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：

```
Cycle  0
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1
ld f2 45 f3
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2
```

    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：

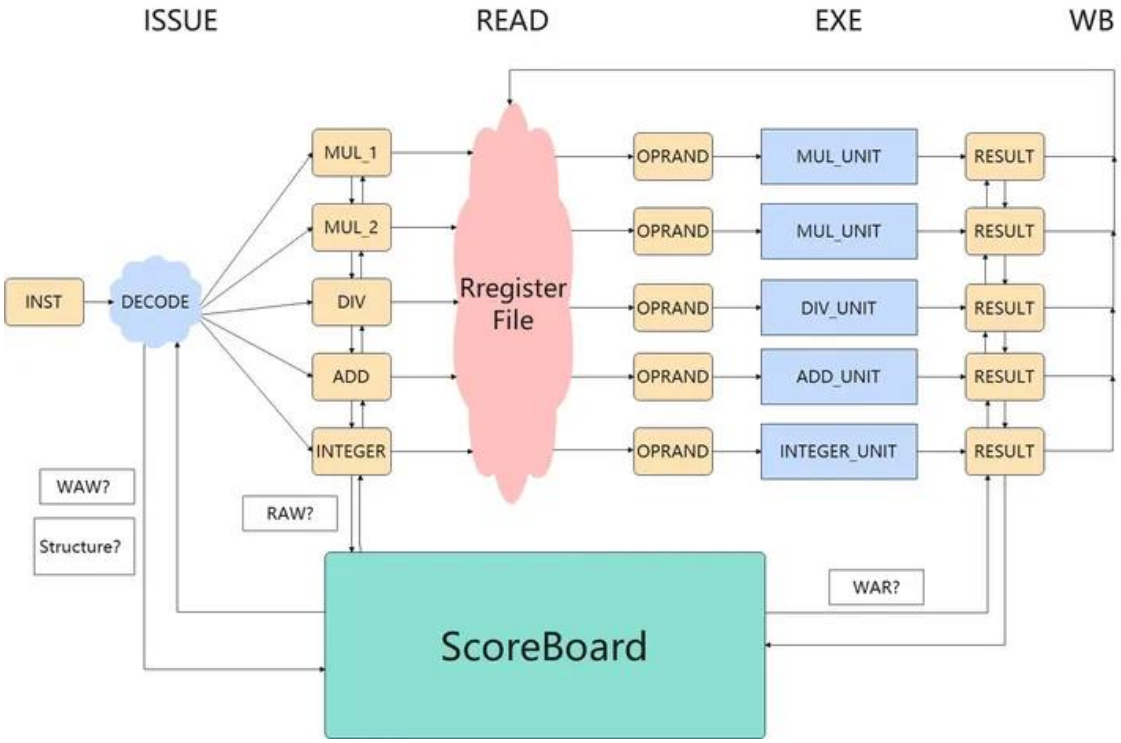
```
Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int
mul1
mul2
div
add
```

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

```
Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
```

- 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.

```
Cycle  1
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1
ld f2 45 f3
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f6      null    f1                      True    True    0
mul1
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
                                                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.

```
Cycle  2
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2
ld f2 45 f3
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f6      null    f1                      False   False   0
mul1
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
                                        int
```

- 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.

```
Cycle  3
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3
ld f2 45 f3
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f6      null    f1                      False   False   0
mul1
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
                                        int
```

- 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.

```
Cycle  4
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int
mul1
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
```

- 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.

```
Cycle  5
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5
mul f0 f2 f4
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f2      null    f3                      True    True    1
mul1
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
                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.

```
Cycle  6
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5       6
mul f0 f2 f4    6
sub f8 f6 f2
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f2      null    f3                      False   False   1
mul1    True    mul     f0      f2      f4      int             False   True    2
mul2
div
add

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
mul1            int
```

- 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.

```
Cycle  7
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5       6       7
mul f0 f2 f4    6
sub f8 f6 f2    7
div f10 f0 f6
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int     True    ld      f2      null    f3                      False   False   1
mul1    True    mul     f0      f2      f4      int             False   True    2
mul2
div
add     True    sub     f8      f6      f2              int     True    False   3

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
mul1            int                                             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.

```
Cycle  8
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5       6       7       8
mul f0 f2 f4    6
sub f8 f6 f2    7
div f10 f0 f6   8
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int
mul1    True    mul     f0      f2      f4                      True    True    2
mul2
div     True    div     f10     f0      f6      mul1            False   True    4
add     True    sub     f8      f6      f2                      True    True    3

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
mul1                                                            add             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.

```
Cycle  9
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5       6       7       8
mul f0 f2 f4    6       9
sub f8 f6 f2    7       9
div f10 f0 f6   8
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int
mul1    True    mul     f0      f2      f4                      False   False   2
mul2
div     True    div     f10     f0      f6      mul1            False   True    4
add     True    sub     f8      f6      f2                      False   False   3

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
mul1                                                            add             div
```

- 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.

```
Cycle  10
Instruction status table:
INS             IS      RO      EX      WR
ld f6 34 f1     1       2       3       4
ld f2 45 f3     5       6       7       8
mul f0 f2 f4    6       9       10
sub f8 f6 f2    7       9       10
div f10 f0 f6   8
add f6 f8 f2

Functional unit status:
FU      busy    op      fi      fj      fk      qj      qk      rj      rk      num
int
mul1    True    mul     f0      f2      f4                      False   False   2
mul2
div     True    div     f10     f0      f6      mul1            False   True    4
add     True    sub     f8      f6      f2                      False   False   3

Register result status:
f0      f1      f2      f3      f4      f5      f6      f7      f8      f9      f10
mul1                                                            add             div
```