**计算机体系结构实验报告**

冯一鸣 21307346

**实验题目**

* 项目来源(论文)：

王磊.Tomasulo算法与记分牌调度算法研究[J].自动化技术与应用,2013,32(06):23-26.

* 参考：

根据<https://cszhouy.github.io/blog/bfcd1b20.html>所描述的记分牌算法进行实验，并在相同的指令序列上进行验证

**实验代码**

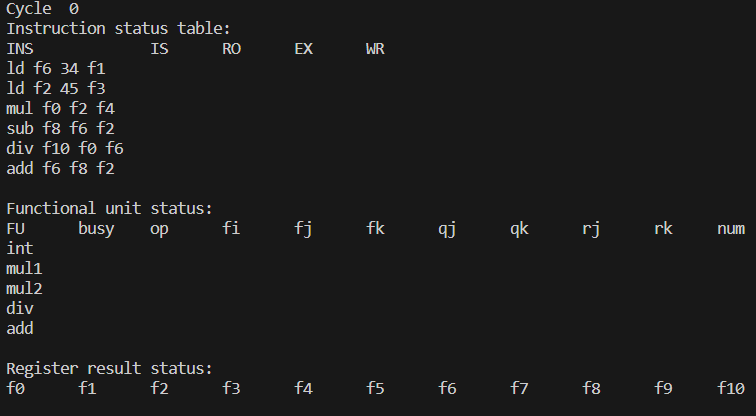
* <https://github.com/Supremacy-ysyyrps/Computer-Arch>

**实验目的**

* 了解记分牌的结构和工作原理
* 了解数据冒险对CPU性能的影响及解决方法
* 掌握记分牌算法并深刻体会乱序执行的优势

**实验内容**

* 使用python语言实现一个记分牌算法模拟程序
* 模拟的程序支持五种指令，包括：
  + 取数指令 ld
  + 乘法指令 mul
  + 除法指令 div
  + 加法指令 add
  + 减法指令 sub
* 拥有五个功能部件，包括：
  + 一个整数部件 int
  + 两个乘法部件 mul1, mul2
  + 一个除法部件 div
  + 一个加法部件 add
* 程序维护三个表格，分别为：
  + 指令状态表，指示每条指令在各时钟周期所处的阶段，各阶段含义如下：



IS表示指令发射阶段，

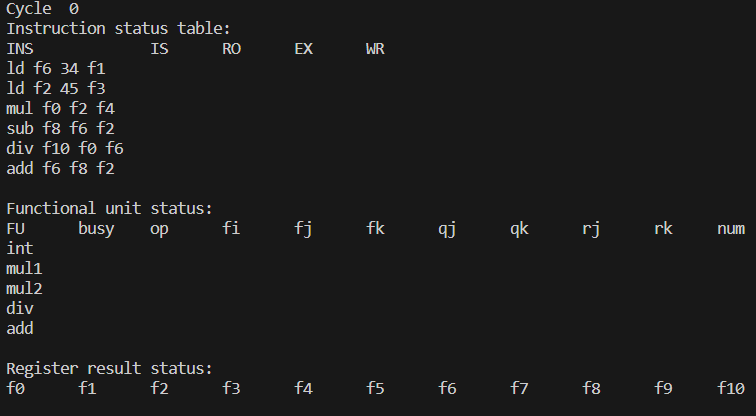
RO表示读取源操作数阶段，

EX表示执行阶段，

WR表示写回阶段。

* + 功能单元状态表，有9个字段表示每一个部件的状态

额外的num字段记录位于该功能单元的指令序号，仅用于展示：



busy：指示该单元是否繁忙

op：该单元正在进行的运算类型

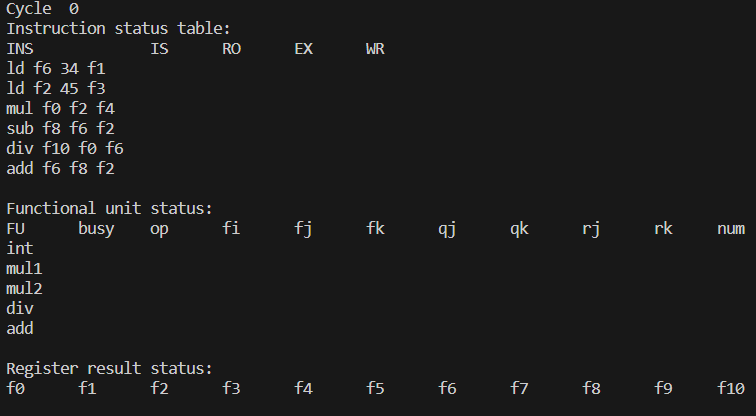
fi：运算结果的目标寄存器

fj、fk：源寄存器

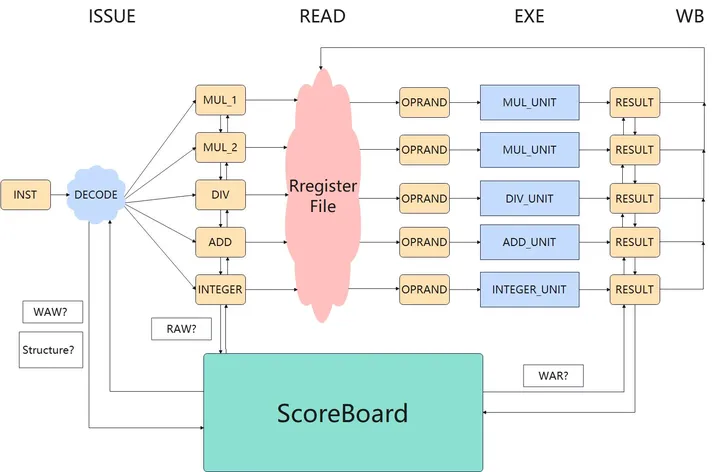
qj、qk：如果源寄存器fj，fk没有就绪，则存储生成它们的值的部件

rj、rk：指示源寄存器fj，fk是否就绪，在读取操作数后设置为False

* + 结果寄存器状态表，指出哪个功能单元将写入哪个寄存器



* CPU数据通路结构图



* 算法原理

由于使用了记分牌的CPU有多个功能部件，因此使用不同部件的指令可以并行执行，即使部分指令阻塞，也不会影响其他指令的执行，这就会导致指令的乱序执行。由此产生的数据冒险包括WAW，RAW和WAR三种，除此之外还存在结构冒险，三种数据冒险分别在IS，RO和WR阶段解决，另外，结构冒险在IS阶段解决。

**IS阶段**：

仅使用记分牌的CPU不具有多发射功能，一次只能发射一条指令。

首先从指令队列中取出位于队首的指令，然后检查功能单元状态表，判断该指令执行所需要的功能单元是否可用。如果该功能单元处于忙碌状态，则该指令在IS阶段阻塞，直到占有该功能单元的指令执行完。此步骤解决了结构冒险。

如果功能单元可用，则检查结果寄存器状态表，判断该指令的目的寄存器是否已经有指令需要写入。如果有，则该指令仍需阻塞。此步骤解决了可能存在的WAW冒险。

若不存在这两种冒险，CPU就可以发射该指令，同时填写功能单元状态表和结果寄存器状态表。

**RO阶段**：

根据IS阶段发射指令时在功能单元状态表的rj，rk字段填入的内容，判断每条已发射指令的所有源操作数是否就绪。若就绪，则读取源操作数，进入下一阶段，同时修改功能单元状态表的rj，rk字段为False（便于后续检查WAR冒险）；否则该指令在RO阶段阻塞，直到所需要的源操作数准备好。此阶段解决了RAW冒险。

**EX阶段**：

所有已经读取源操作数的指令进入功能单元执行指令操作。

**WR阶段**：

对每条等待写回的指令，判断当前是否有其他等待读取源操作数的指令（即rj或rk为true，这里就是为什么RO阶段后修改rj，rk字段为False），其源操作数为该指令的目的操作数。如果有，则该指令在WR阶段阻塞。否则直接写回，同时检查是否有依赖于该指令目的操作数的其他指令，如果有，则修改功能单元状态表中对应于有依赖指令的项。此阶段解决了WAR冒险。

**实验代码**

* 维护CPU当前状态的变量，其中FU\_STAT中的num仅用于更新指令状态表，便于观察。

# 指令的操作类型

OP = ["ld","mul","div","add","sub"]

# 指令状态表

INS\_STAT = []

# 功能部件

FU = ["int","mul1","mul2","div","add"]

# 功能部件状态表，num是占据该功能部件的指令在指令序列中的编号

STAT = ["busy","op","fi","fj","fk","qj","qk","rj","rk","num"]

FU\_STAT = {fu:{s:"" for s in STAT} for fu in FU}

# 结果寄存器状态表

REGS\_STAT = {"f0":"","f1":"","f2":"","f3":"","f4":"","f5":"",

"f6":"","f7":"","f8":"","f9":"","f10":""}

# 上一周期执行完RO的指令，用于本周期执行EX

flag\_ex = {fu:False for fu in FU}

# 上一周期执行完EX的指令，用于本周期执行WR

flag\_wr = {fu:False for fu in FU}

* 每个时钟周期进行如下操作，本周期的所有更新操作都在FU\_STAT\_（新周期的状态表）中进行，用于在周期结束时更新FU\_STAT（当前周期的状态表），所有的判断都在FU\_STAT中进行，因为判断能否进入下一阶段是基于当前周期的状态表的状态的。

# 开始执行

i = 0 # 当前位于指令队列队首的指令序号

c = 0 # 当前时钟周期序号

while i < len(INS\_STAT):

# 显示状态

print("Cycle ",c)

show()

c += 1

# 深拷贝FU\_STAT，因为状态是每个时钟周期更新一次

FU\_STAT\_ = copy.deepcopy(FU\_STAT)

# IS...

# RO...

# EX...

# WR...

# 周期结束，更新状态...

* IS阶段，输入的指令为字符串”op rd rs rt”或”op rt imm rs”，如”ld f6 34 f1”。

INS[0] = op，INS[1] = 目的寄存器，INS[2] = 源操作数或立即数，INS[3] = 另一个源操作数。

# IS

flag\_is = 0 # 本周期是否可以发射指令

INS = INS\_STAT[i]["INS"].split() # 取出当前位于指令队列队首的指令

if(INS[0] == "ld"): # 判断指令类型

# 根据指令类型检查所需功能单元是否可用（排除结构冒险）

# 检查是否有其他指令需要写入同一目的寄存器（排除WAW冒险）

# 若没有上述两种冒险，则发射指令并更新功能单元状态表和寄存器状态表

if(not FU\_STAT["int"]["busy"] and not REGS\_STAT[INS[1]]):

        FU\_STAT\_["int"]["busy"] = True

        FU\_STAT\_["int"]["op"] = "ld"

        FU\_STAT\_["int"]["fi"] = INS[1]

        FU\_STAT\_["int"]["fj"] = "null"

        FU\_STAT\_["int"]["fk"] = INS[3]

        FU\_STAT\_["int"]["qj"] = ""

        FU\_STAT\_["int"]["qk"] = REGS\_STAT[INS[3]]

        FU\_STAT\_["int"]["rj"] = not FU\_STAT\_["int"]["qj"]

        FU\_STAT\_["int"]["rk"] = not FU\_STAT\_["int"]["qk"]

        FU\_STAT\_["int"]["num"] = i

        REGS\_STAT[INS[1]] = 'int'

        flag\_is = 1

elif(INS[0] == 'mul'):

    m = ''

    if(not FU\_STAT["mul1"]["busy"]):

        m = "mul1"

    elif(not FU\_STAT["mul2"]["busy"]):

        m = "mul2"

    if(m and not REGS\_STAT[INS[1]]):

        FU\_STAT\_[m]["busy"] = True

        FU\_STAT\_[m]["op"] = "mul"

        FU\_STAT\_[m]["fi"] = INS[1]

        FU\_STAT\_[m]["fj"] = INS[2]

        FU\_STAT\_[m]["fk"] = INS[3]

        FU\_STAT\_[m]["qj"] = REGS\_STAT[INS[2]]

        FU\_STAT\_[m]["qk"] = REGS\_STAT[INS[3]]

        FU\_STAT\_[m]["rj"] = not FU\_STAT\_[m]["qj"]

        FU\_STAT\_[m]["rk"] = not FU\_STAT\_[m]["qk"]

        FU\_STAT\_[m]["num"] = i

        REGS\_STAT[INS[1]] = m

        flag\_is = 1

elif(INS[0] == "div"):

    if(not FU\_STAT["div"]["busy"] and not REGS\_STAT[INS[1]]):

        FU\_STAT\_["div"]["busy"] = True

        FU\_STAT\_["div"]["op"] = "div"

        FU\_STAT\_["div"]["fi"] = INS[1]

        FU\_STAT\_["div"]["fj"] = INS[2]

        FU\_STAT\_["div"]["fk"] = INS[3]

        FU\_STAT\_["div"]["qj"] = REGS\_STAT[INS[2]]

        FU\_STAT\_["div"]["qk"] = REGS\_STAT[INS[3]]

        FU\_STAT\_["div"]["rj"] = not FU\_STAT\_["div"]["qj"]

        FU\_STAT\_["div"]["rk"] = not FU\_STAT\_["div"]["qk"]

        FU\_STAT\_["div"]["num"] = i

        REGS\_STAT[INS[1]] = "div"

        flag\_is = 1

elif(INS[0] == "add"):

    if(not FU\_STAT["add"]["busy"] and not REGS\_STAT[INS[1]]):

        FU\_STAT\_["add"]["busy"] = True

        FU\_STAT\_["add"]["op"] = "mul"

        FU\_STAT\_["add"]["fi"] = INS[1]

        FU\_STAT\_["add"]["fj"] = INS[2]

        FU\_STAT\_["add"]["fk"] = INS[3]

        FU\_STAT\_["add"]["qj"] = REGS\_STAT[INS[2]]

        FU\_STAT\_["add"]["qk"] = REGS\_STAT[INS[3]]

        FU\_STAT\_["add"]["rj"] = not FU\_STAT\_["add"]["qj"]

        FU\_STAT\_["add"]["rk"] = not FU\_STAT\_["add"]["qk"]

        FU\_STAT\_["add"]["num"] = i

        REGS\_STAT[INS[1]] = "add"

        flag\_is = 1

elif(INS[0] == "sub"):

    if(not FU\_STAT["add"]["busy"] and not REGS\_STAT[INS[1]]):

        FU\_STAT\_["add"]["busy"] = True

        FU\_STAT\_["add"]["op"] = "sub"

        FU\_STAT\_["add"]["fi"] = INS[1]

        FU\_STAT\_["add"]["fj"] = INS[2]

        FU\_STAT\_["add"]["fk"] = INS[3]

        FU\_STAT\_["add"]["qj"] = REGS\_STAT[INS[2]]

        FU\_STAT\_["add"]["qk"] = REGS\_STAT[INS[3]]

        FU\_STAT\_["add"]["rj"] = not FU\_STAT\_["add"]["qj"]

        FU\_STAT\_["add"]["rk"] = not FU\_STAT\_["add"]["qk"]

        FU\_STAT\_["add"]["num"] = i

        REGS\_STAT[INS[1]] = "add"

        flag\_is = 1

# 如果可以发射指令，则更新指令状态表的当前指令行

# 同时可以取下一条指令

if flag\_is:

INS\_STAT[i]["IS"] = c

i += 1

* RO阶段，flag\_ro用于存储本周期可以读取操作数的功能单元，在下一周期赋值给flag\_ex。

将rj、rk的值置为false是为了方便WR阶段判断WAR冒险。

# RO阶段

flag\_ro = {fu:False for fu in FU}

# 检查每条已发射指令

for fu in FU:

# 如果源操作数就绪，则可读取操作数，同时更新FU\_STAT

if FU\_STAT[fu]["rj"] and FU\_STAT[fu]["rk"]:

FU\_STAT\_[fu]["rj"] = False

FU\_STAT\_[fu]["rk"] = False

FU\_STAT\_[fu]["qj"] = ""

FU\_STAT\_[fu]["qk"] = ""

flag\_ro[fu] = True

INS\_STAT[FU\_STAT\_[fu]["num"]]["RO"] = c

* EX阶段，flag\_ex用于存储本周期可以执行相应操作的功能单元，其值来自于上一周期的flag\_ro，同时在下一周期用于更新flag\_wr。

# EX

for fu in FU:

if flag\_ex[fu]:

INS\_STAT[FU\_STAT\_[fu]["num"]]["EX"] = c

* WR阶段，flag\_wr用于存储本周期等待写回的功能单元

# WR

for fu in FU:

# 对每条等待写回的指令

if flag\_wr[fu]:

flag\_wr\_ = 1 # 功能单元fu所执行的指令是否可以写回

# 判断是否有WAR冒险

for fu\_ in FU:

if ((FU\_STAT[fu\_]["fj"] == FU\_STAT[fu]["fi"] and

FU\_STAT[fu\_]["rj"]) or

(FU\_STAT[fu\_]["fk"] == FU\_STAT[fu]["fi"] and

FU\_STAT[fu\_]["rk"])):

flag\_wr\_ = 0

break

# 如果没有WAR冒险，可以写回

if flag\_wr\_:

# 写回后修改对该指令有依赖的指令在功能单元状态表中对应的项

for fu\_ in FU:

if FU\_STAT[fu\_]["qj"] == fu:

FU\_STAT\_[fu\_]["qj"] = ""

FU\_STAT\_[fu\_]["rj"] = True

if FU\_STAT[fu\_]["qk"] == fu:

FU\_STAT\_[fu\_]["qk"] = ""

FU\_STAT\_[fu\_]["rk"] = True

# 更新指令状态表

INS\_STAT[FU\_STAT\_[fu]["num"]]["WR"] = c

# 释放寄存器状态表

REGS\_STAT[FU\_STAT\_[fu]["fi"]] = ""

# 释放功能状态表项

for k in FU\_STAT\_[fu].keys():

FU\_STAT\_[fu][k] = ""

flag\_wr[fu] = False

* 时钟上升沿，更新状态

# 周期结束，更新状态

# 更新功能状态表

FU\_STAT = FU\_STAT\_

# 更新flag\_wr

# 包括本周期因WAR冒险不能写回的，加上EX阶段执行完的功能单元

for fu in flag\_wr.keys():

flag\_wr[fu] = flag\_wr[fu] or flag\_ex[fu]

# 更新flag\_ex为本周期成功读取源操作数的功能单元

flag\_ex = flag\_ro

**实验结果**

* 周期一

第〇条指令(ld f6 34 f1)：

修改指令状态表：

取数运算，需要int功能单元，当前可用，进入发射(IS)状态.

修改功能单元int状态表：

置busy项为True，op项为ld；

目的操作数fi为f6；

源操作数fj为常数(存null)，fk为f1；

没有功能单元需要写入fj，因此qj为空；

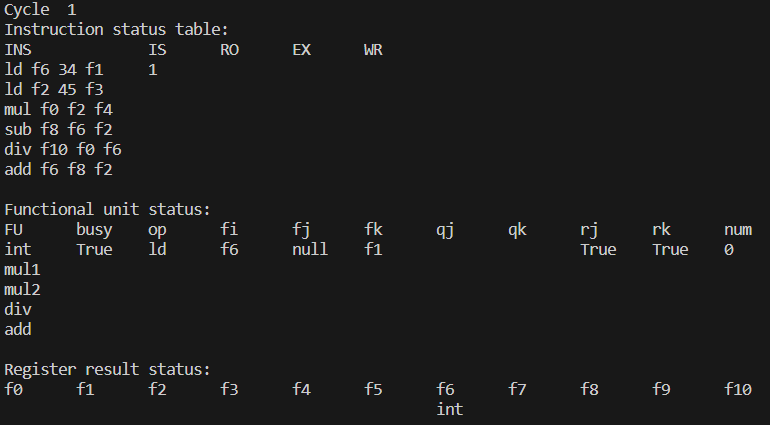
没有功能单元需要写入fk，因此qk为空；

源操作数fj已经就绪，rj为True；

源操作数fk已经就绪，rk为True.

修改结果寄存器状态表：

f6需要被int功能单元写入，f6的项为int.



* 周期二

第〇条指令(ld f6 34 f1)：

修改指令状态表：

可以读取操作数，进入读取源操作数(RO)状态.

修改功能单元int状态表：

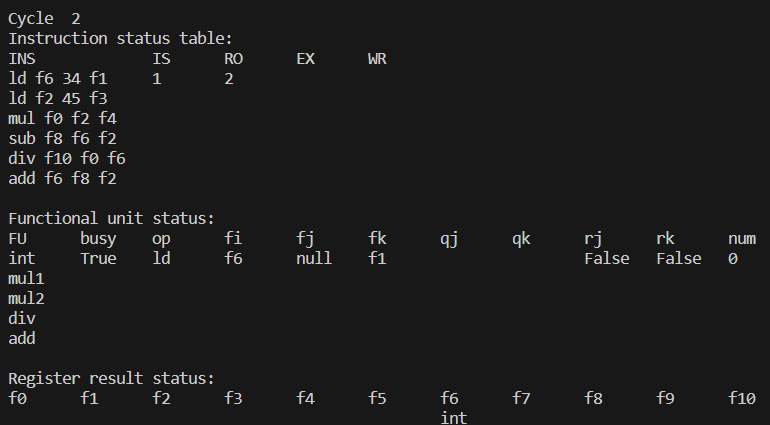
修改rj、rk的值为false，表示已经读完，后面的指令对这两个寄存器的写可以进行而不会影响对WAR冒险的判断.

修改结果寄存器状态表：

目的寄存器f6的项无需修改.

第一条指令(ld f2 45 f3)：

取数运算，需要int功能单元，当前不可用，出现结构冲突，阻塞.



* 周期三：

第〇条指令(ld f6 34 f1)：

修改指令状态表：

可以执行运算，进入执行(EX)状态.

修改功能单元int状态表：

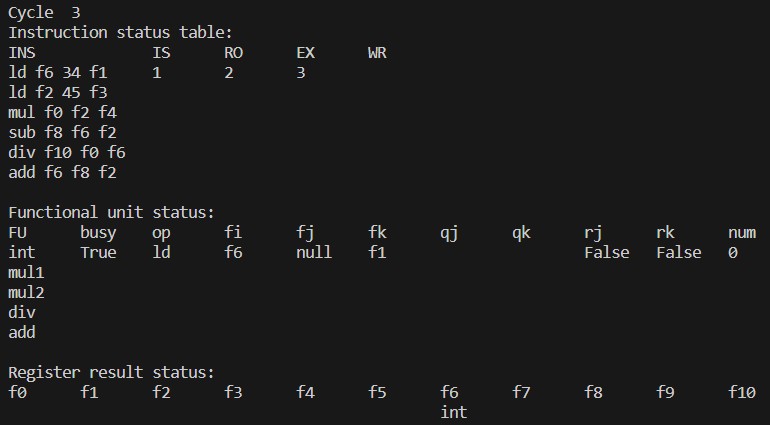
int功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f6的项无需修改.

第一条指令(ld f2 45 f3)：

取数运算，需要int功能单元，当前不可用，出现结构冲突，阻塞.



* 周期四：

第〇条指令(ld f6 34 f1)：

修改指令状态表：

可以进行写回，进入写回(WR)状态.

修改功能单元int状态表：

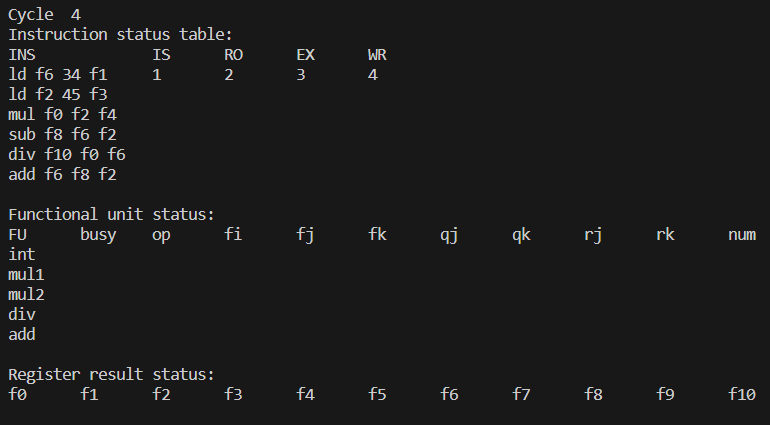
int功能单元被释放，状态清空.

修改结果寄存器状态表：

写回阶段结束，f6的项置空.

第一条指令(ld f2 45 f3)：

取数运算，需要int功能单元，本周期刚释放，下周期可用.



* 周期五：

第一条指令(ld f2 45 f3)：

修改指令状态表：

取数运算，需要int功能单元，当前可用，进入发射(IS)状态.

修改功能单元int状态表：

置busy项为True，op项为ld；

目的操作数fi为f2；

源操作数fj为常数(存null)，fk为f3；

没有功能单元需要写入fj，因此qj为空；

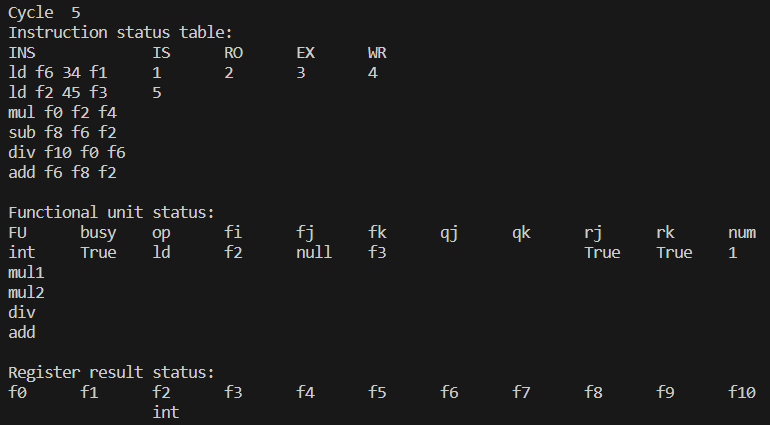
没有功能单元需要写入fk，因此qk为空；

源操作数fj已经就绪，rj为True；

源操作数fk已经就绪，rk为True.

修改结果寄存器状态表：

f2需要被int功能单元写入，f2的项为int.



* 周期六：

第一条指令(ld f2 45 f3)：

修改指令状态表：

可以读取操作数，进入读取源操作数(RO)状态.

修改功能单元int状态表：

修改rj、rk的值为false，表示已经读完.

修改结果寄存器状态表：

目的寄存器f2的项无需修改.

第二条指令(mul f0 f2 f4)：

修改指令状态表：

乘法运算，需要mul功能单元，当前可用，进入发射(IS)状态.

修改功能单元mul1状态表：

置busy项为True，op项为mul；

目的操作数fi为f0；

源操作数fj为f2，fk为f4；

fj需要等待功能单元int的结果，因此qj为int；

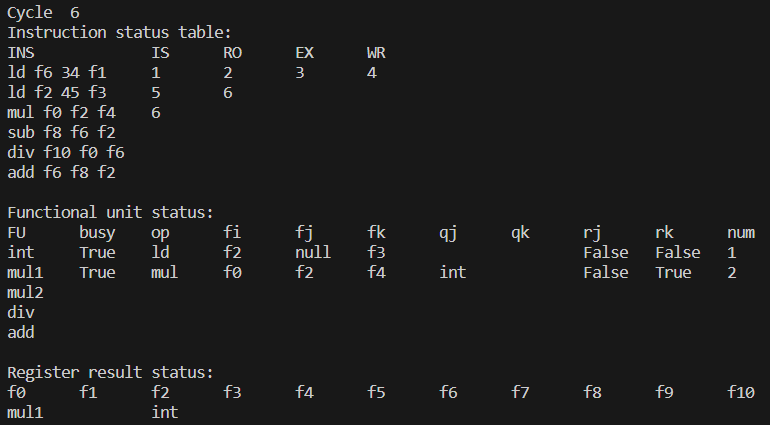
没有功能单元需要写入fk，因此qk为空；

源操作数fj尚未就绪，rj为False；

源操作数fk已经就绪，rk为True.

修改结果寄存器状态表：

f0需要被mul1功能单元写入，f0的项为mul1.



* 周期七：

第一条指令(ld f2 45 f3)：

修改指令状态表：

可以执行运算，进入执行(EX)状态.

修改功能单元int状态表：

int功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f2的项无需修改.

第二条指令(mul f0 f2 f4)：

修改指令状态表：

源操作数尚未准备好，阻塞.

修改功能单元mul1状态表：

mul1功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f0的项无需修改.

第三条指令(sub f8 f6 f2)：

修改指令状态表：

减法运算，需要add功能单元，当前可用，进入发射(IS)状态.

修改功能单元add状态表：

置busy项为True，op项为sub；

目的操作数fi为f8；

源操作数fj为f6，fk为f2；

没有功能单元需要写入fj，因此qj为空；

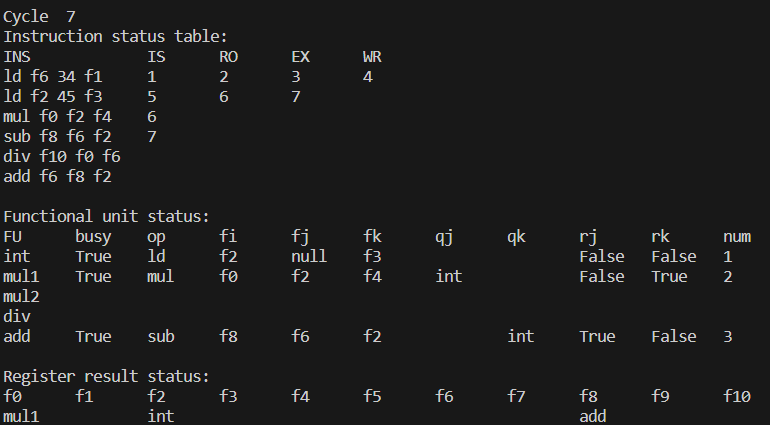
fk需要等待功能单元int的结果，因此qk为int；

源操作数fj已经就绪，rj为True；

源操作数fk尚未就绪，rk为False.

修改结果寄存器状态表：

f8需要被add功能单元写入，f8的项为add.



* 周期八：

第一条指令(ld f2 45 f3)：

修改指令状态表：

可以进行写回，进入写回(WR)状态.

修改功能单元int状态表：

int功能单元被释放，状态清空.

修改结果寄存器状态表：

写回阶段结束，f2的项置空.

第二条指令(mul f0 f2 f4)：

修改指令状态表：

源操作数刚由int功能单元写入，继续阻塞，下周期执行.

修改功能单元mul1状态表：

int功能单元写入fj寄存器，因此qj清空；

源操作数fj已经就绪，rj为True.

修改结果寄存器状态表：

目的寄存器f0的项无需修改.

第三条指令(sub f8 f6 f2)：

修改指令状态表：

源操作数刚由int功能单元写入，继续阻塞，下周期执行.

修改功能单元add状态表：

int功能单元写入fk寄存器，因此qk清空；

源操作数fk已经就绪，rk为True.

修改结果寄存器状态表：

目的寄存器f8的项无需修改.

第四条指令(div f10 f0 f6)

修改指令状态表：

除法运算，需要div功能单元，当前可用，进入发射(IS)状态.

修改功能单元div状态表：

置busy项为True，op项为div；

目的操作数fi为f10；

源操作数fj为f0，fk为f6；

fj需要等待功能单元mul1的结果，因此qj为mul1；

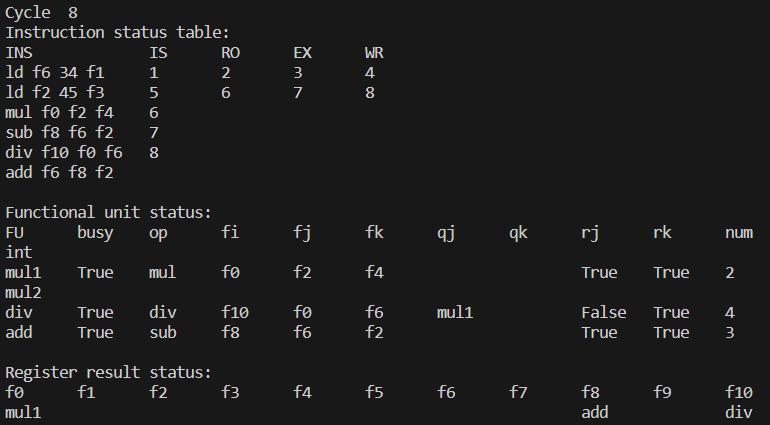
没有功能单元需要写入fk，因此qk为空；

源操作数fj尚未就绪，rj为False；

源操作数fk已经就绪，rk为True.

修改结果寄存器状态表：

f10需要被div功能单元写入，f10的项为div.



* 周期九：

第二条指令(mul f0 f2 f4)：

修改指令状态表：

可以读取操作数，进入读取源操作数(RO)状态.

修改功能单元mul1状态表：

修改rj、rk的值为false，表示已经读完.

修改结果寄存器状态表：

目的寄存器f0的项无需修改.

第三条指令(sub f8 f6 f2)：

修改指令状态表：

可以读取操作数，进入读取源操作数(RO)状态.

修改功能单元add状态表：

修改rj、rk的值为false，表示已经读完.

修改结果寄存器状态表：

目的寄存器f8的项无需修改.

第四条指令(div f10 f0 f6)

修改指令状态表：

源操作数尚未准备好，阻塞.

修改功能单元div状态表：

div功能单元的状态不需要修改.

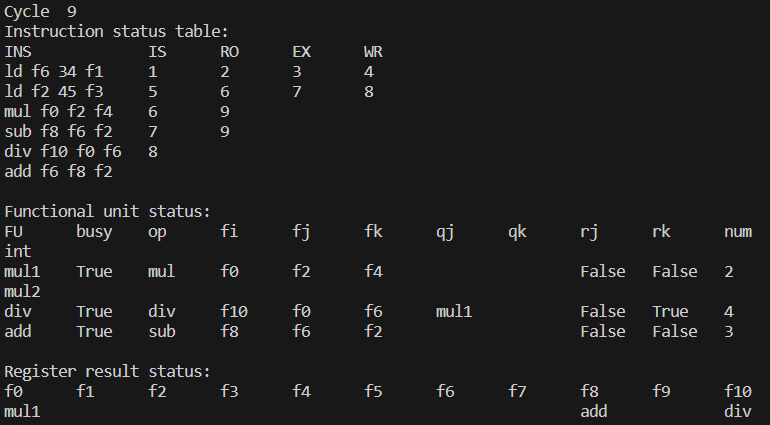
修改结果寄存器状态表：

目的寄存器f10的项无需修改.

第五条指令(add f6 f8 f2)

修改指令状态表：

加法运算，需要add功能单元，当前不可用，出现结构冲突，阻塞.



* 周期十：

第二条指令(mul f0 f2 f4)：

修改指令状态表：

可以执行运算，进入执行(EX)状态.

修改功能单元mul1状态表：

mul1功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f0的项无需修改.

第三条指令(sub f8 f6 f2)：

修改指令状态表：

可以执行运算，进入执行(EX)状态.

修改功能单元add状态表：

add功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f8的项无需修改.

第四条指令(div f10 f0 f6)

修改指令状态表：

源操作数尚未准备好，阻塞.

修改功能单元div状态表：

div功能单元的状态不需要修改.

修改结果寄存器状态表：

目的寄存器f10的项无需修改.

第五条指令(add f6 f8 f2)

修改指令状态表：

加法运算，需要add功能单元，当前不可用，出现结构冲突，阻塞.

