Q1

> *You are tasked with designing a new processor microarchitecture and you are trying to determine how best to allocate your hardware resources. Which of the hardware and software techniques you learned in Chapter 3 should you apply? You have a list of latencies for the functional units and for memory, as well as some representative code. Your boss has been somewhat vague about the performance requirements of your new design, but you know from experience that, all else being equal, faster is usually better. Start with the basics. Figure below provides a sequence of instructions and list of latencies.*
>
> *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，也就是说，下图I0指令假设在第5周期将结果写回f2，则I1指令的执行阶段从第六周期开始*

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
```

Q1.a

> *What is the baseline performance (in cycles, per loop iteration) of the code sequence in the Figure if no new instruction's execution could be initiated until the previous instruction's execution had completed? Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued.*

> *（假设所有的指令都在同一个FU执行，但是不同指令有不同的latency，因此，每条指令都是等待前一条的FU执行完写回之后再去进入，比如Loop指令在第五周期进入WB，那I0的FU周期就从第六周期开始）*

INSTTRUCTION	IF/IS	FU	WB
Loop	0	1	1+1+3=5
I0:	1	6	6+1+4=11
I1:	2	12	12+1+10=23
I2:	3	24	24+1+3=28
I3:	4	29	29+1+2=32
I4:	5	33	33+1+2=36
I5:	6	37	37+1+1=39
I6:	7	40	40+1+0=41
I7:	8	42	42+1+0=43
I8:	9	44	44+1+0=45
I9:	10	46	46+1+1=48

Answer:48

Q1.b

> *List the dependency relation of previous code list,(for example, f2 in I1 depends on the outcome of I0) and consider if there are different and infinitely function unit, what's the table below will look like?*
>
> *(假设有无限个Function unit，重新填写上面那张表)*

这里仅列出对

- f2 in I0 depends on the outcome of Loop
- f2 in I1 depends on the outcome of I0
- f8 in I4 depends on the outcome of I1
- f4 in I3 depends on the outcome of I2
- Rx in I8 depends on the outcome of I6

INSTTRUCTION	IF/IS	FU	WB
Loop	0	1	1+1+3=5

INSTTRUCTION	IF/IS	FU	WB
I0:	1	6	6+1+4=11
I1:	2	12	12+1+10=23
I2:	3	4	4+1+3=8
I3:	4	9	9+1+2=12
I4:	5	24	24+1+2=27
I5:	6	13	13+1+1=15
I6:	7	8	8+1+0=9
I7:	8	9	9+1+0=10
I8:	9	10	10+1+0=11
I9:	10	12	12+1+1=14

Answer:27(if we consider the situation that some instructions run in WB state in the same time as legal)

Q1.c

> *suppose your machine can issue two instruction at one time, what will the table look like?*
>
> *(假设一次能发射两条指令，FU数量仍为无限个)*

INSTTRUCTION	IF/IS	FU	WB
Loop	0	1	1+1+3=5
I0:	0	6	6+1+4=11
I1:	1	12	12+1+10=23
I2:	1	2	2+1+3=6
I3:	2	7	7+1+2=10
I4:	2	24	24+1+2=27
I5:	3	11	11+1+1=13
I6:	3	4	4+1+0=5
I7:	4	12	12+1+0=13
I8:	4	6	6+1+0=7
I9:	5	8	8+1+1=10

Answer:27(if we consider the situation that some instructions run in WB state in the same time as legal)

Q2

> *介绍一下VLIW技术，谈一下他是怎么解决RAW和WAR冒险的，他的优缺点在哪*

> *参考：*
>
> *计算机指令集架构 – 超长指令字（VLIW） - 知乎 (zhihu.com)*
>
> *L23 (nju.edu.cn)*

VLIW(超长指令字)处理器使用多个独立功能单元，VLIW没有尝试向这些单元发射多条独立指令，而是将多个操作包装在一个非常长的指令中，或者要求发射包中的指令满足同样的的束条件。

解决RAW和WAR冒险：

- 做法1：完全由编译器通过代码调度和插入nop指令来消除所有冒险， 指令来消除所有冒险，无需硬件实现冒险检测和流水线阻塞
- 做法2：由编译器通过静态分支预测和代码调度来消除同时发射指令间内部依赖，由硬件检测数据冒险并进行流水线阻塞， 由硬件检测数据冒险并进行流水线阻塞，即保证打包指令内部不会出现冒险

优点：

- 简化了硬件电路，不需要动态调度硬件
- 不需要在VLIW指令中进行依赖项检查，因为编译器已经完成了检查，只需要简单的指令多发射硬件
- 并行化程度高，降低CPI

缺点：

- 对编译器性能要求较高，编译性能直接决定了使用该技术时的冲突解决性能优劣
- 当执行宽度(N)、指令延迟、功能单元改变时需要重新编译（超标量架构不需要重新编译）
- Binary code compatibility二进制代码兼容性差
- 代码长度增加，例如循环展开带来的代码长度增加
- function slots的使用率较低

- Limitations of lockstep operation
- 任何功能单元中的停滞都可能导致整个处理器停滞

Q3

> *Assume a five-stage single-pipeline microarchitecture (fetch, decode, execute, memory, write-back) and the code. All ops are one cycle except LW and SW, which are 1+2 cycles, and branches, which are 1+1 cycles. There is no forwarding.*
>
> *本题需要给出过程，否则不给分。*

Loop:

lw x3,0(x0)

lw x1,0(x3)

addi x1,x1,1

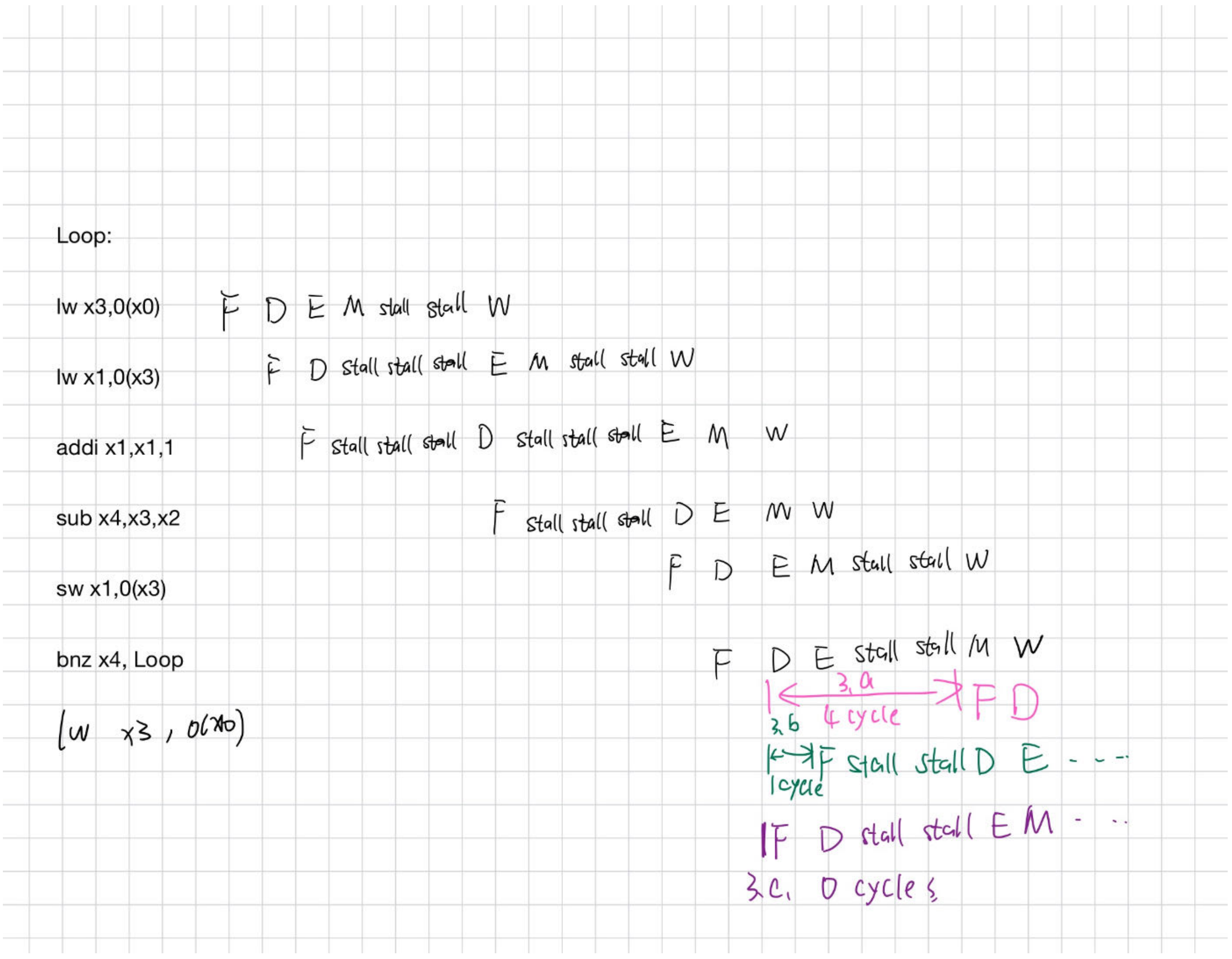
sub x4,x3,x2

sw x1,0(x3)

bnz x4, Loop

Q3.a

> *Show the phases of each instruction per clock cycle for one iteration of the loop. How many clock cycles per loop iteration are lost to branch overhead?*
>
> *提示：画表，每行代表一条指令，每列代表一个cycle，内容填写FDEMW五种阶段和- (stall)，这里WB和EX可以同时发生，比如假设第一条指令在第七周期WB，则依赖于其结果的第二条指令第7周期直接进入execute阶段。*
>
> *在表中要包括代码进入新一轮循环的第一条指令，其应该在跳转指令EX阶段结束后开始执行。*
>
> *cycles per loop iteration are lost to branch overhead，是指跳转指令的fetch周期和第二轮循环第一条指令的fetch周期之间的差值减1，即若跳转指令fetch后第二轮循环第一条指令在下一周期立马fetch，视为0 overhead*



The **pink** part in the picture is the `fetch` stage, the clock cycles per loop iteration are lost to branch overhead is **2**.`Fetch` occurs in the end of stall.

Q3.b

> *Assume a static branch predictor, capable of recognizing a backward branch in the Decode stage. Now how many clock cycles are wasted on branch overhead?*
>
> *假设一个静态分支预测器，能够在解码阶段识别反向分支。现在，在分支开销上浪费了多少时钟周期？*

The **green** part in the picture is the `fetch` stage with a static predictor, the clock cycles per loop iteration are lost to branch overhead is **1**.`Fetch` occurs in the end of `Decode` of last instruction since of prediction placing in `Decode` .

Q3.c

> *Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?*
>
> *假设一个动态分支预测器。一个正确的预测会损失多少个周期?*

The **purple** part in the picture is the `fetch` stage with a dynamic predictor, the clock cycles per loop iteration are lost to branch overhead is **0**.

Dynamic predictor remembers the branch would occur,so there is no lost cycles waited for fetch .