第3章 流水线模型机

- C.1 流水线的基本概念
- C.2 流水线的主要障碍---流水线冒险
- C.3 流水线处理机设计
- C.4 异常事件处理
- C.5 扩展流水线到多周期操作

程序例子的执行时序图

instruction	1	2	3	4	5	6	7	8	9	10
addi r1, r1, 4	addi	r1,	+		r1					
lw r2, 100(r3)		lw	r3,	+	mem	r2				
sub r4, r5, r6			sub	r5,r6	-		r4			
add r7, r5, r6				add	r5, r6	+		r7		
sw r8, 200(r9)					sw	r9, r8	+	mem		
addi r10, r10, -1						addi	r10,	-		r10

Stage	1	2	3	4	5	6	7	8	9	10
IF	addi	Lw	sub	add	sw	addi				
ID		r1,	r3,	r5, r6	r5, r6	r9,	r10,			
EXE			+	+	-	+	+	+		
MEM					mem			mem		
WB					r1	r2	r4	r7		r10

C.3 流水线处理机及其设计

- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

C.3.1 流水线处理机的指令系统

- 流水线(pipeline)是一种能够使多条指令重叠执行的处理机的实现技术，它已成为现代处理机设计中最为关键的技术。

20条MIPS指令格式

```
op      rs      rt      rd      sa      func    ; R format instructions
000000  rs      rt      rd      00000   100000  ; add   rd, rs, rt
000000  rs      rt      rd      00000   100010  ; sub   rd, rs, rt
000000  rs      rt      rd      00000   100100  ; and   rd, rs, rt
000000  rs      rt      rd      00000   100101  ; or    rd, rs, rt
000000  rs      rt      rd      00000   100110  ; xor   rd, rs, rt
000000  00000   rt      rd      sa      000000  ; sll   rd, rt, sa
000000  00000   rt      rd      sa      000010  ; srl   rd, rt, sa
000000  00000   rt      rd      sa      000011  ; sra   rd, rt, sa
000000  rs      00000   00000   00000   001000  ; jr    rs
op      rs      rt      imm                     ; I format instructions
001000  rs      rt      imm                     ; addi rt, rs, imm
001100  rs      rt      imm                     ; andi rt, rs, imm
001101  rs      rt      imm                     ; ori  rt, rs, imm
001110  rs      rt      imm                     ; xori rt, rs, imm
100011  rs      rt      imm                     ; lw   rt, imm(rs)
101011  rs      rt      imm                     ; sw   rt, imm(rs)
000100  rs      rt      imm                     ; beq  rs, rt, imm
000101  rs      rt      imm                     ; bne  rs, rt, imm
001111  00000   rt      imm                     ; lui  rt, imm
op      addr                                    ; J format instructions
000010  addr                                    ; j    addr
000011  addr                                    ; jal  addr
```

例

- 指令执行时用到的主要功能部件和它们所需的时间如下：
- 指令存储器和数据存储器：10ns；
- ALU和地址加法器：10ns；
- 寄存器堆：5ns；
- 假定其它部件，如PC寄存器、多路器、控制部件等花费的时间为0。

各类指令执行所需要的时间

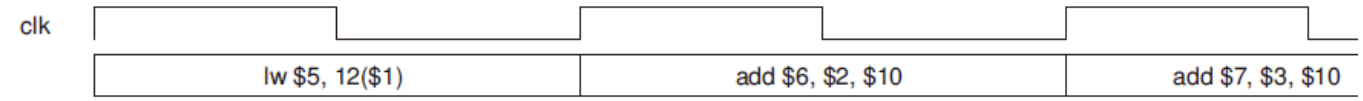
	10	5	10	10	5	
ALU指令	指令存储器	寄存器堆	ALU		寄存器堆	30
load指令	指令存储器	寄存器堆	ALU	数据存储器	寄存器堆	40
store指令	指令存储器	寄存器堆	ALU	数据存储器		35
转移指令	指令存储器	加法器				20

单周期处理机的时钟周期：40ns
多周期处理机的时钟周期：10ns
流水线处理机的时钟周期：10ns

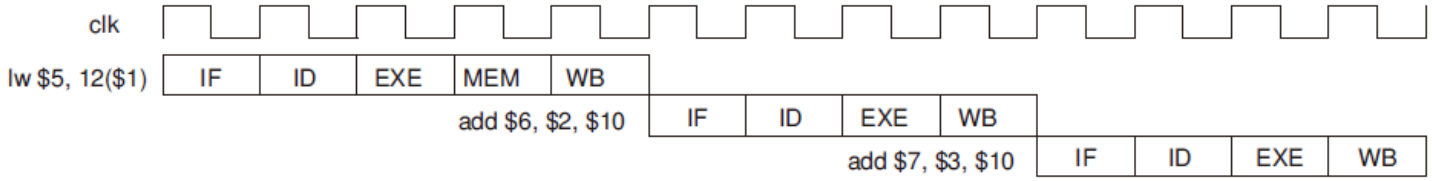
使用流水线/多周期技术后指令执行所需要的时间

	10	10	10	10	10	
ALU指令	指令存储器	寄存器堆	ALU		寄存器堆	50
load指令	指令存储器	寄存器堆	ALU	数据存储器	寄存器堆	50
store指令	指令存储器	寄存器堆	ALU	数据存储器		40
转移指令	指令存储器	加法器				20

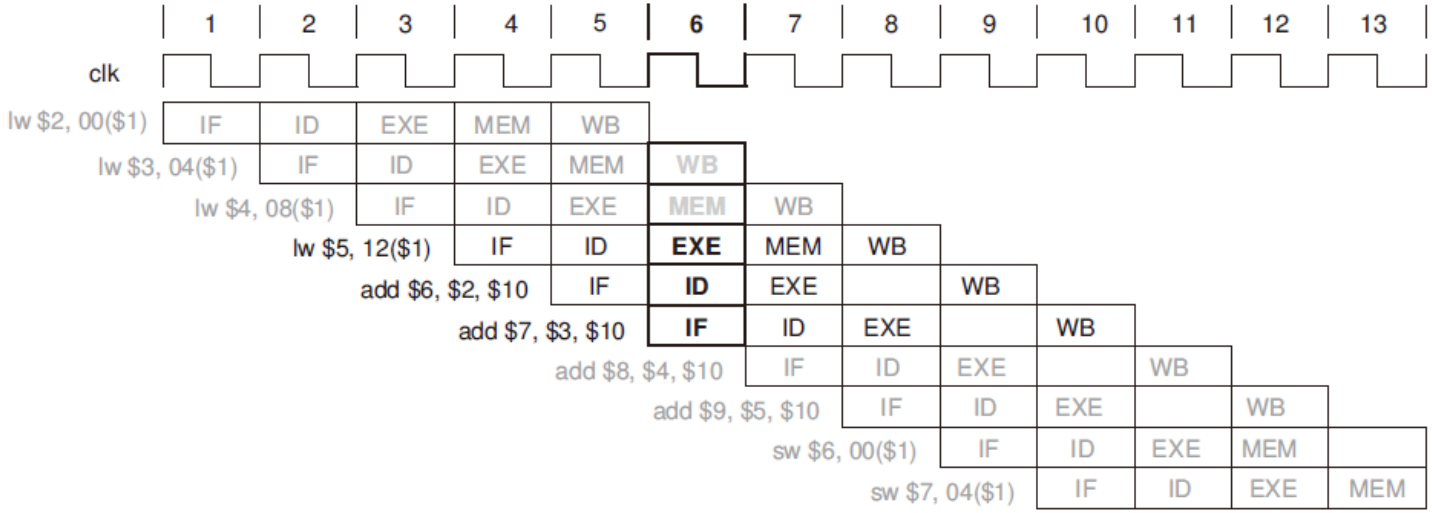
三种处理机执行时序比较



(a) Timing chart of single cycle CPU



(b) Timing chart of multiple cycle CPU



(c) Timing chart of pipelined CPU

C.3 流水线处理机及其设计

- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

C.3.2 流水线处理机的数据路径

流水线处理机的数据通路结构与工作原理。

一、流水线级

非流水线单周期处理机组成结构如后图

把执行指令的过程分为5部分，使其能够按流水线方式执行指令：

（1）IF(instruction fetch)取指令；

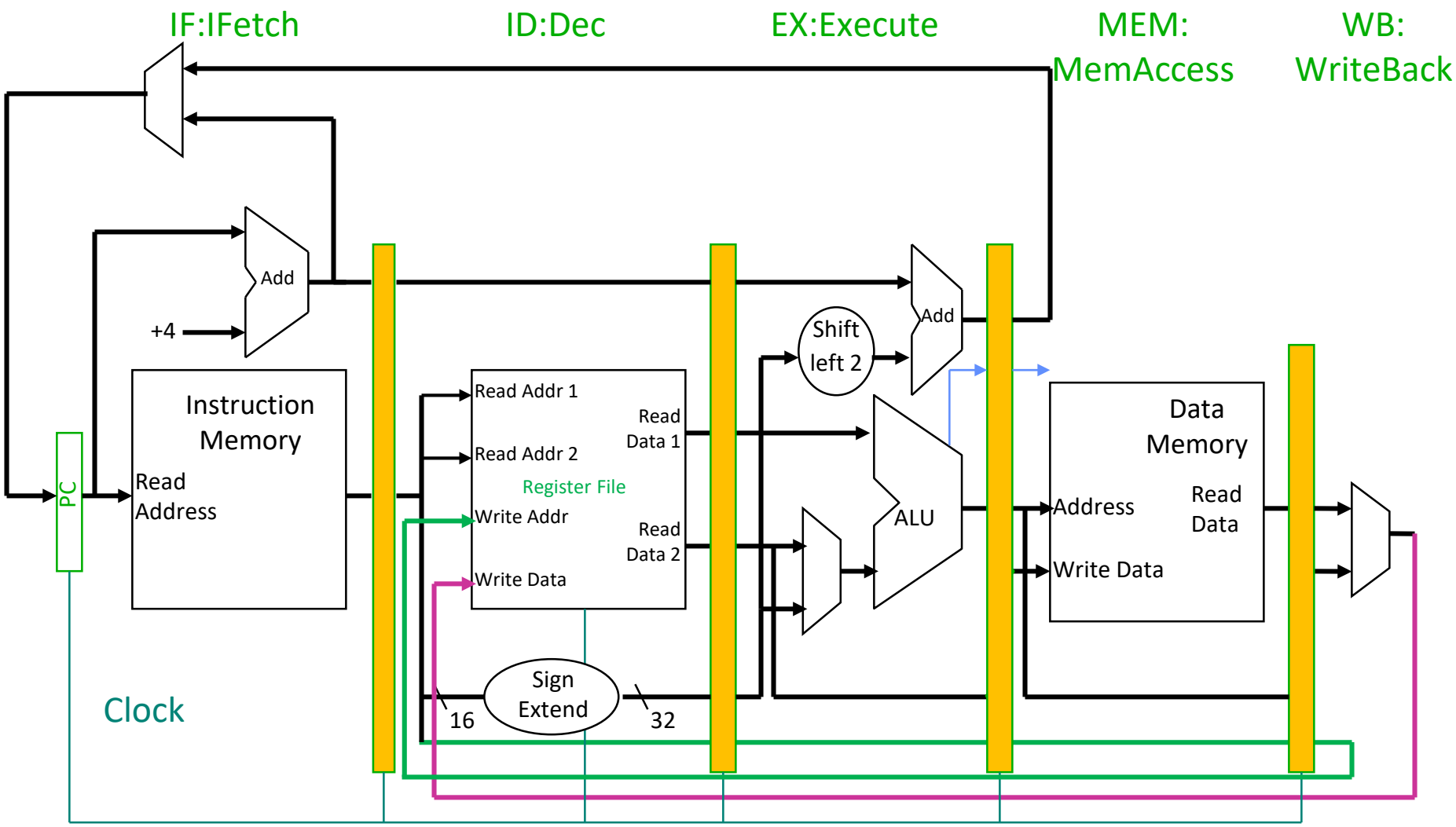
（2）ID(instruction decode)指令译码并读寄存器操作数；

（3）EXE(execution)执行；

（4）MEM(memory access)存储器访问；

（5）WB(write back)写回。

简化的流水线数据通路


IF:IFetch
ID:Dec
EX:Execute
MEM:
MemAccess
WB:
WriteBack
Add
+4
PC
Instruction
Memory
Read
Address
Read Addr 1
Read Addr 2
Register File
Write Addr
Write Data
Read
Data 1
Read
Data 2
Shift
left 2
Add
ALU
Data
Memory
Address
Read
Data
Write Data
Sign
Extend
16
32
Clock

- 指令执行从左移向右通过电路。
- 但有两处从右向左的例外：
- WB级：把运算结果写回寄存器堆中；
- IF级：把经过计算的下一条指令的地址写入程序计数器PC中。下一条指令地址的计算方法有两种：
  * 一种是当前PC+4；
  * 另一种是当前PC+4+字地址偏移量，
- 偏移量在转移指令中定义出。

流水线处理机

- 特点：每一个时钟周期取出一条指令来执行。所有的指令按取出的先后次序通过数据路径。
- 依照指令类型的不同，每种指令在执行过程中可能会用到数据路径的不同的部分。
- 例如：load指令实际上是使用功能部件最多的指令

流水线寄存器的引入

- 在单周期处理机中，如果一条指令还没有执行完毕，PC的内容不会改变。这就使得在一条指令的整个执行过程中，IM始终输出当前指令。
- 流水线处理机每个时钟周期都要取出一条指令。因此，当流水线处理机已从存储器取出一条指令并把它送到ID级去译码时，下一条指令也正在从指令存储器中取出。如果先取出的指令没被保存，则它后面正在被取出的指令会对它造成影响。也就是说，必须要使用寄存器来保存从存储器取出的指令。
- 推而广之，我们必须要在流水线的各级之间安排一组寄存器，用以保存当前时钟周期各流水级操作的结果，以便为下个周期使用（我们只能使用触发器寄存器，它把时钟上升沿时数据输入端的信息打入寄存器中；而不能使用锁存器，因为锁存器的输出在时钟高电平时跟随输入的变化而变化。）我们称这些寄存器为流水线寄存器。

- PC（32位）：一个特殊的流水线寄存器，非流水线处理机中PC也是需要的。由于流水线处理机每个时钟周期都从指令存储器取出一条指令，它的值在每次周期结束时都将被改变。
- 程序不发生转移时，PC+4；（字地址加1，字节地址是加4）
- 条件转移时，PC+4+符号扩展的偏移量。
- 指令寄存器IR（32位）：第一级与第二级之间的流水线寄存器。

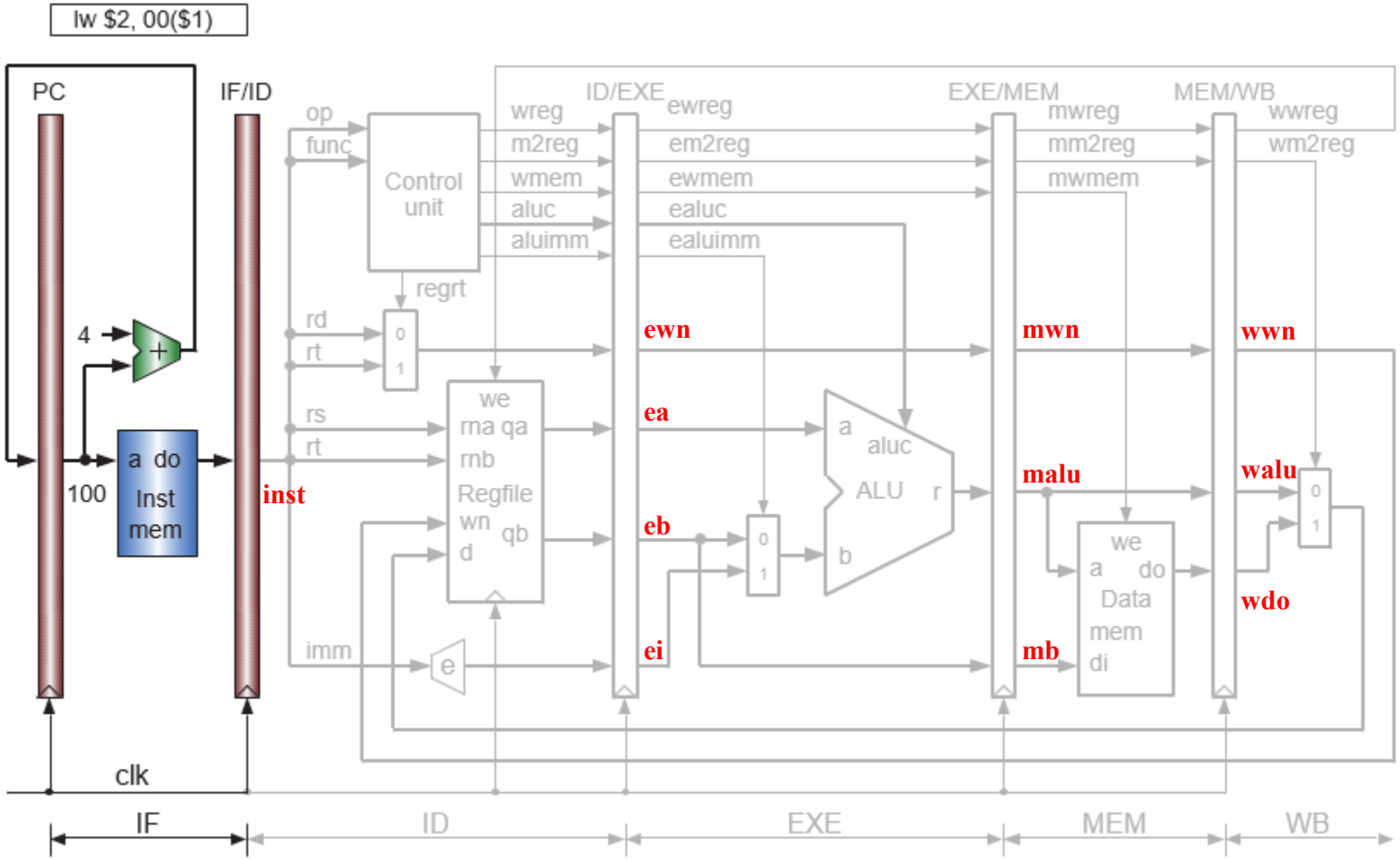
- 第二和第三级之间需要较多的寄存器：
- 从寄存器堆中读出的两个32位数据A和B必须要保存。
- 经符号扩展后的32位立即数也要保存。
- 一个5位寄存器，它被用来保存目的寄存器号；因为指令的操作结果要在WB级写入寄存器堆，目的寄存器号也要在那时使用，因此必须要同步跟随过去。

- 一般的ALU操作指令，要么是A与B操作，要么是A与I操作。我们能不能在ID级增加一个多路器而省去B或I寄存器，只使用一个32位寄存器呢?答案是否定的。原因出在store指令上。store指令同时使用A，B和I。A和I用来计算存储器地址，B用来保存写入存储器的数据。在流水线处理机中，这些数据必须要。在一个时钟周期中同时产生，以保证不引起资源冲突。

- 第四级和第五级之间还有如下的寄存器：
- MemoryReadOutData （32位）：存放load指令从存储器中读出的数据；
- ALUResult （32位）：保存ALU的运算结果。

IF级：取指令级

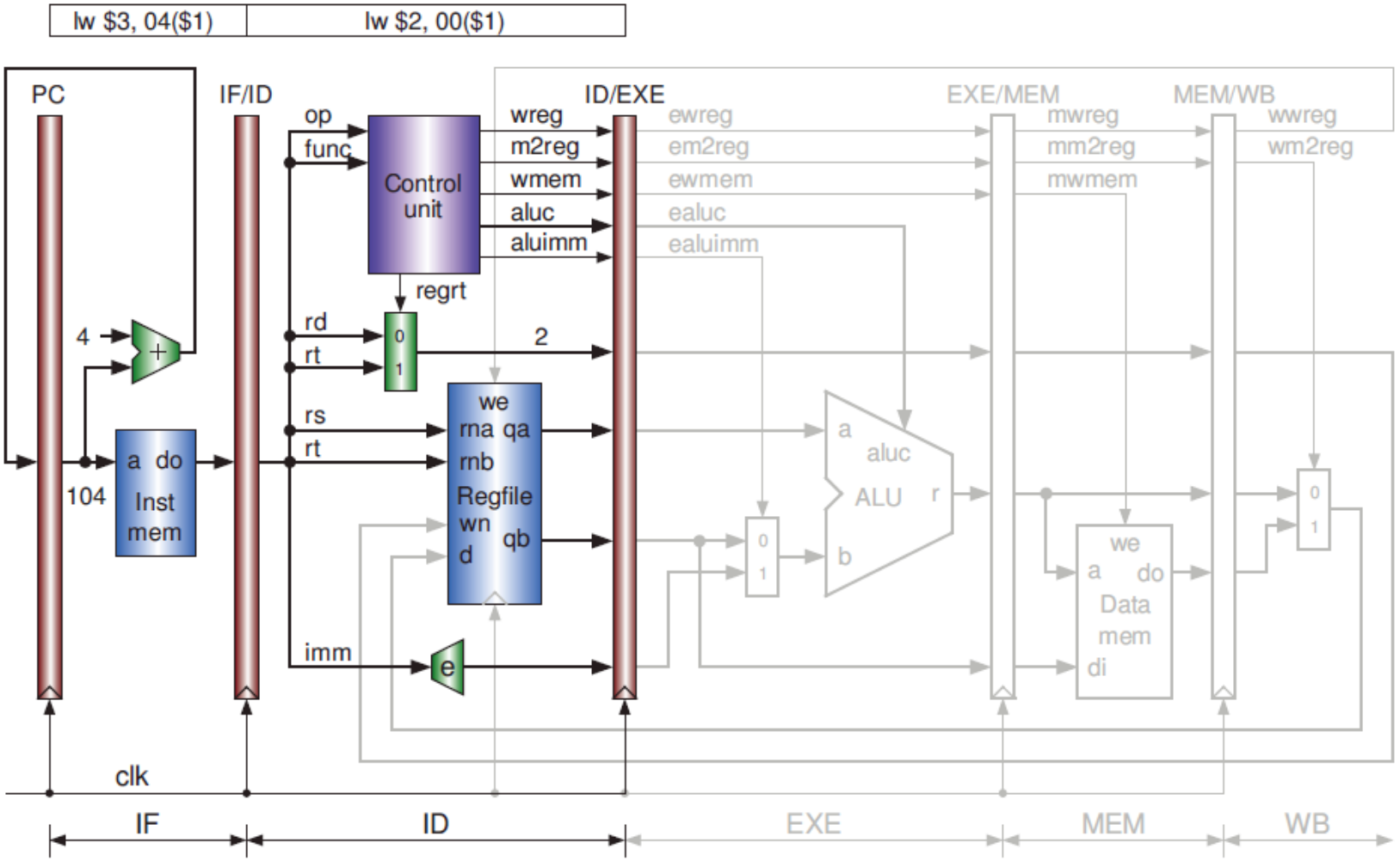
- 使用PC的内容访问指令存储器，取出指令，并在该级结束时，把指令打入IR寄存器。

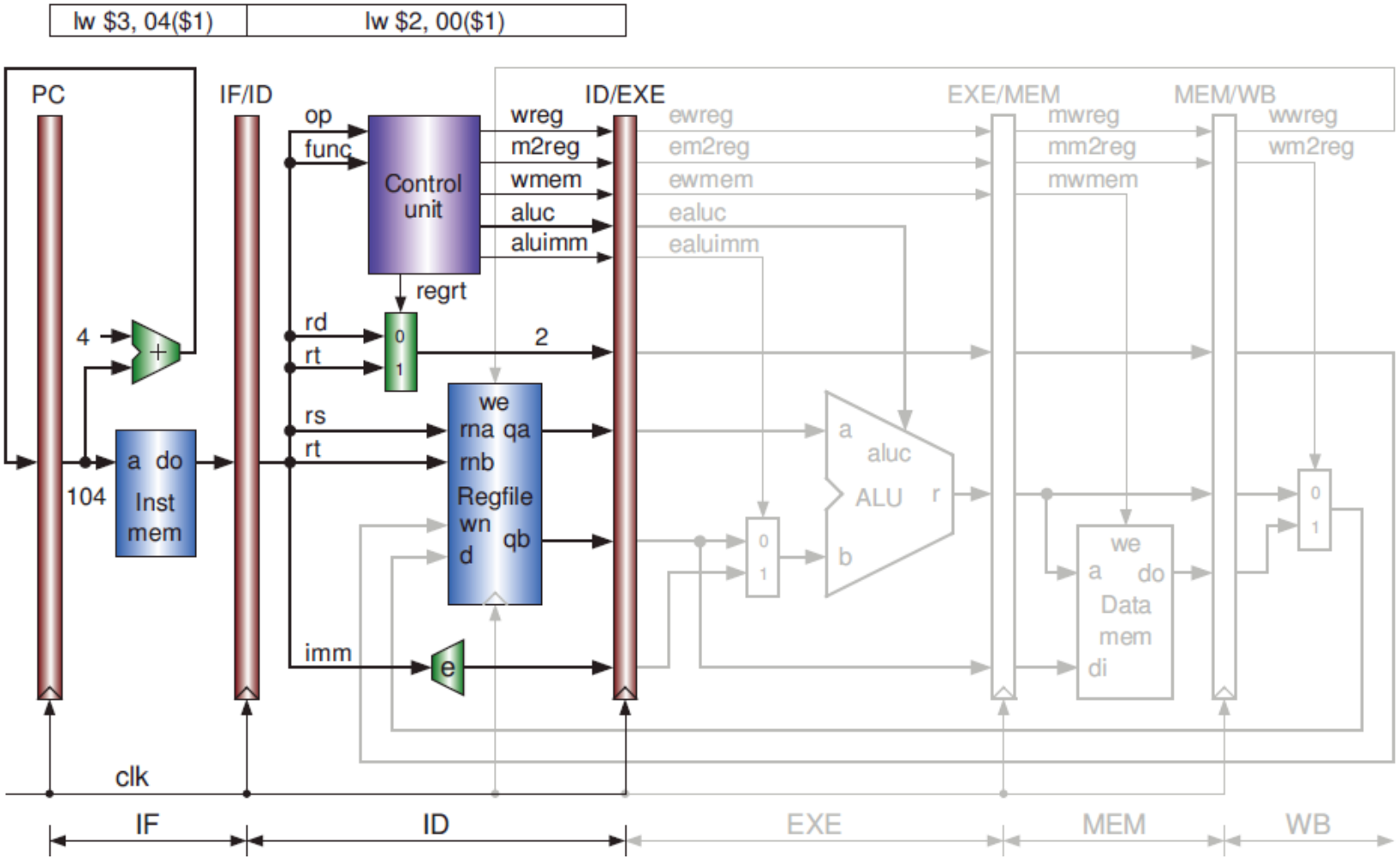


- 下一条指令的地址也在这一级计算出，并把它打入PC寄存器。新的PC的计算有两种情况：
- a、程序不发生转移时，新的PC地址由当前PC值加4得到，即指向下一条顺序的指令；
- b、发生转移时的情况较复杂，将在后面描述。

ID级：指令译码级

- 主要有两件事情要做：
- 从寄存器堆中读寄存器操作数和对指令中的立即数部分进行符号扩展。
- 产生控制信号





处在ID级的第1条指令由IR送出。根据指令中的rs和rt,两个32位的数据从寄存器堆读出。与此同时，指令中的16位的立即数也被扩展为32位。注意，lw指令并不使用从rt寄存器读出的数据。对该指令的译码由控制部件完成。控制信号的意义如下。

regrt为1时选择rt作为目的寄存器号，为0时选择rd;

aluimm为1时选择立即数送给ALU的b输入端，为0时选择寄存器数据

aluc是ALU的操作控制码

wmem为1时写存储器，为0时不写

m2reg为1时选择存储器数据，为0时选择ALU的计算结果

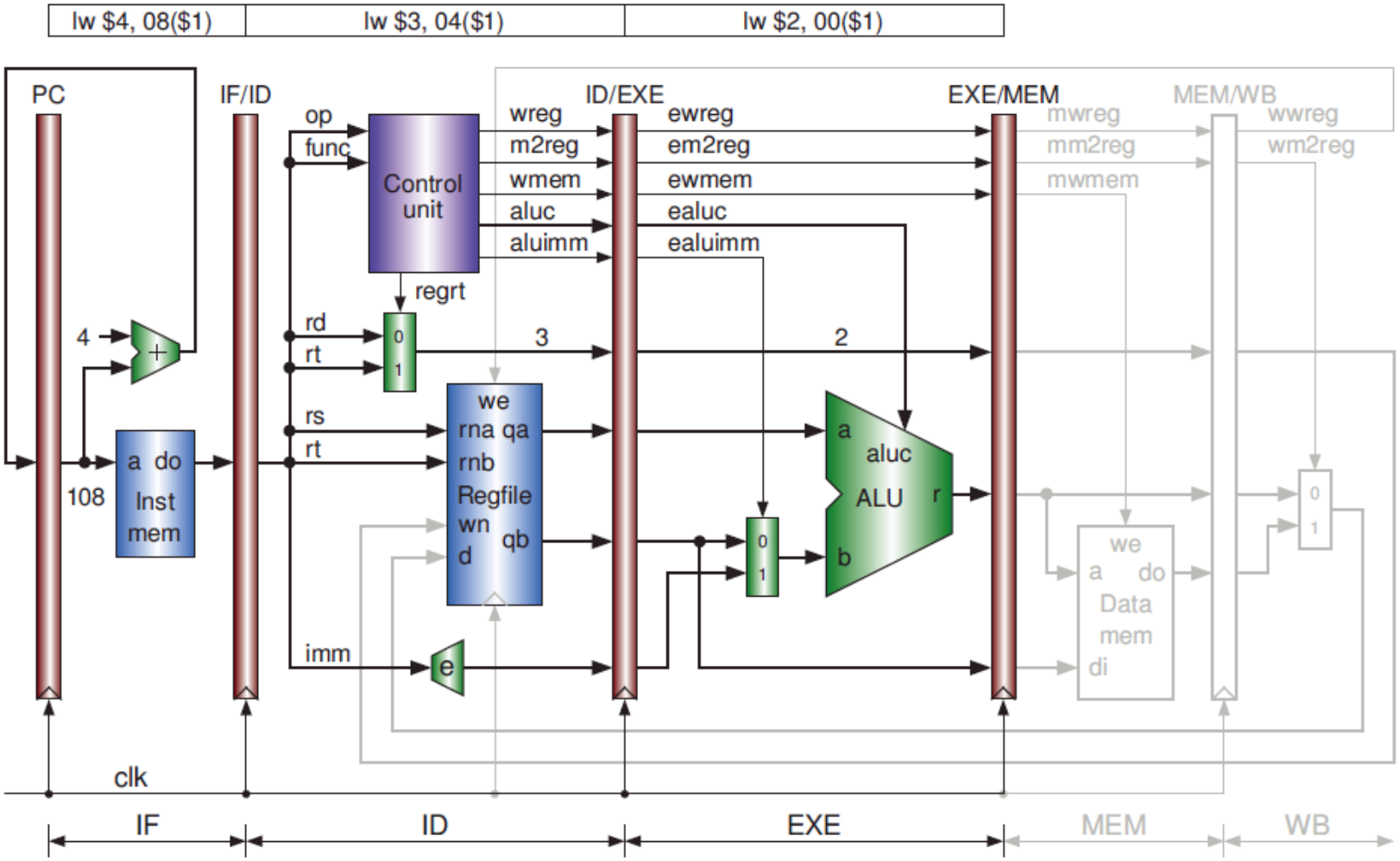
wreg为1时把结果写入寄存器堆，为0时不写。

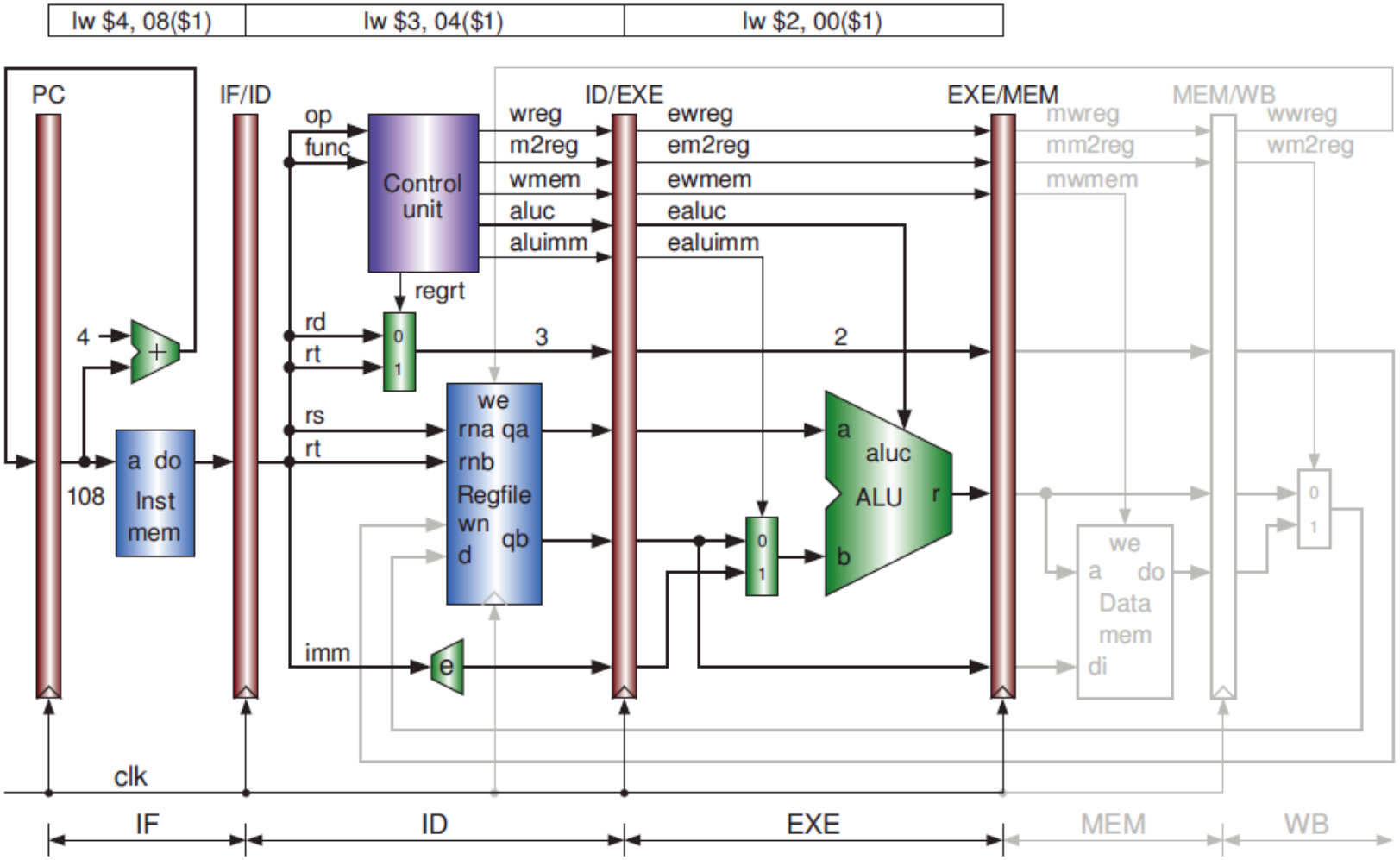
除了regrt只在本级使用外，其他控制信号要被写入流水线寄存器，以备后续周期使用。

因为当前指令为lw,所以regrt= 1,　aluimm= 1,　aluc　= X000,　wmem= 0,　m2reg= 1,　wreg= 1

EXE级：执行级

- ALU运算类指令将在本级计算结果。
- ALU的两个操作数，一个来自于寄存器rs1，另一个或是rs2操作数（B中内容），或是立即数。





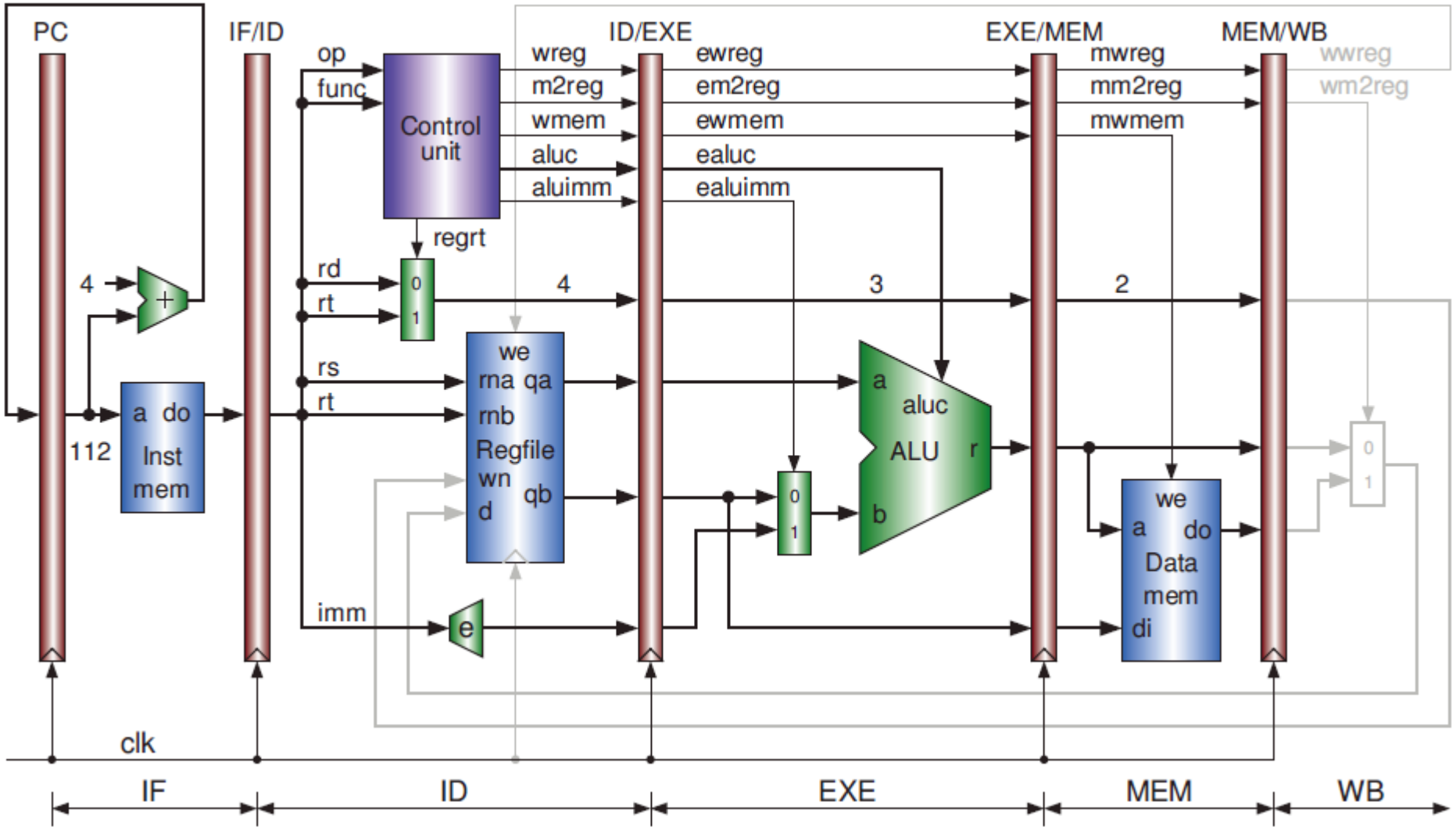
在 EXE 级，ALU 为 lw指令计算存储器地址；该级使用两个控制信号：ealuimm 选择立即数 、ealuc 告诉ALU 做加法。

在EXE 级的控制信号的名称前面都加了一个字母 e, 表示是 EXE 级的信号，以便与 ID 级的相应信号区别开来。

MEM级：存储器访问级

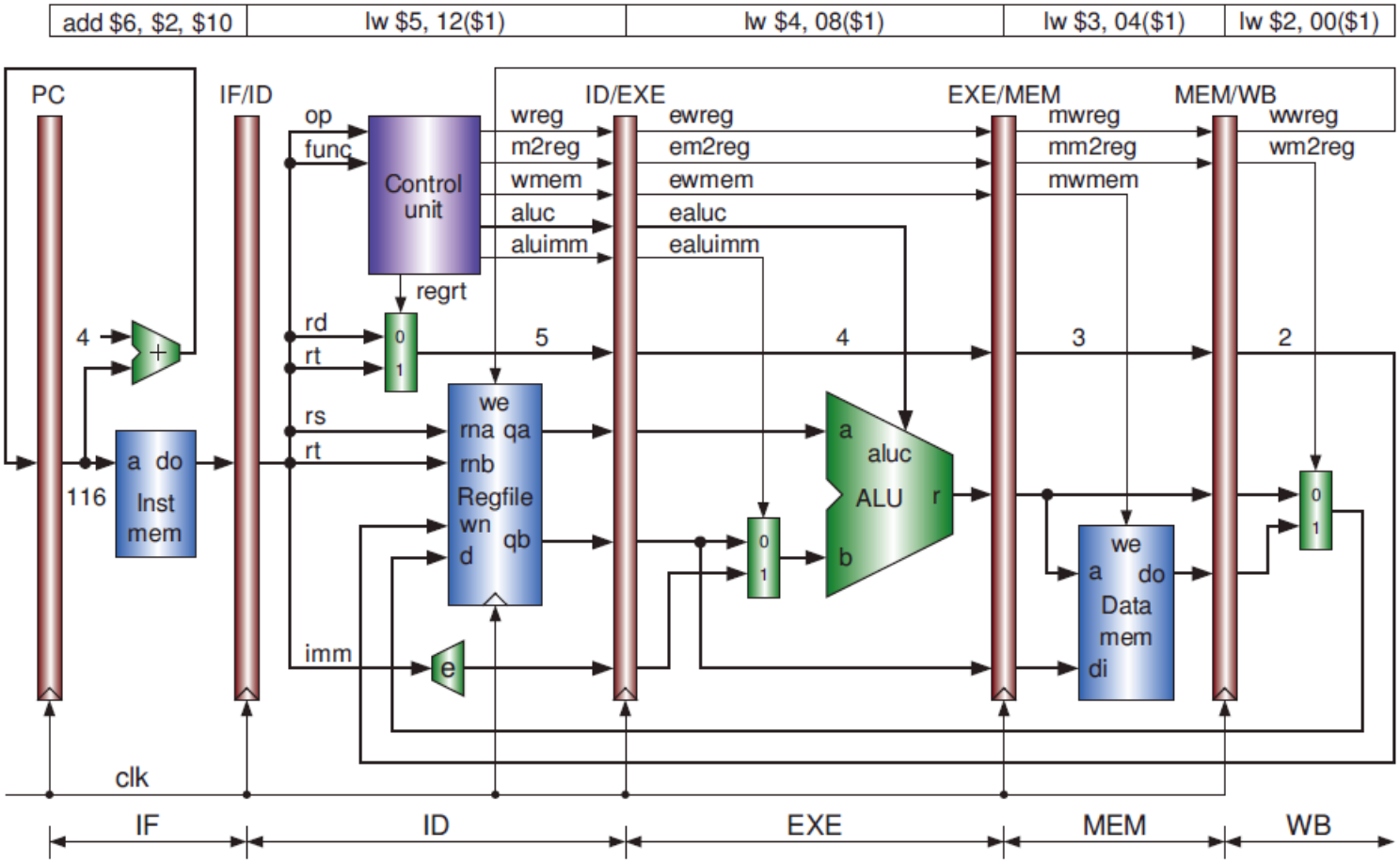
- 专为load / store指令而设。均使用Alu的输出作为访问存储器的地址

lw $5, 12($1)	lw $4, 08($1)	lw $3, 04($1)	lw $2, 00($1)



WB级：结果写回级

- 把结果写回到寄存器堆。目的寄存器号由从ID级一直传递过来。
- 写入的数据来源有两个：load指令读出内容，即存储器数据；另一个是ALU指令的计算结果。



C.3 流水线处理机及其设计

- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

C.3.3 流水线处理机的控制

- 为非流水线多周期处理机设计控制部件时，有两种方法可供我们选择：
- 一种是有限状态机的硬布线方法；
- 另一种是微程序控制方法。

- 流水线处理机也是多周期处理机，但由于多条指令的重叠执行，则很难描述某个时钟周期处理机到底处在何种状态，因为有太多的不同指令间的组合情况。微程序控制方法也会遇到类似的问题。

- 流水线模型机的控制部件设计方法，先考虑用软件方法解决冒险：
- 通过一个无相关的程序例子，分析出运算类指令与Load/Store指令在流水线各级需要的控制信号；
- 分析转移类指令需要的控制信号；
- 推导出各控制信号的逻辑表达式；
- 用组合逻辑线路实现控制信号的逻辑表达式。

C.3 流水线处理机及其设计

- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

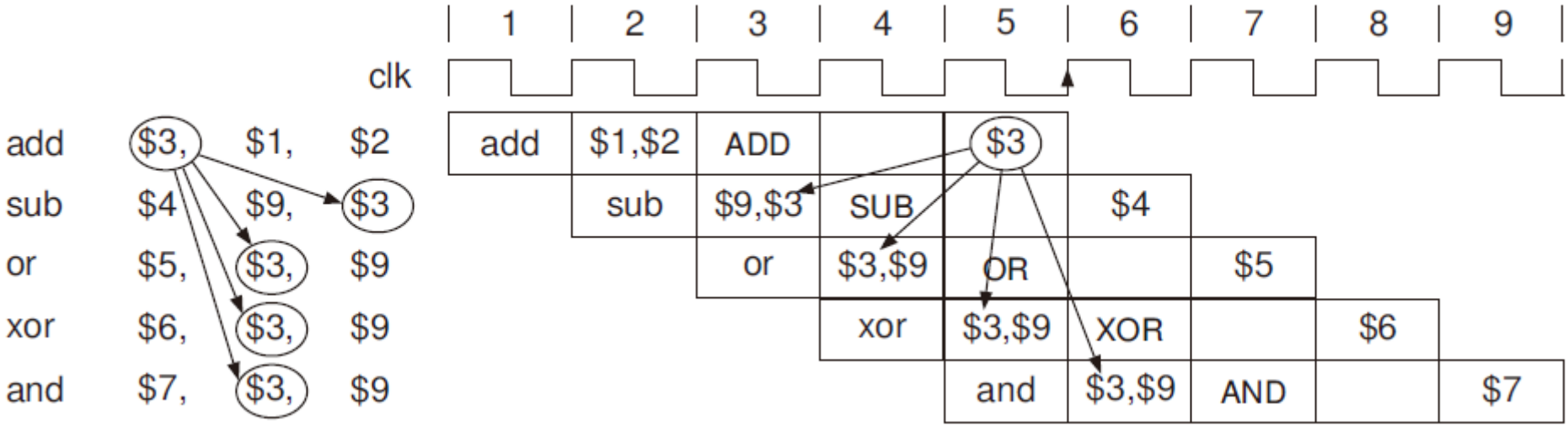
C.3.4 结构相关及解决方法

- 结构相关（冒险）：硬件资源冲突
- 解决：可以增加硬件资源；
- 或者功能部件完全流水；
- 否则，只有停顿流水线

C.3 流水线处理机及其设计

- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

数据相关


1
2
3
4
5
6
7
8
9
clk
add $3, $1, $2
sub $4 $9, $3
or $5, $3, $9
xor $6, $3, $9
and $7, $3, $9
add
$1,$2
ADD
$3
sub
$9,$3
SUB
$4
or
$3,$9
OR
$5
xor
$3,$9
XOR
$6
and
$3,$9
AND
$7

- 解决相关最简单的方式就是停顿流水线：
  - 软件方式：*由编译器在相关指令之间插入nop*
  - 硬件方式：*由逻辑电路在ID级检测相关；*
- *有相关则禁止改变PC和IR、封锁写信号暂停流水线一个或多个时钟周期。*
- 但是，暂停流水线两个或一个周期造成了处理机性能的损失。
- 硬件方法—内部前推则能完全避免ALU指令相关而造成的流水线停顿。

Load相关

- 由编译器处理。当一条指令与它上面的load指令数据相关时，在它们中间插入两条nop指令，然后优化，用两条不相关的指令替换两条nop指令。替换一条是一条，替换不掉就保留nop指令。
- 由硬件负责检测与Load指令的相关性。采用暂停流水线一个周期的方法消除第一个“气泡”。第二个“气泡”用内部前推技术加以消除。

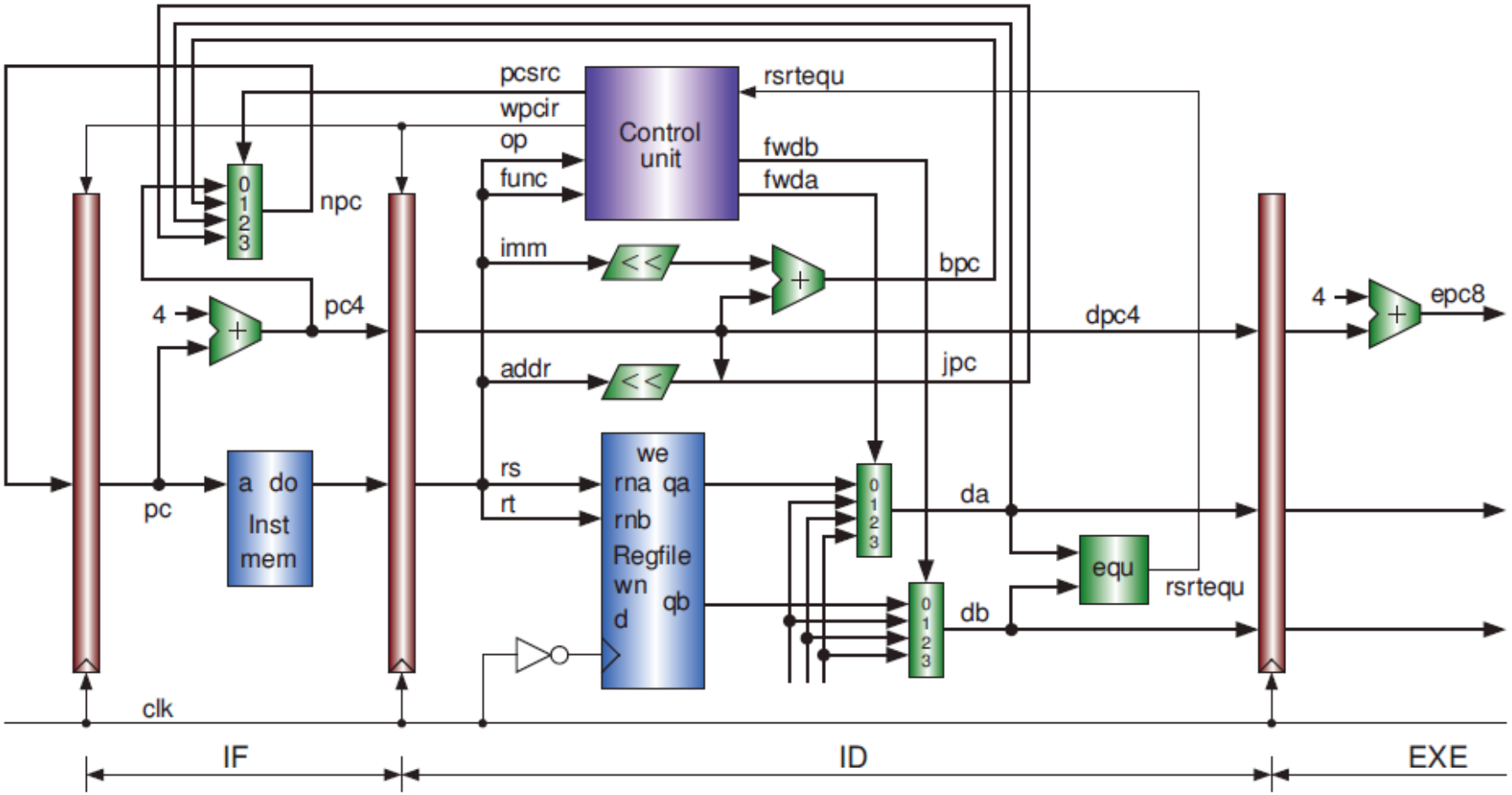
C.3 流水线处理机及其设计

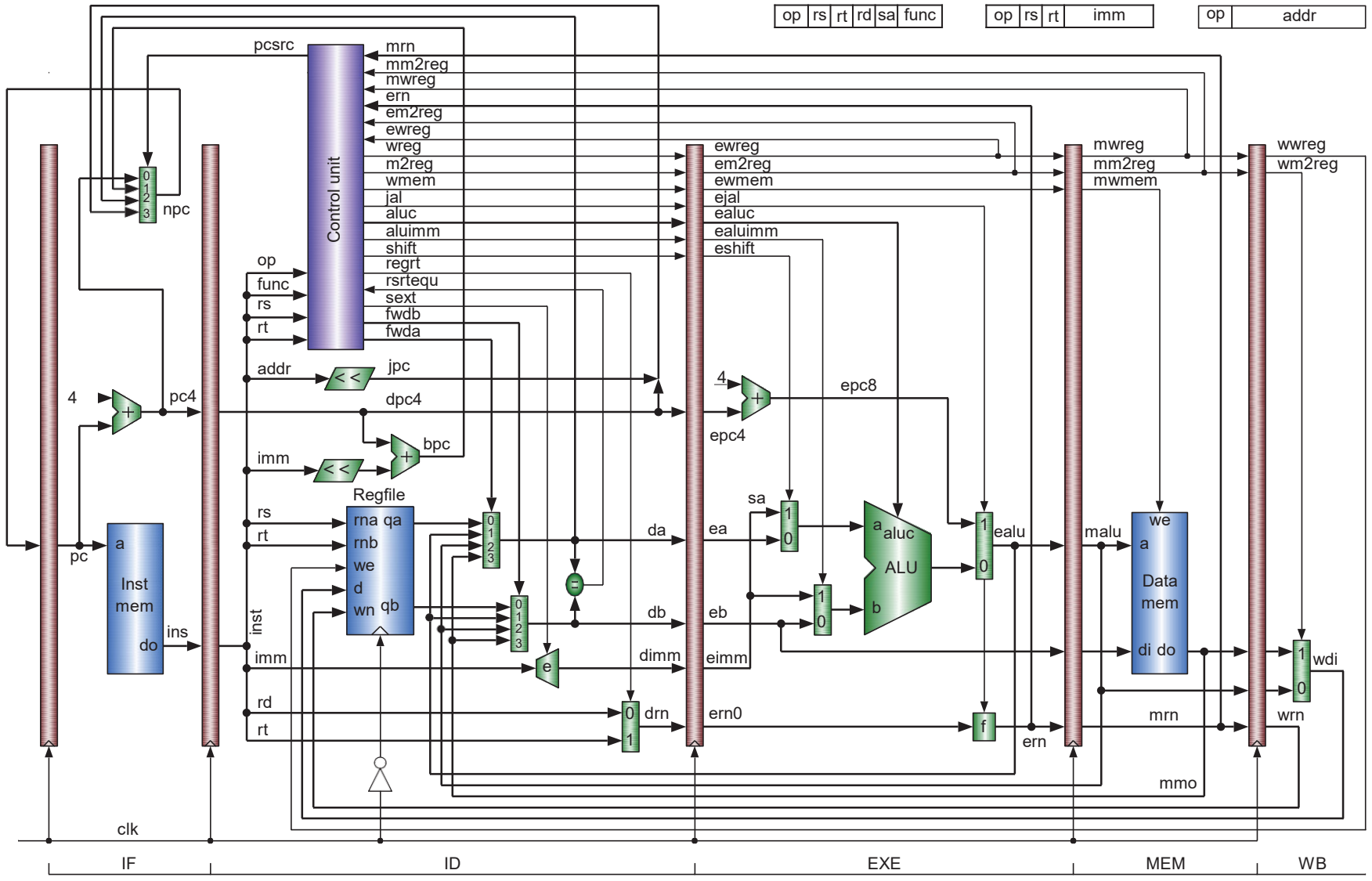
- C.3.1 流水线处理机的指令系统
- C.3.2 流水线处理机的数据路径
- C.3.3 流水线处理机的控制
- C.3.4 结构相关及解决方法
- C.3.5 数据相关及解决办法
- C.3.6 转移相关及解决方法

转移相关

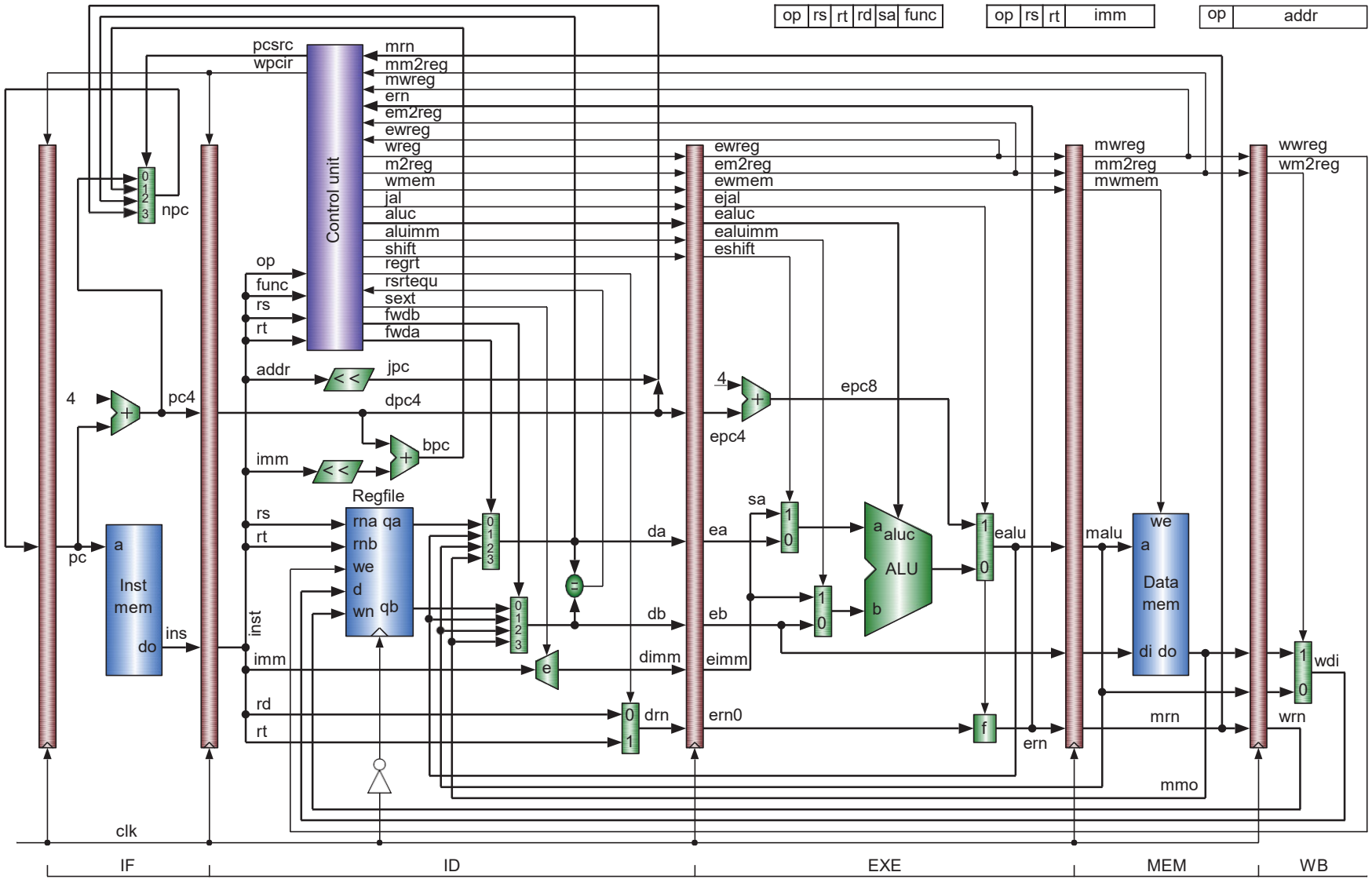
- 一般比较复杂的处理机往往需要3～5个周期才能完成转移指令的执行，受影响的指令至少2条。
- 在流水线模型机方案中，已经把转移指令所需的周期数减至最少，总共只需2个时钟周期，只有一条后续指令受到影响。处理转移相关问题的3种技术：
- （1）暂停流水线
- （2）假定转移不发生（猜测）
- （3）延迟转移

Implementation of delayed branch with one delay slot





pipelined with forwarding CPU



Detailed circuit of the pipelined CPU

模型机相关处理总结

- 流水线模型机中的相关处理：
- 结构相关：停顿流水线（插nop），增加硬件资源，功能部件完全流水
- 算术指令数据相关：插nop（软件），
- 暂停流水线（硬件），数据前推（硬件优化）
- LOAD数据相关：插nop，暂停与前推结合（硬件优化）
- 转移相关：插nop，条件转移暂停（硬件），假定转移不发生（硬件优化），延迟转移（优化、软硬件）

Discussion

- COD 5e Exercise
- 4.5
- 4.8

COD 5e Exercise 4.5

Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

假定没有流水线阻塞，各种类型的指令所占的比例如下:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

- **In what fraction of all cycles is the data memory used?**
- **数据存储器的平均使用率(占总周期数的百分比)是多少?**
- **In what fraction of all cycles is the input of the sign-extend circuit needed?**
- **符号扩展电路的输入平均使用率(占总周期数的百分比)是多少?**

4.5.1

Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

假定没有流水线阻塞，各种类型的指令所占的比例如下:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

- **In what fraction of all cycles is the data memory used?**
- **数据存储器的平均使用率(占总周期数的百分比)是多少?**

4.5.1

Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

假定没有流水线阻塞，各种类型的指令所占的比例如下:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

- **In what fraction of all cycles is the data memory used?**
- **The data memory is used by LW and SW instructions, so the answer is: 25% + 10% = 35%**

4.5.2

For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

假定没有流水线阻塞，各种类型的指令所占的比例如下:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

- **In what fraction of all cycles is the input of the sign-extend circuit needed?**
- **符号扩展电路的输入平均使用率(占总周期数的百分比)是多少?**

4.5.2

For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

- **In what fraction of all cycles is the input of the sign-extend circuit needed?**
- **The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for ADD and NOT instructions. The input of the sign-extend circuit is needed for ADDI (to provide the immediate ALU operand), BEQ (to provide the PC-relative offset), and LW and SW (to provide the offset used in addressing memory) so the answer is:**
- **Is needed : 20% + 25% + 25% + 10% = 80%**

4.8

Problems in this exercise assume that individual stages of the datapath have the following latencies:
下面给出了数据通路中不同阶段的延迟情况:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

Also, assume that instructions executed by the processor are broken down as follows:
另外， 假定处理器执行各种指令的比率如下面所示:

alu	beq	lw	sw
45%	20%	20%	15%

4.8.1

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **What is the clock cycle time in a pipelined and single-cycle processor?**
- **流水线处理器与单周期处理器的时钟周期分别是多少？**

4.8.1

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **What is the clock cycle time in a pipelined and single-cycle processor?**

Pipelined	Single-cycle
Maximum Latency in 5 stages:350 ps	Sum of all stages：250+350+150+300+200=1250 ps

4.8.2

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **What is the total latency of an LW instruction in a pipelined and single-cycle processor?**
- **lw指令在流水线处理器和单周期处理器中的总延迟分别是多少?**

4.8.2

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **What is the total latency of an LW instruction in a pipelined and non-pipelined processor?**

Pipelined	Single-cycle
5 Pipelined Clock Cycles 5*350=1750 ps	One Single-cycle Clock Cycle:1250 ps

4.8.3

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?**
- **如果可以将原流水线数据通路的某一级划分为两级， 每级的延迟是原级的一半， 那么你会选择哪一级进行划分?划分后处理器的时钟周期为多少?**

4.8.3

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?**

IF	ID1	ID2	EX	MEM	WB
250ps	<=300ps	<=300ps	150ps	300ps	200ps

Stage to split	New clock cycle time
ID	Maximum Latency in 6 stages:300 ps

4.8.4

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **Assuming there are no stalls or hazards, what is the utilization of the data memory?**
- **假设没有阻塞和冒险，数据存储器的利用率是多少(占总周期数的百分比)?**

4.8.4

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **Assuming there are no stalls or hazards, what is the utilization of the data memory?**

lw + sw => 35%

4.8.5

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?**
- **假设没有阻塞和冒险，寄存器堆模块写端口的利用率是多少(占总周期数的百分比)?**

4.8.5

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

alu	beq	lw	sw
45%	20%	20%	15%

- **Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?**

lw + alu => 65%

Classroom Test

- **Exercise C.7**
- In this problem, we will explore how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data and control hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every 5 instructions, whereas the 12-stage pipeline experiences 3 stalls every 8 instructions. In addition, branches constitute 20% of the instructions, and the misprediction rate for both machines is 5%. （Keep three digits after the point)
- 在这个问题中，我们将研究流水线的加深如何以两种不同方式来影响性能：加快时钟周期，因为数据与控制冒险而延长停顿。假定原机器是一个5级流水线，其时钟周期为1 ns。第二种机器为12级流水线，时钟周期为0.6 ns。由于数据冒险，5 级流水线每5 条指令经历1次停顿，而12 级流水线每8条指令经历3 次停顿。此外，分支占全部指令的20% ，两台机器的错误预测率都是5%。

- (a) What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?
- 仅考虑数据冒险， 12 级流水线相对于5 级流水线的加速比为多少?
- (b) If the branch mispredict penalty for the first machine is 2 cycles but the second machine is 5 cycles, what are the CPIs of each, taking into account the stalls due to branch mispredictions?
- 如果第一台机器的分支错误预测代价为2 个周期.而第二台机器为5 个周期，则每种机器的CPI 为多少?由于分支错误预测而导致的停顿考虑在内。

Classroom Test(a)

- **Exercise C.7**
- In this problem, we will explore how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data and control hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every 5 instructions, whereas the 12-stage pipeline experiences 3 stalls every 8 instructions. In addition, branches constitute 20% of the instructions, and the misprediction rate for both machines is 5%. （Keep three digits after the point)
- 在这个问题中，我们将研究流水线的加深如何以两种不同方式来影响性能：加快时钟周期，因为数据与控制冒险而延长停顿。假定原机器是一个5级流水线，其时钟周期为1 ns。第二种机器为12级流水线，时钟周期为0.6 ns。由于数据冒险，5 级流水线每5 条指令经历1次停顿，而12 级流水线每8条指令经历3 次停顿。此外，分支占全部指令的20% ，两台机器的错误预测率都是5%。

- (a) What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?
- 仅考虑数据冒险， 12 级流水线相对于5 级流水线的加速比为多少?
- Execution Time = I * CPI * Cycle Time
- Speedup = (I * 6 / 5 * 1) / (I * 11/8 * 0.6) = 1.455

Classroom Test(b)

- **Exercise C.7**
- In this problem, we will explore how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data and control hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every 5 instructions, whereas the 12-stage pipeline experiences 3 stalls every 8 instructions. In addition, branches constitute 20% of the instructions, and the misprediction rate for both machines is 5%. （Keep three digits after the point)
- 在这个问题中，我们将研究流水线的加深如何以两种不同方式来影响性能：加快时钟周期，因为数据与控制冒险而延长停顿。假定原机器是一个5级流水线，其时钟周期为1 ns。第二种机器为12级流水线，时钟周期为0.6 ns。由于数据冒险，5 级流水线每5 条指令经历1次停顿，而12 级流水线每8条指令经历3 次停顿。此外，分支占全部指令的20% ，两台机器的错误预测率都是5%。

- (b) If the branch mispredict penalty for the first machine is 2 cycles but the second machine is 5 cycles, what are the CPIs of each, taking into account the stalls due to branch mispredictions?
- 如果第一台机器的分支错误预测代价为2 个周期.而第二台机器为5 个周期，则每种机器的CPI 为多少?由于分支错误预测而导致的停顿考虑在内。
- **CPI_{5_stage}=6/5 + 0.2 * 0.05 * 2 = 1.22**
- **CPI_{12_stage}=11/8 + 0.2 * 0.05 * 5 = 1.425**
- **Speedup =(1 * 1.22 * 1) / (1 * 1.425 * 0.6) = 1.4269 = 1.427**

Discussion

- Exercise C.4
- A reduced hardware implementation of the classic five-stage RISC pipeline might use the EX stage hardware to perform a branch instruction comparison and then not actually deliver the branch target PC to the IF stage until the clock cycle in which the branch instruction reaches the MEM stage. Control hazard stalls can be reduced by resolving branch instructions in ID, but improving performance in one respect may reduce performance in other circumstances. Write a small snippet of code in which calculating the branch in the ID stage causes a data hazard, even with data forwarding.
- 经典5级RISC流水线的精简硬件实现可能使用EX级硬件来执行分支指令对比。分支指令会在某一时钟周期到达MEM 级， 在此时钟周期之前， 不会将分支目标PC实际提交给IF级。通过求解ID中的分支指令可以缩减控制冒险停顿，但某一方面的性能提升可能会降低其他情况下的性能。写一小段代码， 在此代码中计算ID 级的分支时会导致数据冒险，甚至在拥有数据前递时也是如此。
- （假设用ALU、DataMEM每级的输入做前推，没有用他们的输出做前推）

Exercise C.4

- 经典5级RISC流水线的精简硬件实现可能使用EX级硬件来执行分支指令对比。分支指令会在某一时钟周期到达MEM 级， 在此时钟周期之前， 不会将分支目标PC实际提交给IF级。通过求解ID中的分支指令可以缩减控制冒险停顿，但某一方面的性能提升可能会降低其他情况下的性能。写一小段代码， 在此代码中计算ID 级的分支时会导致数据冒险，甚至在拥有数据前递时也是如此。
- （假设用ALU、DataMEM每级的输入做前推，没有用他们的输出做前推）

Answer: Calculating the branch in the ID stage does not help if the branch is the one receiving data from the previous instruction. For example, loops which exit depending on a memory value have this property, especially if the memory is being accessed through a linked list rather than an array. There are many correct answers to this.

```
LOOP:   LW R1, 4(R2)        # R1 = r2->value
        ADD R3, R3, R1      # sum = sum + r2->value
        LW R2, 0(R2)        # r2 = r2->next
        BNE R2, R0, LOOP    # while (r2 != null) keep looping
```

The second LW and ADD could be reordered to reduce stalls, but there would still be a stall between the second LW and BNE.