DeepL
订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

第 3 章 ILP 开发的作业 3

浙江大学计算机学院

1. 您的任务是设计一种新的处理器微体系结构，您正在尝试
确定如何最好地分配硬件资源。你应该应用第 3 章中学到的哪些硬件和软件技术？您有一份功能单元和内存延迟列表，以及一些有代表性的代码。你的老板对新设计的性能要求含糊其辞，但根据经验，在其他条件相同的情况下，速度越快越好。从最基本的开始。下图提供了指令序列和延迟列表。
提示：假设取指、译码和发射共用同一个周期，WB 需要一个周期。
要注意，题目假设寄存器是上升沿写入。

Latencies beyond single cycle	
Memory LD	+3
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
fadd.d	+2
fmul.d	+4
fdiv.d	+10

```
Loop:   fld     f2,0(Rx)
I0:     fmul.d  f2,f0,f2
I1:     fdiv.d  f8,f2,f0
I2:     fld     f4,0(Ry)
I3:     fadd.d  f4,f0,f4
I4:     fadd.d  f10,f8,f2
I5:     fsd     f4,0(Ry)
I6:     addi    Rx,Rx,8
I7:     addi    Ry,Ry,8
I8:     sub     x20,x4,Rx
I9:     bnz     x20,Loop
```

a. [10] 如果在上一条指令执行完毕之前，不能执行任何新指令，那么图中代码序列的 基准性能（以周期为单位，每次循环迭代）是多少？暂且假定执行不会因缺少下一条指令而停滞，但每循环只能发出一条指令。假设执行了分支指令，并且存在一个周期的分支延迟槽。
提示：本题没过程不给分。本题需要写出答案，并填写下面的表格。

IF/IS	FU	WB

```
Loop:   fld      f2,0(Rx)
I0:     fmul.d   f2,f0,f2
I1:     fdiv.d   f8,f2,f0
I2:     fld      f4,0(Ry)
I3:     fadd.d   f4,f0,f4
I4:     fadd.d   f10,f8,f2
I5:     fsd      f4,0(Ry)
I6:     addi     Rx,Rx,8
I7:     addi     Ry,Ry,8
I8:     sub      x20,x4,Rx
I9:     bnz      x20,Loop
```

b. [10] 想想延迟数字的真正含义--它们表示特定函数产生输出所需的周期数。如果整个流水线在每个功能单元的延迟周期内停滞，那么至少可以保证任何一对背靠背的指令（"生产者 "后接 "消费者"）都能正确执行。但并非所有指令对都存在生产者/消费者关系。有时，相邻的两条指令互不相关。如果流水线能检测到真正的数据依赖关系，并只在这些数据依赖关系上停滞，而不是因为某个功能单元繁忙就盲目停滞所有指令，那么图中代码序列中的循环体需要多少个周期？ 提示：一条延迟为 +2 的指令需要两个 <stall> 周期才能插入代码序列）。可以这样想：一个周期指令的延迟为 1+0，这意味着没有额外的等待状态。因此，延迟 1+1 意味着一个停滞周期；延迟 1+N 意味着 N 个额外的停滞周期。

提示：假设 core 是单发射的，每种 FU unit 有无限个。本题没过程不给分。

c. [15] 考虑多问题设计。假设有两条执行流水线，每条流水线每个周期能开始执行一条指令，前端有足够的取指/解码带宽，因此不会中断执行。假设结果可以立即从一个执行单元转发到另一个执行单元，或转发到自身。进一步假设，执行流水线停滞的唯一原因是观察到真正的数据依赖性。现在循环需要多少个周期？

提示：本小问使用类似 tomasulo 的算法，不需要拿到操作数就可以发射。

d. [20] 重新排列指令，以提高图中代码的性能。假设

练习 c 中的双管机器，并且已经成功解决了无序完成问题。现在只需注意观察真正的数据依赖性和功能单元延迟。重新排序的代码需要多少周期?

提示：本题没过程不给分。

e. [10] 在练习 d 中重新排序后的代码中，算上两个管道，有多少个周期被浪费（没有启动新的操作）？

f. [10] 循环展开是一种标准的编译器技术，用于在代码中发现更多的并行性，以尽量减少失去的性能机会。在练习 d 中重新排序的代码中手动展开两次循环迭代。

提示：展开时，展开的那个循环中的指令请用不同颜色标出，新指令使用的寄存器 f0->f10, f1->f11...给出指令的时序和耗时。本题没过程不给分。

g. [10] 你的速度提高了多少？(在本练习中，只需将 N+1 次迭代的指令涂成绿色，以区别于第 N 次迭代的指令；如果实际上是展开循环，则必须重新分配寄存器，以防止迭代之间发生碰撞）。

h. [15] 计算机的大部分时间都花在循环上，因此多次循环迭代是投机性地寻找更多工作以保持 CPU 资源繁忙的好地方。不过，任何事情都不是一帆风顺的；编译器只会生成一份该循环的代码副本，因此即使多个迭代处理的是不同的数据，它们看起来也会使用相同的寄存器。为了避免多个迭代使用的寄存器发生冲突，我们需要重新命名它们的寄存器。图 3.48 显示了我们希望硬件重新命名的示例代码。编译器本可以简单地展开循环并使用不同的寄存器来避免冲突，但如果我们希望硬件展开循环，那么它也必须进行寄存器重命名。怎么做？假设硬件有一个临时寄存器池（称作 T 寄存器，假设有 64 个，从 T0 到 T63），可以替代编译器指定的寄存器。这个重命名硬件以 src（源）寄存器名称为索引，表中的值是最后一个目标寄存器的 T 寄存器。(将这些表中的值视为生产者，而 src 寄存器则是消费者；只要消费者能找到它，生产者把结果放在哪里并不重要）。请看图 3.48 中的代码序列。每次看到代码中的目的寄存器时，用下一个可用的 T 代替，从 T9 开始。然后相应地更新所有 src 寄存器，以保持真正的数据相关性。显示生成的代码。(提示：参见图 3.49）。

```
Loop:          fld             f2,0(Rx)
I0:            fmul.d          f5,f0,f2
I1:            fdiv.d          f8,f0,f2
I2:            fld             f4,0(Ry)
I3:            fadd.d          f6,f0,f4
I4:            fadd.d          f10,f8,f2
I5:            sd              f4,0(Ry)
```

Figure 3.48 Sample code for register renaming practice.

```
I0:          fld               T9,0(Rx)
I1:          fmul.d            T10,F0,T9
...
```

Figure 3.49 Expected output of register renaming.

2. [超长指令字（VLIW）的设计者在寄存器使用的架构规则方面需要做出一些基本选择。假设 VLIW 采用自降频执行流水线设计：一旦启动一个操作，其结果将在最多 L 个周期后出现在目标寄存器中（L 为操作的延迟时间）。寄存器永远都不够用，因此人们总是想最大限度地利用现有寄存器。如图 3.52 所示，加载操

```
Loop:   lw      x1,0(x2);     lw      x3,8(x2)
        <stall>
        <stall>
        addi    x10,x1,1;     addi    x11,x3,1
        sw      x1,0(x2);     sw      x3,8(x2)
        addi    x2,x2,8
        sub     x4,x3,x2
        bnz     x4,Loop
```

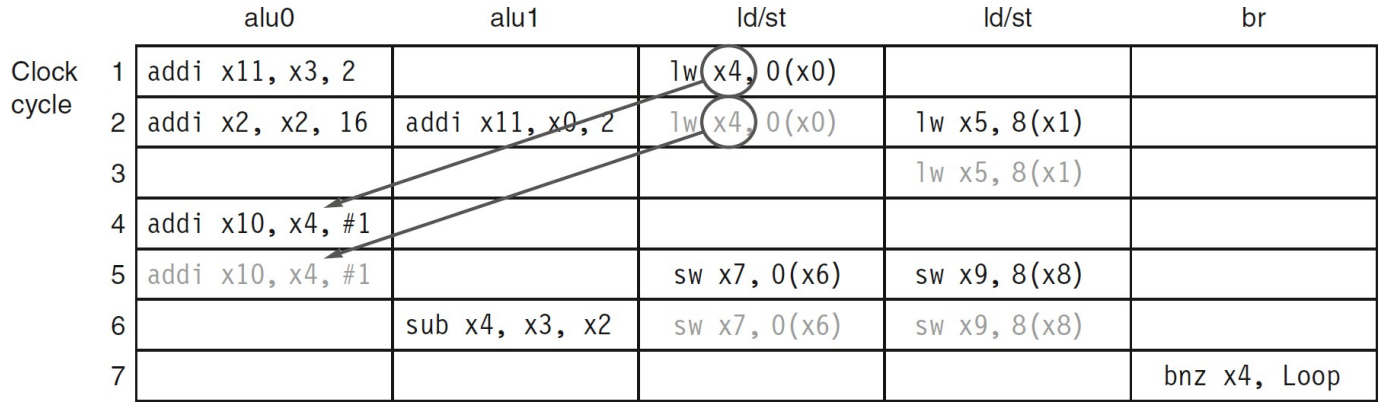
Figure 3.52 Sample VLIW code with two adds, two loads, and two stalls.

作的延迟为 1+2 个周期。

a. [10] 展开这个循环一次，显示结果。

b. [10] 观察下图。展示了在没有任何流水线中断或停滞的情况下，每周期能进行两次加载和两次加法运算的 VLIW 如何使用最少的寄存器数量。

c. [10] 请举例说明，在有自排水管道的情况下，有哪些事件可能会破坏这种管道连接并产生错误的结果。



补充：[20] vliw 是如何解决 raw 的？vliw 是如何解决 war 的？

3. [30] 假设采用五级单管道微体系结构（获取、解码、执行、内存、回写）和代码。除 LW 和 SW 为 1+2 周期外，所有操作均为一个周期

循环，而分支则是 1+1 循环。没有转发。本题需要给出过程，否则不给分。

循环：

lw x3,0(x0)

lw x1,0(x3)

addi x 1 , x1,1

sub x4,x3,x2

```
sw x1 , 0(x3)
bnz x4, Loop
```

a. [10] 显示循环迭代一次的每个时钟周期中每条指令的相位。每个循环迭代有多少个时钟周期是由于分支开销造成的?

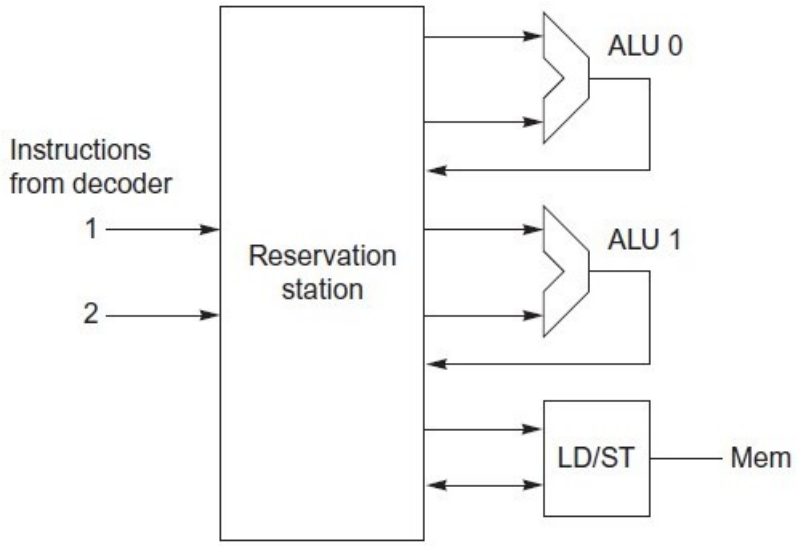
提示：画表，每行代表一条指令，每列代表一个周期，内容填写 FDEMW 五种阶段和 (失速)。

b. [10] 假设有一个静态分支预测器，能够在解码阶段识别后向分支。现在有多少个时钟周期浪费在分支开销上?

c. [10] 假设有一个动态分支预测器。一次正确预测会损失多少个周期?

4. [85] 让我们来考虑一下动态调度可能实现的目标。假设微体系结构如图所示。假设算术逻辑单元（ALU）可以执行所有算术运算（fmul.d、fdiv.d、fadd.d、addi、sub）和分支，而预订站（RS）每个周期最多只能向每个功能单元调度一次运算（向每个 ALU 调度一次运算，再向 fld/fsd 调度一次内存运算）。

提示：忽略取值、解码和写回，仅考虑 fu 阶段。



a. [15] 假设图（问题 1）中序列的所有指令 都存在于 RS 中，并且没有进行重命名。请指出代码中寄存器重命名可提高性能的指令。(提示：查找 "读后写 "和 "写后写 "危害。假设功能单元延迟与图（问题 1）中的相同）。

b. [20] 假设（a）部分代码的寄存器重命名版本在时钟周期 N 驻留在 RS 中，其延迟如图（问题 1）所示。请说明 RS 应如何逐个时钟不按顺序地分派这些指令，以获得该代码的最佳性能。(假定 RS 限制与（a）部分相同。还假设结果必须先写入 RS，然后才能使用--不能旁路）。代码序列需要多少个时钟周期?

c. [20] (b)部分让 RS 尝试对这些指令进行最佳调度。但实际上，RS 中通常并不存在感兴趣的整个指令序列。取而代之的是，各种事件会清除 RS 中的指令，当解码器输入新的代码序列时，RS 必须选择调度它所拥有的指令。假设 RS 是空的。在第 0 个周期，RS 中出现了该序列的前两条寄存器重命名指令。假设调度任何操作都需要一个时钟周期，并假设功能单元延迟与练习 3.2 相同。进一步假设前端（解码器/寄存器重命名器）将继续在每个时钟周期提供两条新指令。显示 RS 的逐周期发送顺序。现在这个代码序列需要多少个时钟周期?

d. [10] 如果你想改进(c)部分的结果，哪种方法最有帮助：(1) 另一个 ALU? (2) 另一个 LD/ST 单元? (3)

将 ALU 的结果完全旁路到后续操作？ 或者 (4) 将最长的延迟时间缩短一半？速度提升了多少？

e. [20] 现在，让我们考虑一下投机，即获取、解码和执行超越

一个或多个条件分支。我们这样做有两方面的动机：我们在第(c)部分中提出的调度计划有很多 nops，而且我们知道计算机的大部分时间都花在执行循环上（这意味着返回循环顶端的分支是非常可预测的）。循环告诉我们哪里有更多的工作要做；我们的稀疏调度计划表明，我们有机会比以前更早地完成一些工作。在第(d)部分中，你找到了通过循环的关键路径。想象一下，将该路径的第二个副本折叠到您在第(b)部分中得到的计划表上。完成两个循环的工作（假设所有指令都驻留在 RS 中）还需要多少个时钟周期？(假设所有功能单元都完全流水线化）。

附录：

1. (50 分）在 Scoreboard 上运行以下代码，其功能单元包括两个 FP 乘法器、一个 FP 除法器、一个可执行浮点加法和乘法运算的 FP 加法器，以及两个负责内存的整数单元

   存取、整数 ALU 运算和分支）。

(1) 在第一个 L.D 指令刚刚完成 WB 阶段时，将 **5th** 时钟周期结束时的记分板状态填入下表。记分牌状态包括指令状态、功能单元状态和 FP 寄存器状态。

Instruction producing result	Instruction consuming result	Latency (in clock cycle)
FP operation	FP operation	3
FP operation	FP store（S.D）	2
FP load（L.D）	FP opereation	1
FP load（L.D）	FP store（S.D）	0

```
L.D     f0,0(r1)        ; □ 0 ←□□□[r1+0]
L.D     f4,0(r2)        ; □ 4 ←□□□[r2+0]
mul.d,f0 □ 6 ←□ 0  * f4 add.df2
        □ 8 ←□ 0  + f2
s.df6   , 0(r3)         ; mem[r3 + 0] ← f6
s.df8       , 0(r4)     ; mem[r4 + 0] ← f8
```

说明	钟	作数(RO) 钟	钟	结 (WB) 钟
L.D F0, 0(R1)	1	2	3-4	5
L.D F4, 0(R2)				
MUL.D F6、F0、F4				
ADD.D F8,F0,F2				
S.D F6, 0(R3)				
S.D F8, 0(R4)				

功能单元	功能单元状态									
	繁忙	作品	Fi	Fj	Fk	Qj	Qk	Rj	Rk	A
整数1										
整数2										
FPMult1										
FPMult2										

FPDivider										
FPAdder										

	寄存器状态								
	F0	F2	F4	F6	F8	F10	F12	.....	F30
FU									

(2) 将第 **6** 个时钟周期结束时的指令状态填入下表，作为表格的第一行。

		作数(RO)	执行(EXE)	结 (WB)
L.D F0, 0(R1)	1	2	3-4	5
L.D F4, 0(R2)				
MUL.D F6、F0、F4				
ADD.D F8,F0,F2				
S.D F6, 0(R3)				
S.D F8, 0(R4)				

功能单元	功能单元状态									
	繁忙	作品	Fi	Fj	Fk	Qj	Qk	Rj	Rk	A
整数1										
整数2										
FPMult1										
FPMult2										
FPDivider										
FPAdder										

	寄存器状态								
	F0	F2	F4	F6	F8	F10	F12	.....	F30
FU									

2. （50 分）针对下列指令序列：

L.D F6, 34(R2)

L.D F2, 45(R3) mul.D

F0, F2, F4 sub.D F8,
F2, F6 div.D F10, F0,
F6

添加 d f6、f8、f2

(1) 填写 Tomasulo 算法在第一条指令完成并写入结果时使用的表格。假设内存访问单元执行指令需要两个时钟周期：一个用于计算地址，一个用于访问内存。

指令状态			
	问题	执行	写入结果
1			
2			
3			
4			
5			
6			

预订站							
姓名	繁忙	作品	Vj	Vk	Qj	Qk	A
载荷1							
载荷2							
Add1							
Add2							
Add3							
多种1							
多种2							

注册状态									
现场	F0	F2	F4	F6	F8	F10	F12	...	F30
气									

(2) 假设延迟时间如下：加载为 1 个时钟周期；加法为 2 个时钟周期；乘法为 10 个时钟周期；除法为 40 个时钟周期。当指令 MUL.D 即将写入结果时（在下一个周期写入结果），请填写表格。

指令状态			
	问题	执行完毕	写入结果
	时钟	时钟	时钟
1			
2			
3			
4			
5			
6			

预订站							
姓名	繁忙	作品	Vj	Vk	Qj	Qk	A
载荷1							
载荷2							
Add1							
Add2							
Add3							
多种1							
多种2							

注册状态									
现场	F0	F2	F4	F6	F8	F10	F12	...	F30
气									