homework 1

1. 将下面的汇编指令/二进制编码转换为对应的二进制编码/汇编指令

1. sll x4, x3, x3

7bit	5bit	5bit	3bit	5bit	7bit
0000000	rs2	rs1	001	rd	opcode
0000000	00011	00011	001	00100	0110011

Answer: `0x319233`

2. jalr x2, 0x100(x3)

> jalr rd, rs1, imm

12bit	5bit	3bit	5bit	7bit
imm	rs1	000	rd	opcode
0001_0000_0000	00011	000	00100	1100111

Answer: `0x10018267`

3. lui x4,0x12345

20bit	5bit	7bit
imm	rd	opcode
0001_0010_0011_0100_0101	00100	0110111

Answer: `0x12345237`

4. 0x00f0c093

12bit	5bit	3bit	5bit	7bit
imm	rs1	func	rd	opcode
0000_0000_1111	00001	100	00001	0010011

Answer:

```
XORI x1, x1, 0x00f
```

5. 0x0020e463

7bit	5bit	5bit	3bit	5bit	7bit
imm[12|10:5]	rs2	rs1	func	imm[4:1|11]	opcode
0|00_0000	00010	00001	110	0100|0	1100011

Answer:

```
BLTU x1,x2,0x008
```

6. 0xfe20bc23

7bit	5bit	5bit	3bit	5bit	7bit
imm[11:5]	rs2	rs1	func	imm[4:0]	opcode
1111111	00010	00001	011	11000	0100011

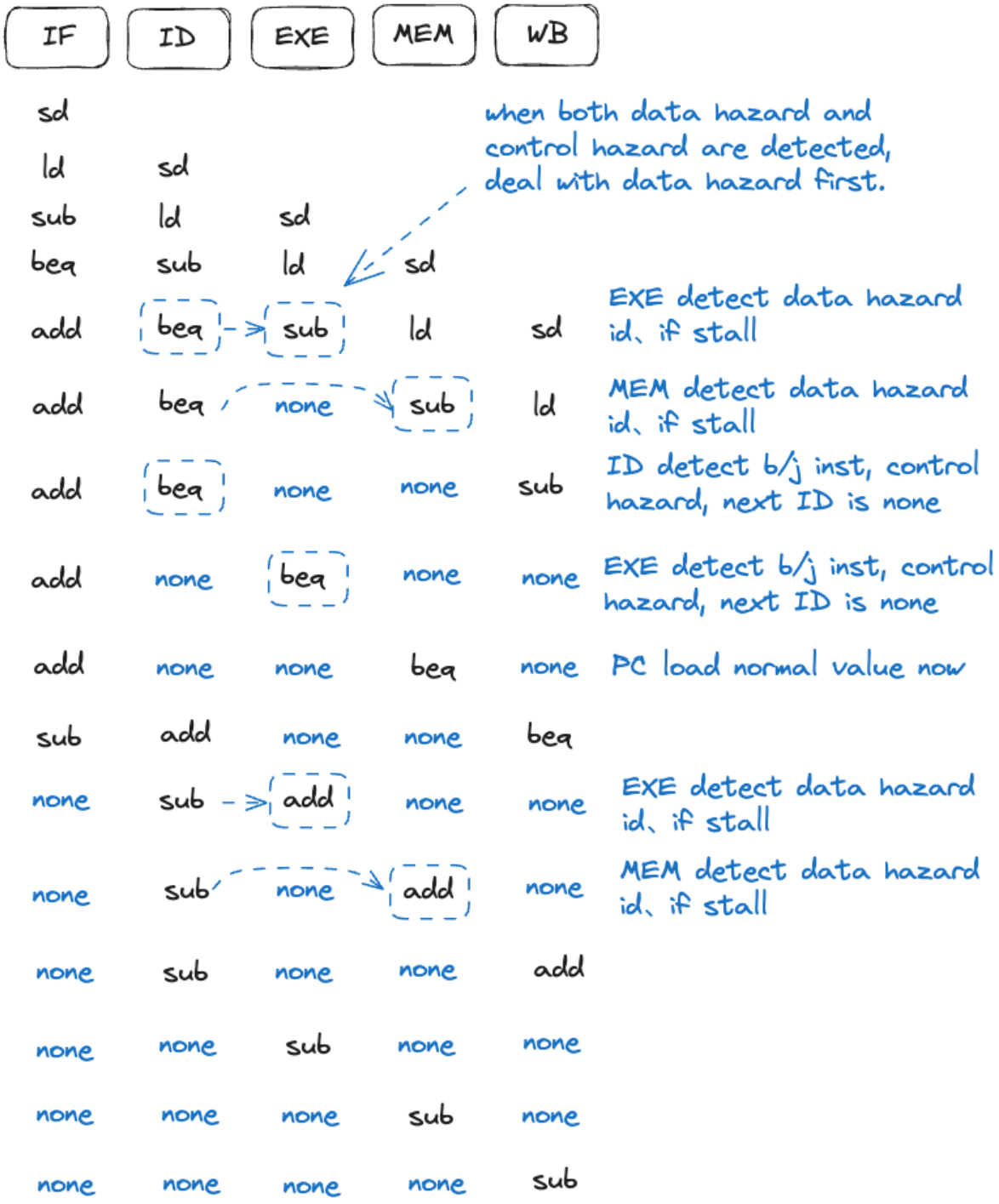
Answer:

```
SD x1, (0xff8)x2
```

2. 计算下面指令序列的 cpi，使用 double bubble 技术和 stall 技术处理数据竞争和控制竞争， 不考虑结构竞争

> 默认 beq 不跳转；double bubble 是指 regfile 下降沿写入的处理方法；不考虑 forward 技术；控制竞争停顿的拍数因硬件结构而异，如果有歧义，大家可以在边上注明自己的硬件实现方法，言之有理即可

```
Label:
        sd  x29, 12(x16)
        ld  x29,  8(x16)
        sub x17, x15, x14  # data hazard
        beq x17, x0, Label # control hazard
        add x15, x11, x14
        sub x15, x30, x15  # data hazard
```



CPI = 16/6 = 8/3

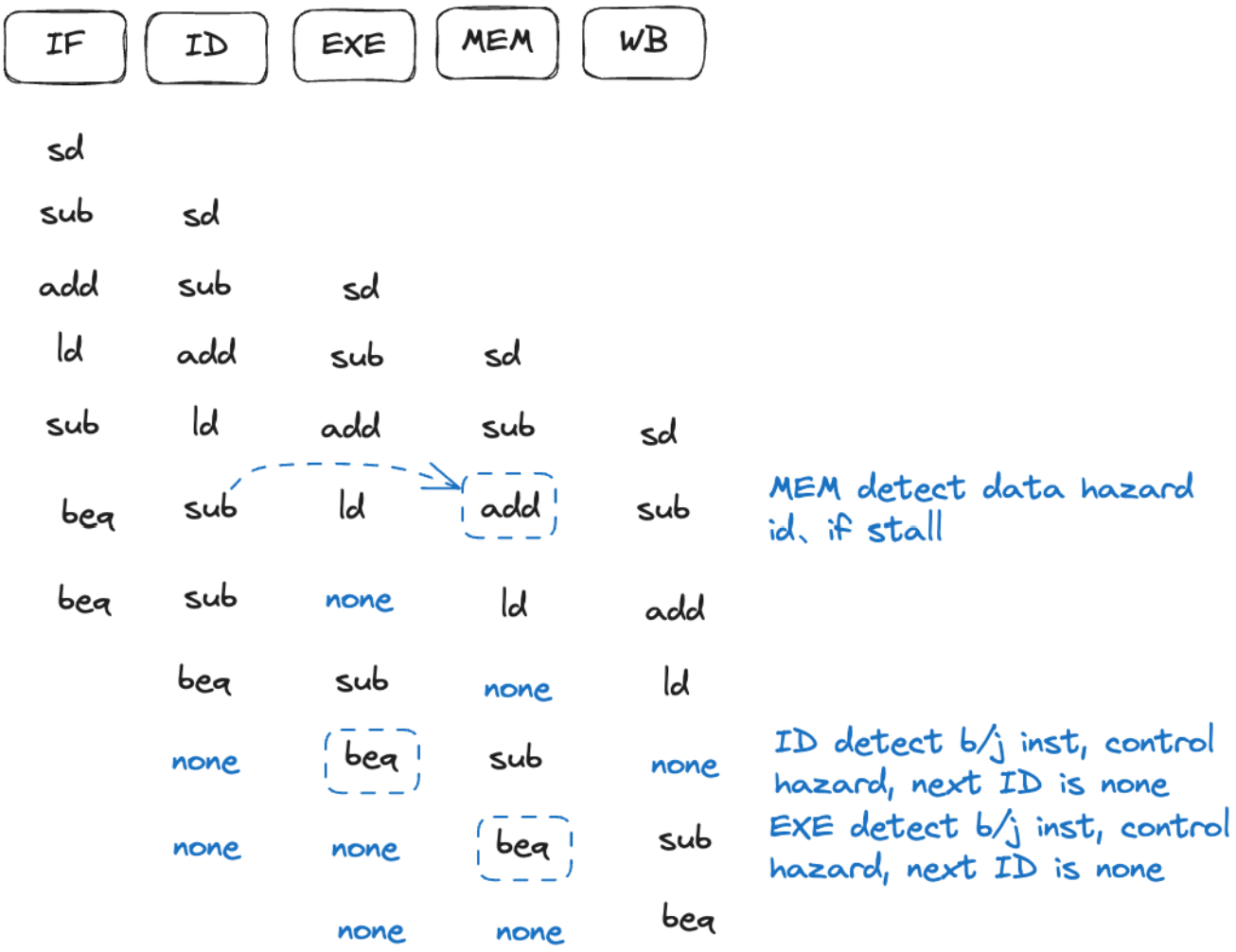
3. 计算下面指令序列的 cpi，使用 double bubble 技术和 stall 技术处理数据竞争和控制竞争， 不考虑结构竞争

```
Label:
        sd x29, 12(x16)
        sub x17, x15, x14 <--------------\
```

```
        add x15, x11, x14 <---\          |
        ld x29, 8(x16).       |          |
        sub x15, x30, x15  # data hazard |
        beq x17, x0, Label # maybe data hazard (in fact no)
```

比较 2 和 3 的指令序列的 cpi 和实现的功能的区别，比较之所以序列 3 的 cpi 更小的原因和 可以借鉴的 cpi 加速的方法

Answer：



cpi = 11/6

比较和分析：

指令序列2与指令序列3的主要区别在于指令的顺序。在指令序列3中，通过重新排序指令，数据依赖性减少，从而减少了因数据冲突而产生的额外周期。这是因为加载指令（可能产生数据冲突的主要来源）被放置在了序列的后半部分，减少了与其他指令的直接数据依赖。

结论和优化技术：

指令序列3相比于指令序列2具有更低的CPI，主要是由于通过指令重排减少了数据冲突和加载延迟的影响。这种优化技术表明，通过减少指令间的依赖性和考虑指令的最佳顺序，可以有效地提高流水线的性能。通过重新组织指令顺

序，以减少数据和控制冲突，从而减少 stall 的发生.