	add $4, $4, $3	addi $2,



Loop:

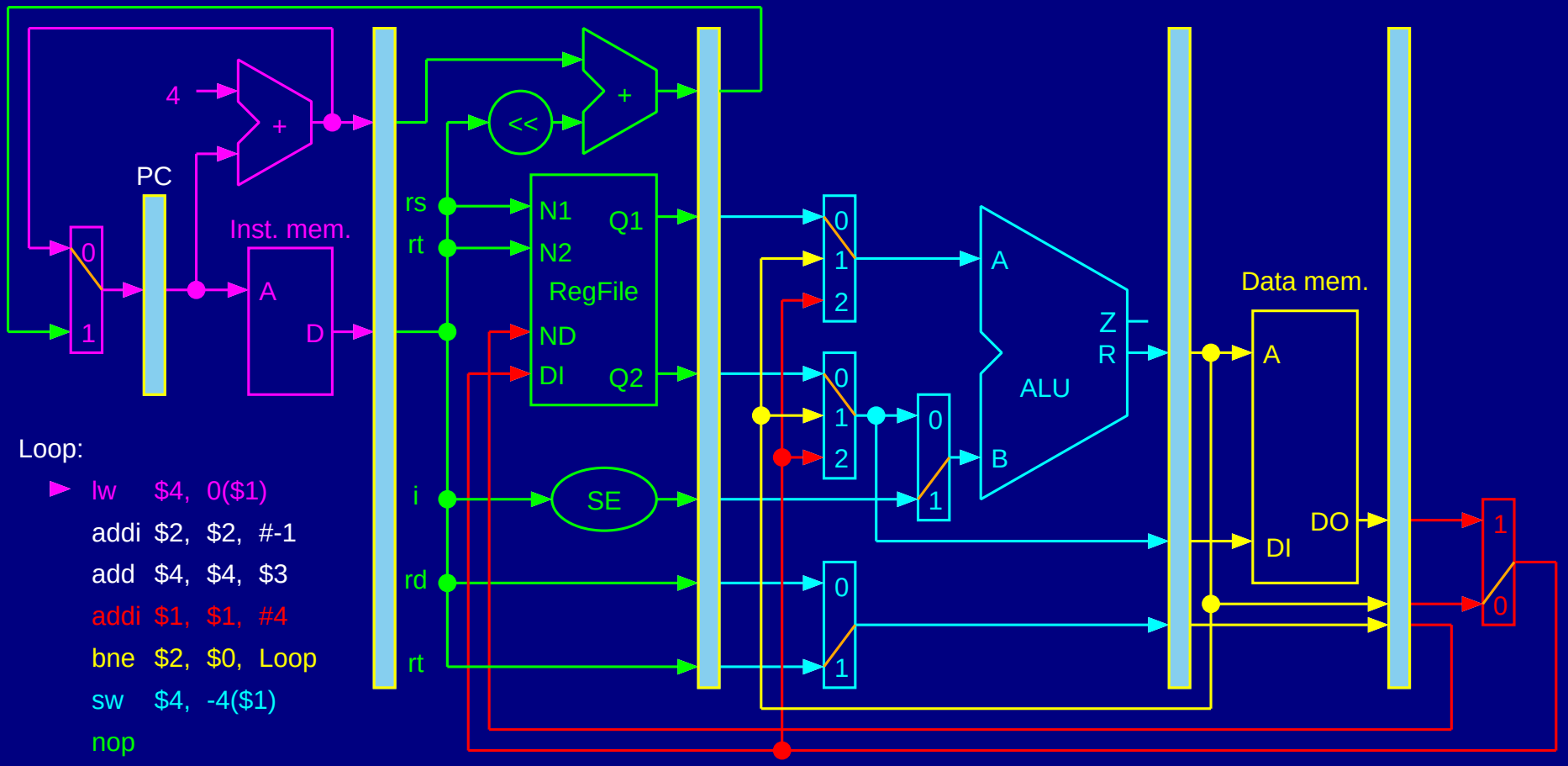
```
    lw   $4, 0($1)
    addi $2, $2, #-1
    add  $4, $4, $3
    addi $1, $1, #4
    bne  $2, $0, Loop
►   sw   $4, -4($1)
    nop
```

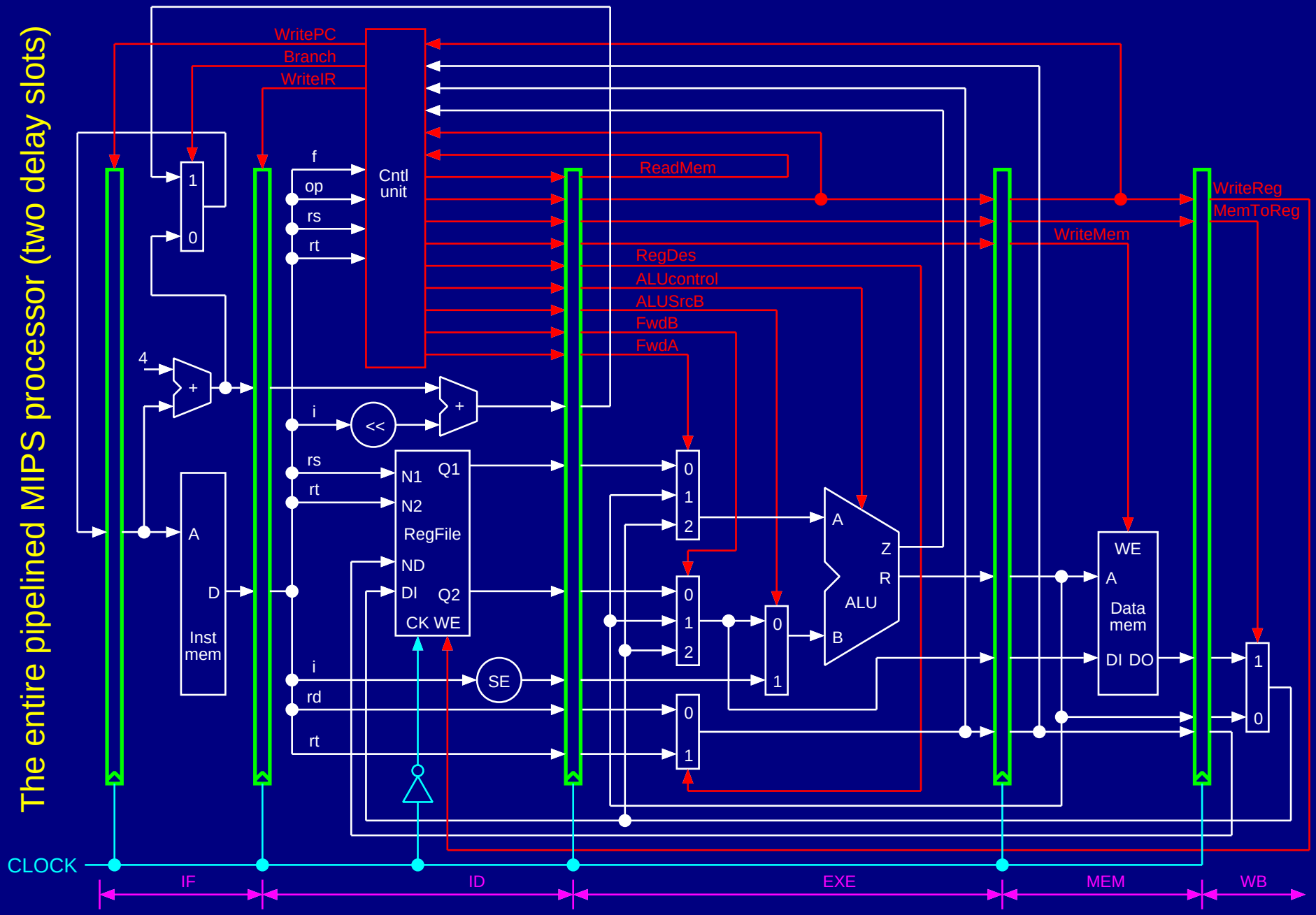
Pipelined MIPS CPU — Cycle 7



Pipelined MIPS CPU — Cycle 8

IF	ID	EXE	MEM	WB
lw $4, 0($1)	nop	sw $4, -4($1)	bne $2,$0,Loop	addi $1,

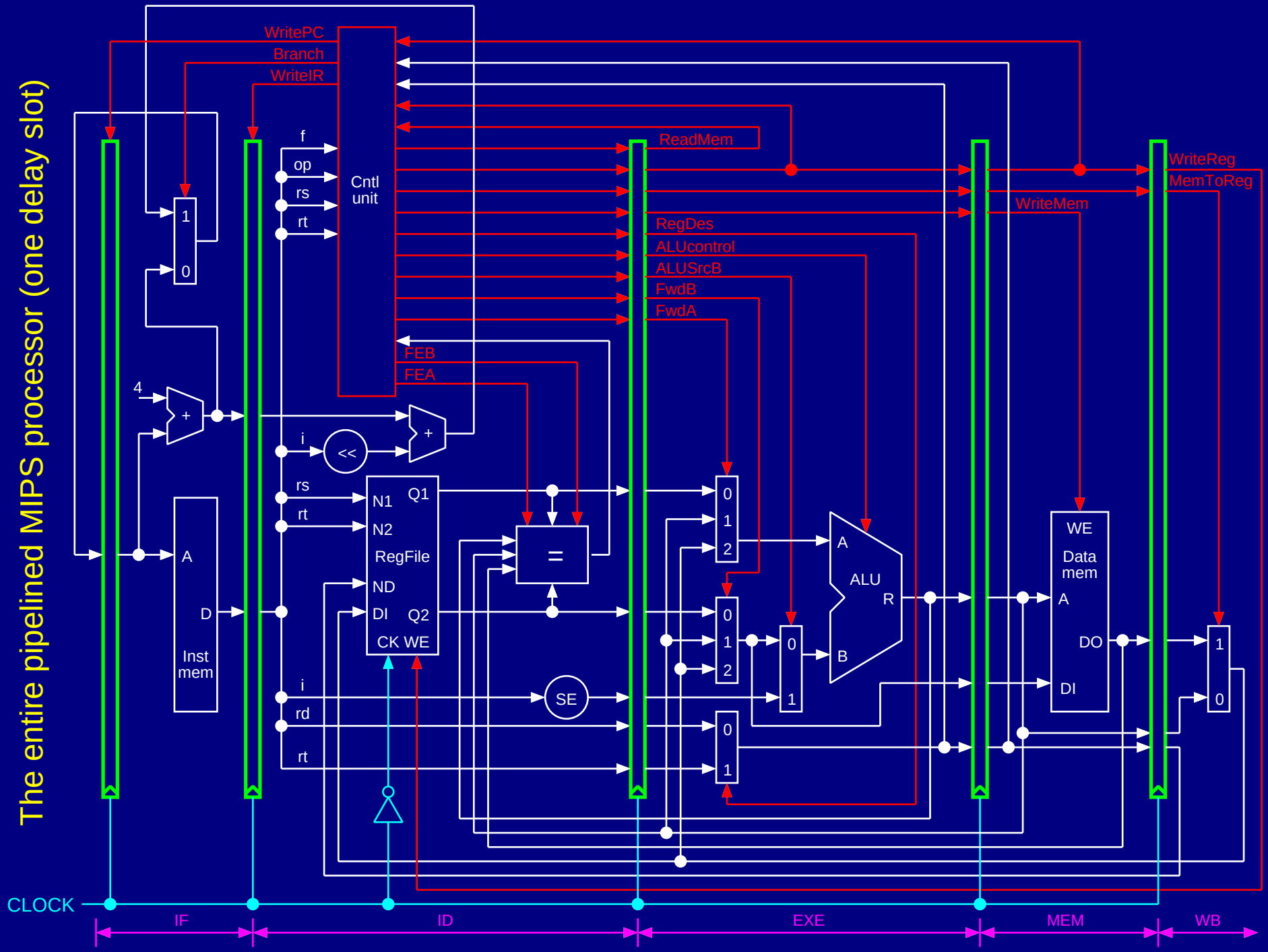



The entire pipelined MIPS processor (two delay slots)
WritePC
Branch
WriteIR
Cntl unit
f
op
rs
rt
ReadMem
WriteReg
MemToReg
WriteMem
RegDes
ALUcontrol
ALUSrcB
FwdB
FwdA
4
+
<<
i
rs
rt
N1
Q1
N2
RegFile
ND
DI
Q2
CK WE
A
D
Inst mem
SE
rd
A
Z
R
ALU
B
WE
Data mem
DI DO
CLOCK
IF
ID
EXE
MEM
WB

Pipelined MIPS Processor

Pipelined MIPS processor with one delay slot — Implementation I

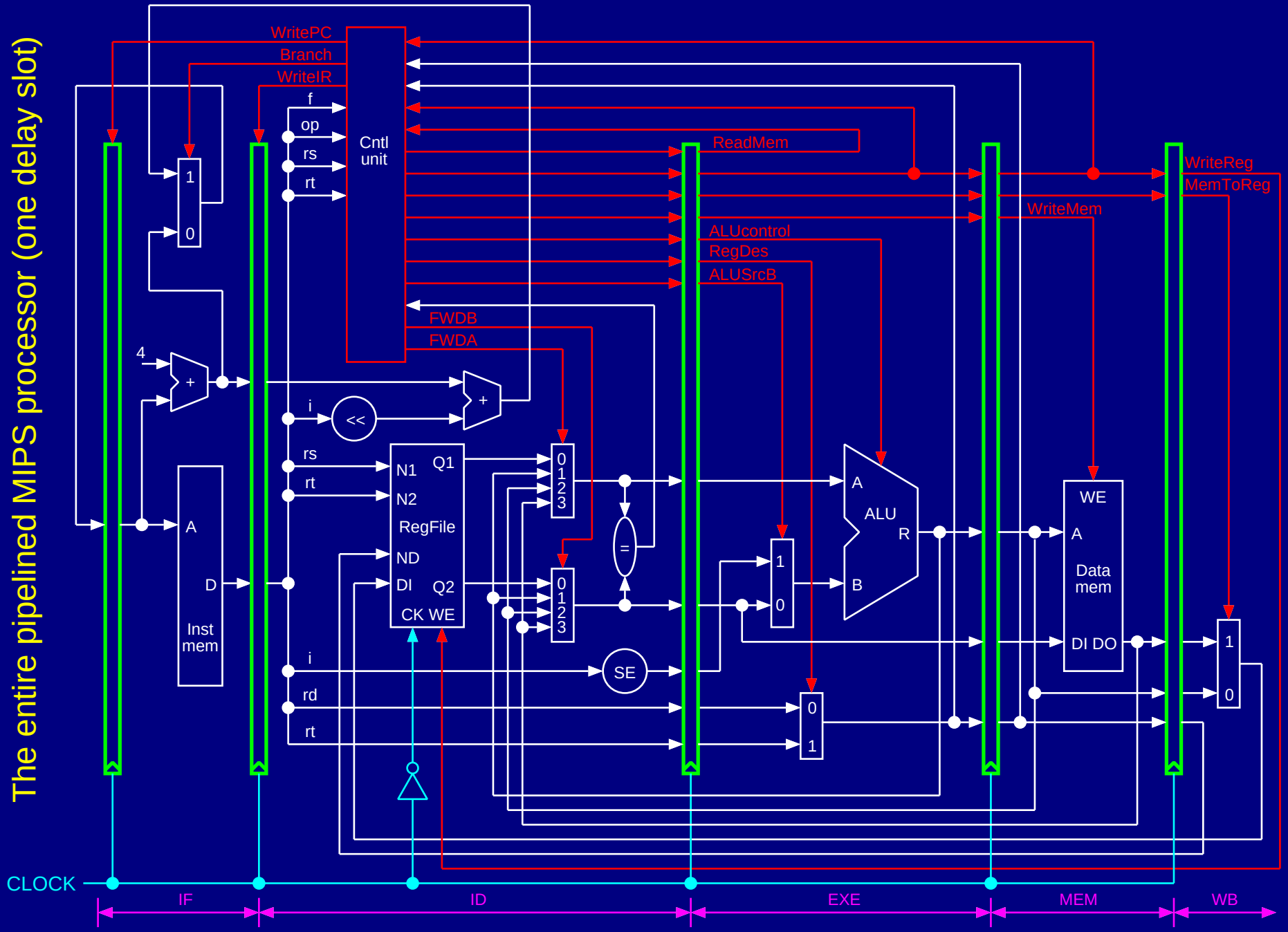
beq is executed in ID stage.


The entire pipelined MIPS processor (one delay slot)
WritePC
Branch
WriteIR
f
op
rs
rt
Cntl unit
ReadMem
WriteReg
MemToReg
WriteMem
RegDes
ALUcontrol
ALUSrcB
FwdB
FwdA
FEB
FEA
1
0
4
+
i
<<
rs
rt
N1
Q1
N2
RegFile
ND
DI
Q2
CK
WE
=
A
D
Inst mem
0
1
2
A
ALU
R
B
WE
Data mem
A
DO
DI
SE
i
rd
rt
CLOCK
IF
ID
EXE
MEM
WB

Pipelined MIPS Processor

Pipelined MIPS processor with one delay slot — Implementation II

beq is executed in ID stage.
Internal forwarding is moved to ID stage.


The entire pipelined MIPS processor (one delay slot)
WritePC
Branch
WriteIR
f
op
rs
rt
Cntl unit
ReadMem
WriteReg
MemToReg
WriteMem
ALUcontrol
RegDes
ALUSrcB
FWDB
FWDA
4
+
i
<<
rs
rt
N1
Q1
N2
RegFile
ND
DI
Q2
CK
WE
A
D
Inst mem
=
SE
A
ALU
R
B
WE
A
Data mem
DI
DO
i
rd
rt
CLOCK
IF
ID
EXE
MEM
WB

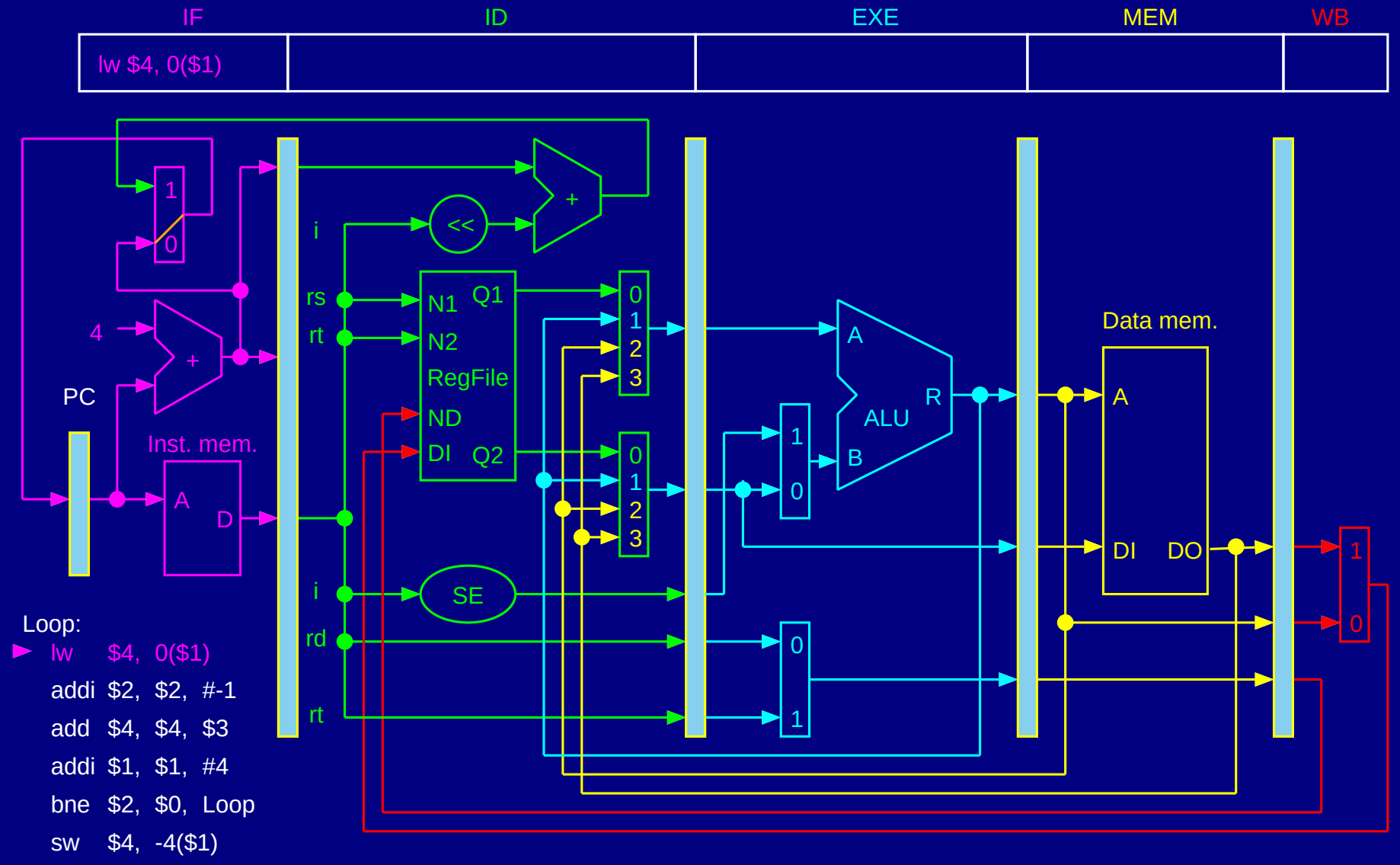
Program Execution on Pipelined MIPS CPU

```
for (i = 0; i < n; i++) {
    x[i] = x[i] + s;
}
```

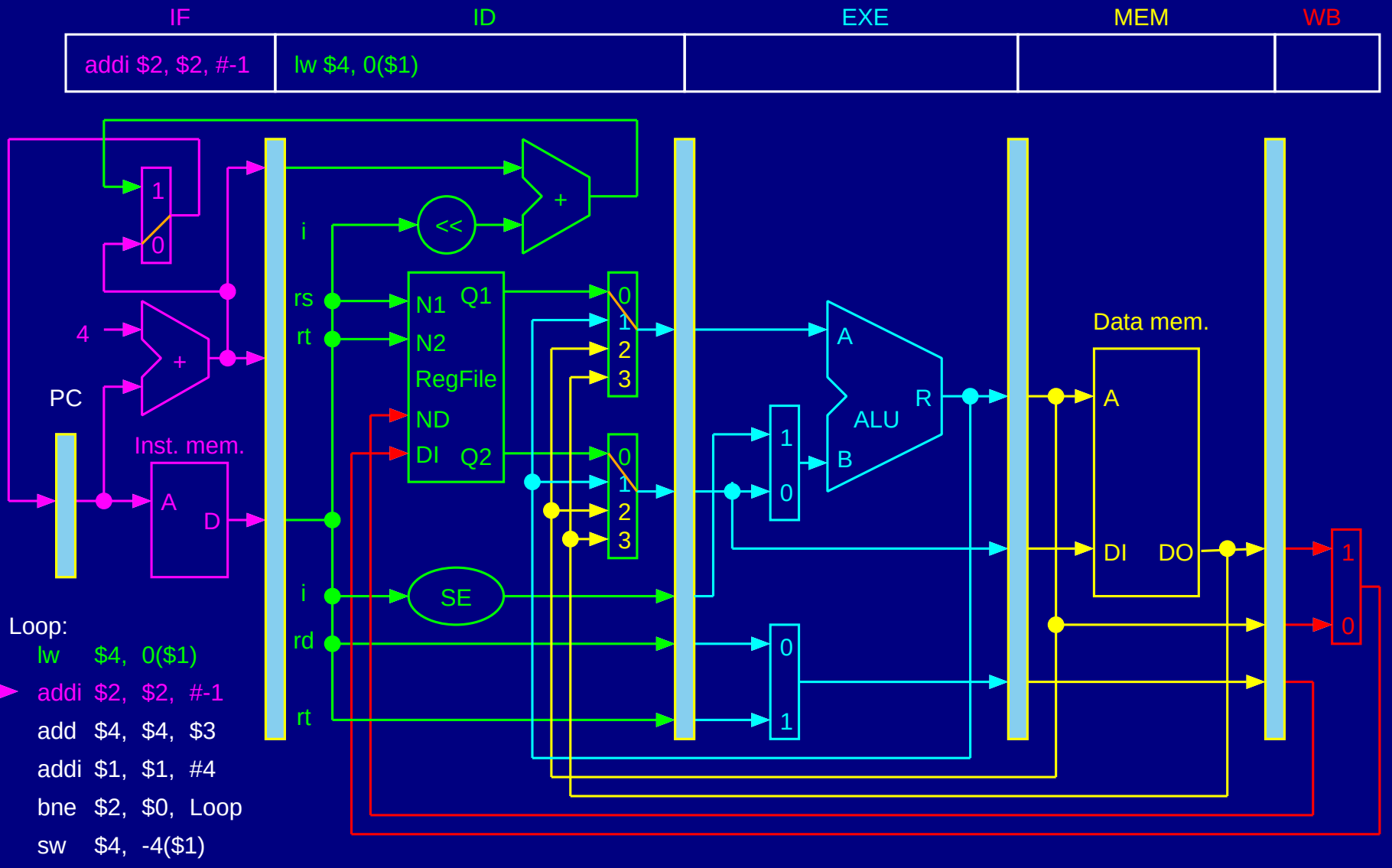
```
Loop:   lw    $4,  0($1)        #  $1: start address of x
        addi  $2,  $2,  #–1     #  $2: counter
        add   $4,  $4,  $3      #  $3: s
        addi  $1,  $1,  #4      #  $1: address + 4
        bne   $2,  $0,  Loop    #  $2: if counter ≠ 0, goto Loop
        sw    $4,  –4($1)       #  $4: x[i] + s, delay slot
```

One-slot delayed branch was used.

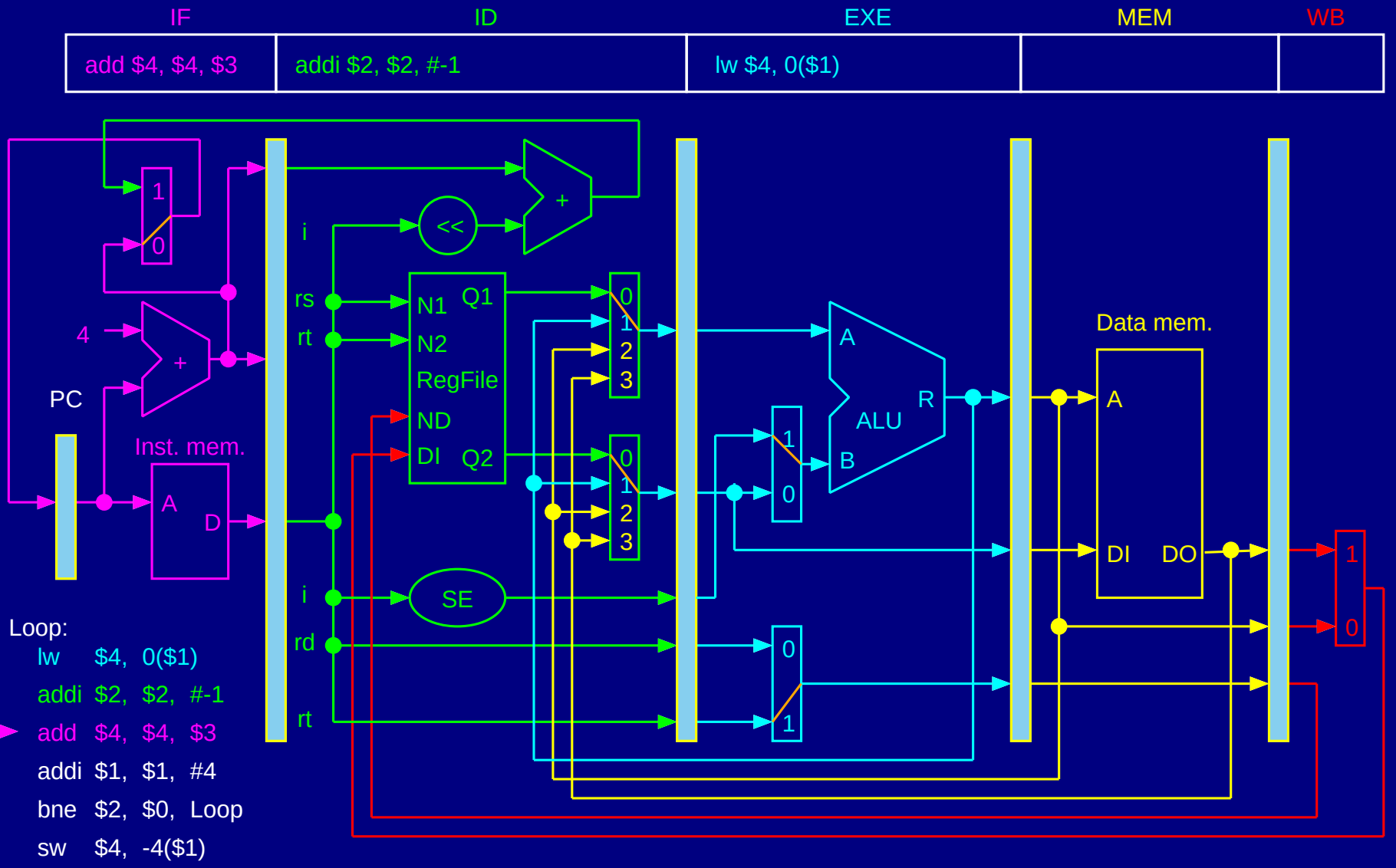
Pipelined MIPS CPU — Cycle 1



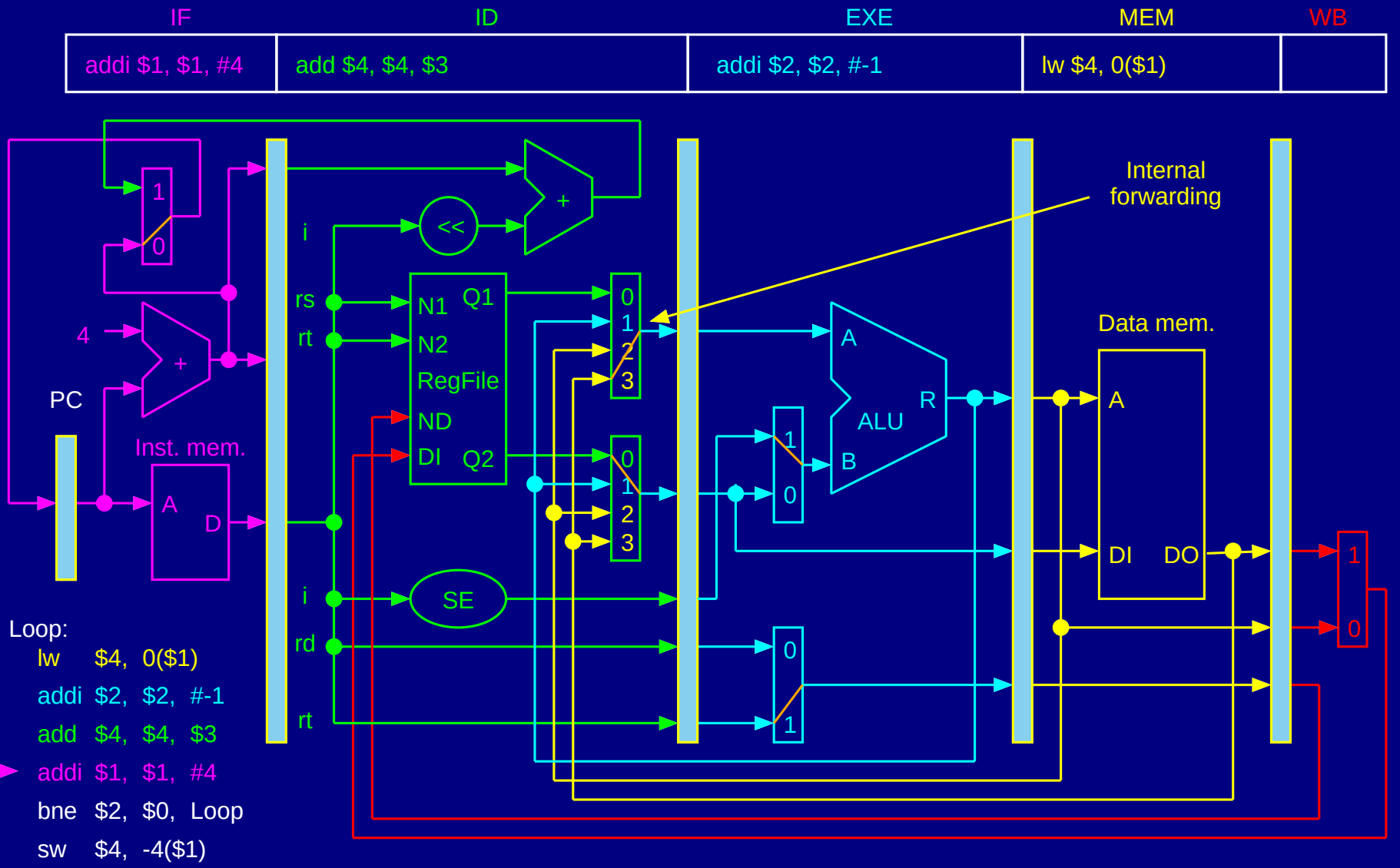
Pipelined MIPS CPU — Cycle 2



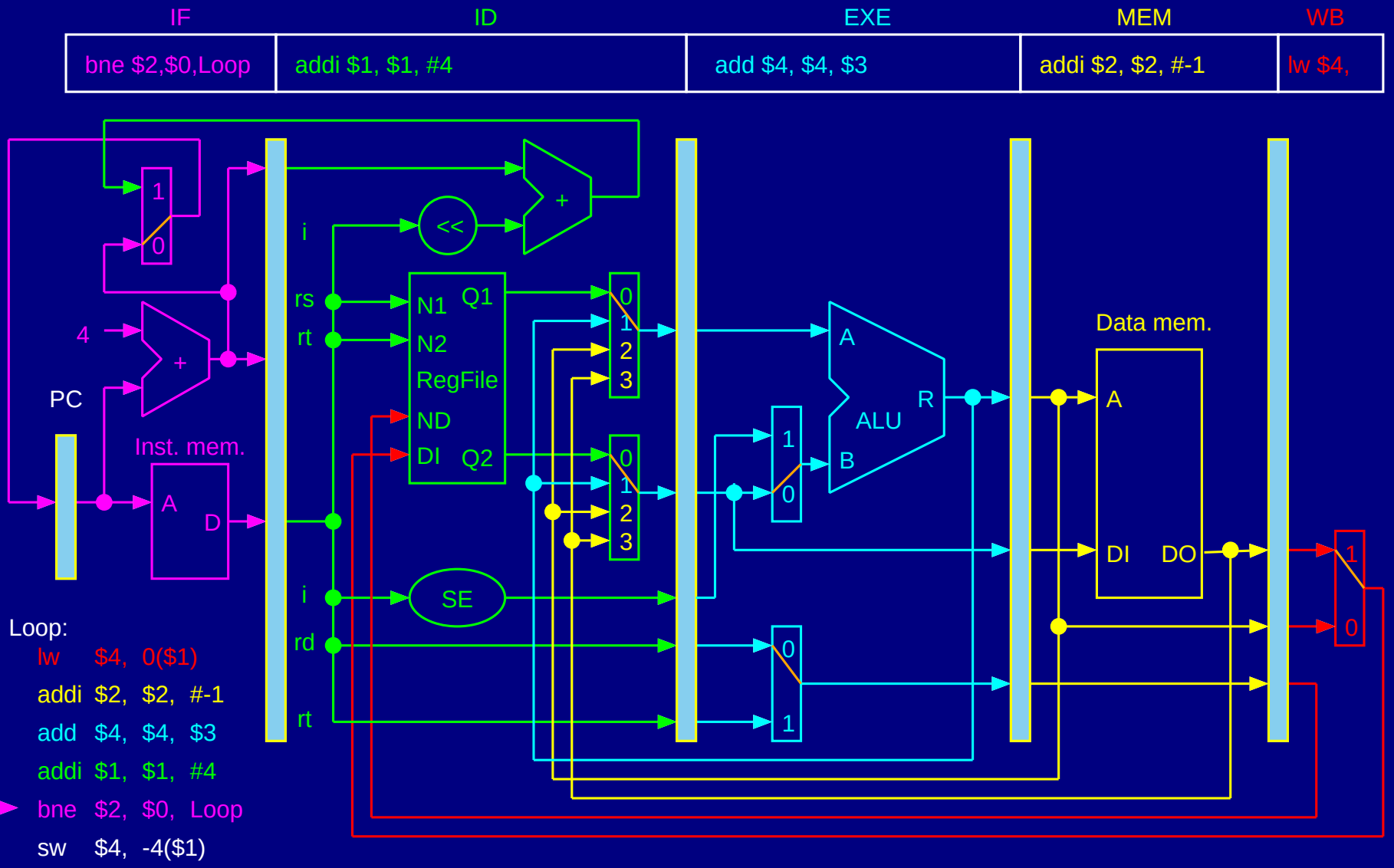
Pipelined MIPS CPU — Cycle 3



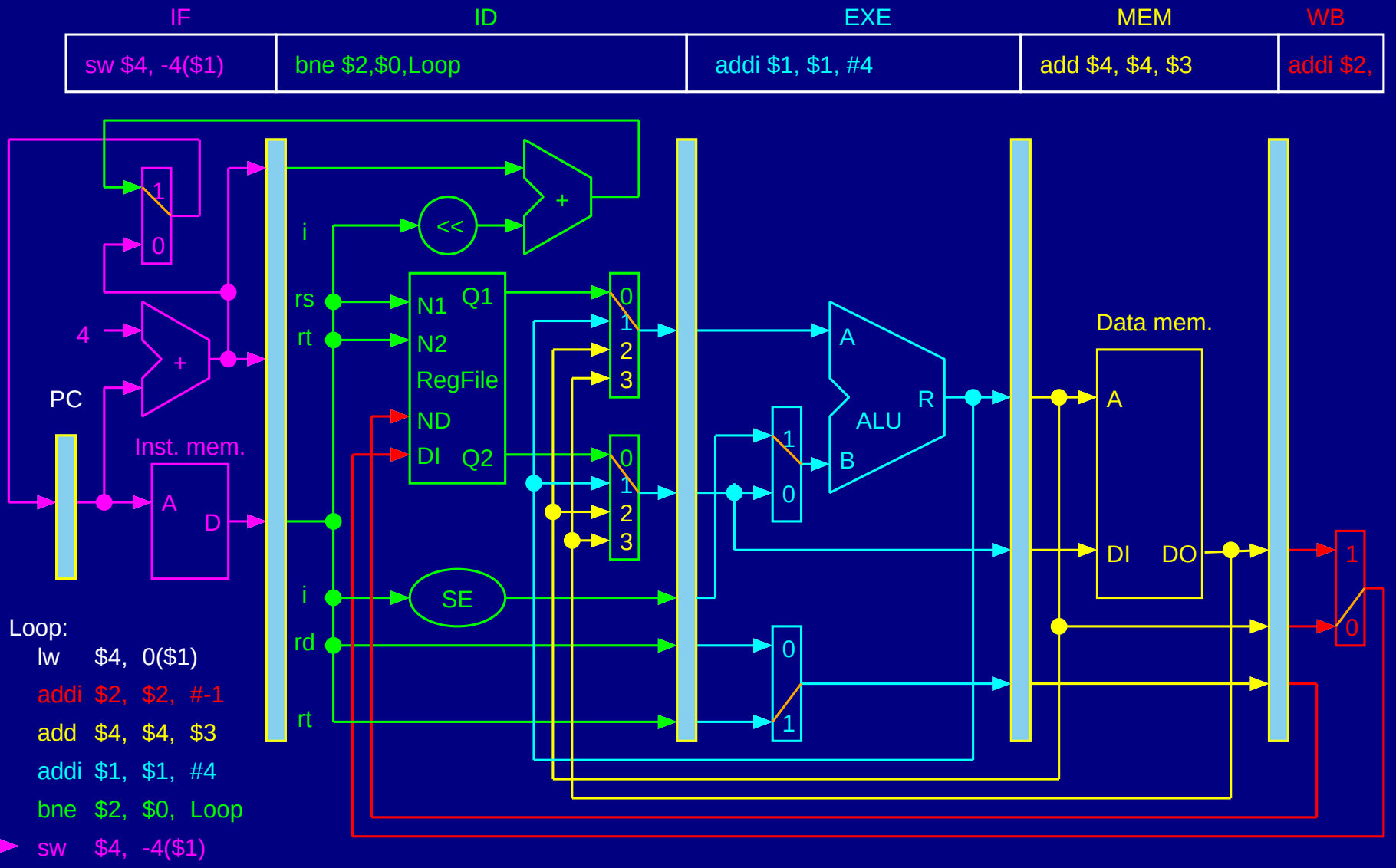
Pipelined MIPS CPU — Cycle 4



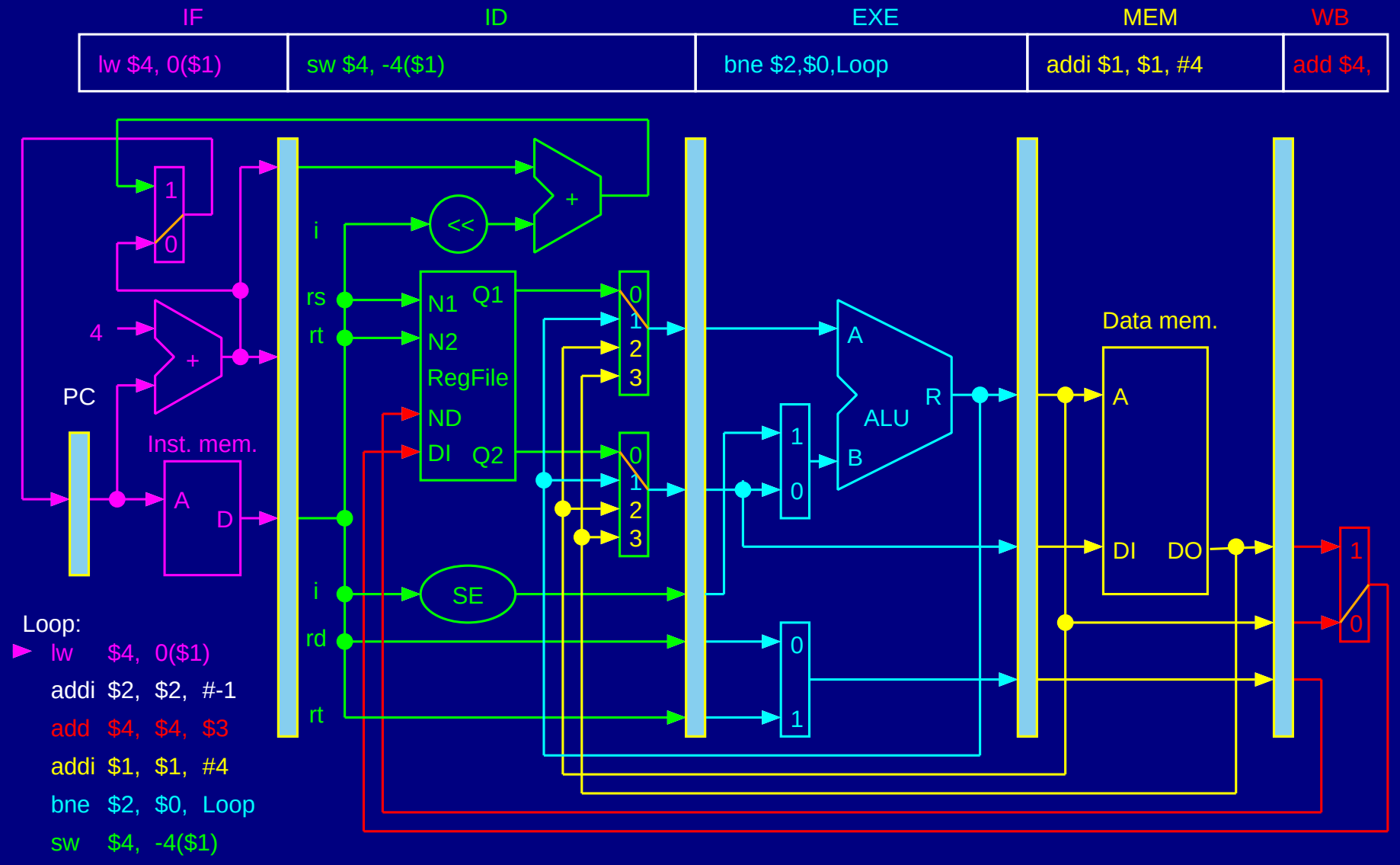
Pipelined MIPS CPU — Cycle 5

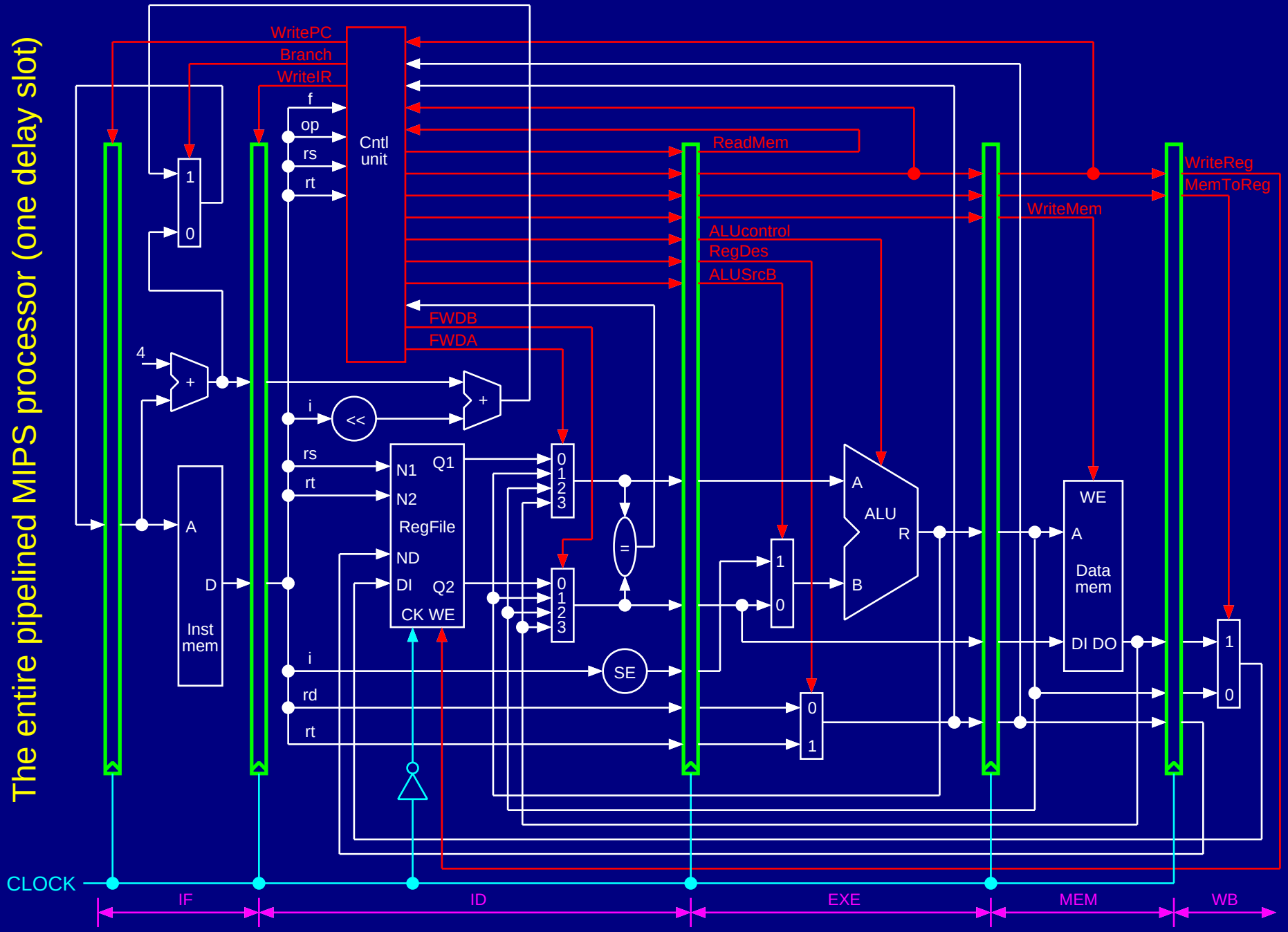


Pipelined MIPS CPU — Cycle 6



Pipelined MIPS CPU — Cycle 7




The entire pipelined MIPS processor (one delay slot)
WritePC
Branch
WriteIR
f
op
rs
rt
Cntl unit
ReadMem
WriteReg
MemToReg
WriteMem
ALUcontrol
RegDes
ALUSrcB
FWDB
FWDA
4
+
<<
i
rs
rt
N1
Q1
N2
RegFile
ND
DI
Q2
CK
WE
A
D
Inst mem
=
SE
rd
ALU
R
B
WE
Data mem
DI
DO
CLOCK
IF
ID
EXE
MEM
WB

Exercise 1

Design a Pipelined MIPS CPU that can execute the following 13 instructions:

- add, sub, and, or, slt, lw, sw, beq, j, bne, addi, andi, ori.

```
addi $17, $18, −1 # $17=$18−1
```

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Immediate
8	18	17	−1
0 0 1 0 0 0	1 0 0 1 0	1 0 0 0 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Exercise 2

- Write a program to perform $C = A \times B$ using the instruction set described in a pipelined processor visual simulator found in

  http://cis.k.hosei.ac.jp/~yamin/applets/pipeviewer/

  and simulate your program using the simulator. You can assume that A and B are in the memory, use load (`ld`) instruction to read A and B into register file, and use store (`st`) to write C back to the memory.