Homework5 Tomasulo Algorithm Implementation

1.Introduction

```
'''
====================================================
===============[Some Prerequisites]================
====================================================
(1) Functional units are not pipelined.
(2) No forwarding, results are communicated by the common data bus (CDB).
(3) Loads require two clock cycle
(4) Issue (IS) and write-back (WB) result stages each require one clock cycle.
(5) Branch on Not Equal to Zero (BNEZ) instruction requires one clock cycle.
(6) There are 3 load buffer slots and 3 store buffer slots.
(7) There are 3 FP adder and 2 FP multiplier reservation stations, respectively.

----------------------------
FP instruction | EX Cycles
fadd           | 2
fsub           | 2
fmult          | 10
fdiv           | 20
----------------------------

output format
(1) While progressing output
Cycle_n
Reservation:(busy #yes or no), (address) # for load and store
Reservation:(busy #yes or no), (op), (Vj), (Vk), (Qj), (Qk) # for others
F0(status), F2(status), F4(status),...., F12(status) # for register result status
# where Vj, Vk indicate the existing source operands, Qj,Qk indicate which reservation
station will produce
the corresponding source operands(V and Q cannot both be set)
(2) Algorithm Complete: (Instruction):(Issue cycle),(Exec comp cyclee),(Write result
cycle)

'''
```

> Input1.txt

```
LD F6 34+ R2
LD F2 45+ R3
MULTD F0 F2 F4
SUBD F8 F6 F2
DIVD F10 F0 F6
ADDD F6 F8 F2
```

> input2.txt

```
LD F2 0 R2
LD F4 0 R3
DIVD F0 F4 F2
MULTD F6 F0 F2
ADDD F0 F4 F2
SD F6 0 R3
MULTD F6 F0 F2
SD F6 0 R1
```

2.Implementation

算法流程：

- （1）IF：从指令队列中取出指令（FIFO），如果指令对应的Reservation Station(RS)没有满，则可以发射（IS），否则等待；如果可以发射，则同时进行译码（ID），并将其放入对应的RS中，更新RS状态表，指令状态表（Instruction Status）和Register Result Status
- （2）EX：检查Load buffer、Store buffer、Add Reservation Station和Mult Reservation Station，若其中有指令可以执行（如操作数已经Ready），则更新其Reservation Station Status和指令状态表直到该指令执行完成后，可以进入下一阶段；
- （3）WB：若当前周期某指令可以写回，则更新指令状态表，若指令为DIVD、MULTD、ADDD或者LD，则更新其写入的寄存器堆，表示对应的操作数已经Ready，并更新Register Result Status，最后在对应的RS中移除该指令的记录。

每一个Cycle都会打印RS状态表及Register Result Status，在算法结束后，会打印指令状态表。

关键代码：

（更多代码注释请看源代码）

> IF,ID,IS阶段

```
 while not instructionQueue.empty() or not finish and not stall_reservationStation_Add
and not stall_reservationStation_Mult:
        ### Issue instruction (FIFO) ###

        # 检测RS是否已满，如果未满且指令队列未空则继续取指令
        occupied_Add, _ = check_if_full_and_fetch(reservationStation_Add)
        occupied_Mult, _ = check_if_full_and_fetch(reservationStation_Mult)

        if not occupied_Add:
            stall_reservationStation_Add = False
        if not occupied_Mult:
            stall_reservationStation_Mult = False


        if not instructionQueue.empty() and not stall_reservationStation_Add and not
stall_reservationStation_Mult:
            if len(stalled_instruction) != 0:
                cur_instruction = stalled_instruction[0]
                stalled_instruction.clear()
                ins_cnt -= 1

            else:
                cur_instruction = instructionQueue.get()

            if cur_instruction['opcode'] == 'ADDD' or cur_instruction['opcode'] ==
'SUBD':
                occupied, loc = check_if_full_and_fetch(reservationStation_Add)
                if not occupied:
                    reservationStation_Add[loc]['id'] = ins_cnt
                    reservationStation_Add[loc]['instruction'] = cur_instruction
                    reservationStation_Add[loc]['occupied'] =
reservationStation_Add[loc]['busy'] = True
                    reservationStation_Add[loc]['opcode'] = cur_instruction['opcode']
                    if register_file[cur_instruction['rs1']]:
                        reservationStation_Add[loc]['vj'] =
register_status_dict[cur_instruction['rs1']]
                    else:
                        reservationStation_Add[loc]['qj'] =
register_status_dict[cur_instruction['rs1']]

                    if register_file[cur_instruction['rs2']]:
                        reservationStation_Add[loc]['vk'] =
register_status_dict[cur_instruction['rs2']]
                    else:
                        reservationStation_Add[loc]['qk'] =
register_status_dict[cur_instruction['rs2']]


                    # 记录该指令所依赖的RS
                    # 用于解决数据依赖
                    if ins_cnt not in dependency.keys():
                        rs1 = register_status_dict[cur_instruction['rs1']]
                        rs2 = register_status_dict[cur_instruction['rs2']]

                        if cur_instruction['rs1'] in operand_from_load_fu.keys():
                            rs1 = operand_from_load_fu[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_fu.keys():
                            rs2 = operand_from_load_fu[cur_instruction['rs2']]

                        if cur_instruction['rs1'] in operand_from_load_mult.keys():
                            rs1 = operand_from_load_mult[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_mult.keys():
                            rs2 = operand_from_load_mult[cur_instruction['rs2']]

                        if cur_instruction['rs1'] in operand_from_load_add.keys():
                            rs1 = operand_from_load_add[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_add.keys():
                            rs2 = operand_from_load_add[cur_instruction['rs2']]

                        # 记录该指令依赖
                        dependency[ins_cnt] = {
                            'rs1':rs1,
                            'rs2':rs2
                        }


                    func_unit = 'Add'+str(loc+1)

                    register_file[func_unit+cur_instruction['rt']] = False
                    operand_from_load_add[cur_instruction['rt']] = func_unit

                    final_state_list[cur_instruction['id']]['issue_cycle'] = cycle

                # 如果Reservation Station是满的
                else:
                    stall_reservationStation_Add = True
                    stalled_instruction.append(cur_instruction)

            if cur_instruction['opcode'] == 'MULTD' or cur_instruction['opcode'] ==
'DIVD':
                occupied, loc = check_if_full_and_fetch(reservationStation_Mult)
                if not occupied:
                    reservationStation_Mult[loc]['id'] = ins_cnt
                    reservationStation_Mult[loc]['instruction'] = cur_instruction
                    reservationStation_Mult[loc]['occupied'] =
reservationStation_Mult[loc]['busy'] = True
                    reservationStation_Mult[loc]['opcode'] = cur_instruction['opcode']

                    if register_file[cur_instruction['rs1']]:
                        reservationStation_Mult[loc]['vj'] =
register_status_dict[cur_instruction['rs1']]
                    else:
                        reservationStation_Mult[loc]['qj'] =
register_status_dict[cur_instruction['rs1']]
                    if register_file[cur_instruction['rs2']]:
                        reservationStation_Mult[loc]['vk'] =
register_status_dict[cur_instruction['rs2']]
                    else:
                        reservationStation_Mult[loc]['qk'] =
register_status_dict[cur_instruction['rs2']]

                    # 记录该指令所依赖的RS
                    # 用于解决数据依赖
                    if ins_cnt not in dependency.keys():
                        rs1 = register_status_dict[cur_instruction['rs1']]
                        rs2 = register_status_dict[cur_instruction['rs2']]
                        if cur_instruction['rs1'] in operand_from_load_fu.keys():
                            rs1 = operand_from_load_fu[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_fu.keys():
                            rs2 = operand_from_load_fu[cur_instruction['rs2']]

                        if cur_instruction['rs1'] in operand_from_load_mult.keys():
                            rs1 = operand_from_load_mult[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_mult.keys():
                            rs2 = operand_from_load_mult[cur_instruction['rs2']]

                        if cur_instruction['rs1'] in operand_from_load_add.keys():
                            rs1 = operand_from_load_add[cur_instruction['rs1']]
                        if cur_instruction['rs2'] in operand_from_load_add.keys():
                            rs2 = operand_from_load_add[cur_instruction['rs2']]

                        # 记录该指令依赖
                        dependency[ins_cnt] = {
                            'rs1':rs1,
                            'rs2':rs2
                        }

                    func_unit = 'Mult' + str(loc + 1)
                    register_file[func_unit+cur_instruction['rt']] = False
                    operand_from_load_mult[cur_instruction['rt']] = func_unit

                    register_status_dict[cur_instruction['rt']] = func_unit

                    final_state_list[cur_instruction['id']]['issue_cycle'] = cycle

                # 如果Reservation Station是满的
                else:
                    stall_reservationStation_Mult = True
                    # 将其放入被stall的指令列表
                    stalled_instruction.append(cur_instruction)


            if cur_instruction['opcode'] == 'LD':
                occupied, loc = check_if_full_and_fetch(load_buffer)
                if not occupied:
                    load_buffer[loc]['id'] = ins_cnt
                    load_buffer[loc]['instruction'] = cur_instruction
                    load_buffer[loc]['occupied'] = load_buffer[loc]['busy'] = True
                    final_state_list[cur_instruction['id']]['issue_cycle'] = cycle
                    op = cur_instruction['rs1']+cur_instruction['rs2']
                    func_unit = 'Load' + str(loc + 1)
                    register_file[func_unit+cur_instruction['rt']] = False

                    operand_from_load_fu[cur_instruction['rt']] = func_unit
                    if cur_instruction['rs1'] == '0':
                        op = cur_instruction['rs2']
                    operand_from_load[cur_instruction['rt']] =
register_status_dict[cur_instruction['rt']] = func_unit
                    load_buffer[loc]['address'] = op


            if cur_instruction['opcode'] == 'SD':
                occupied, loc = check_if_full_and_fetch(store_buffer)
                if not occupied:
                    store_buffer[loc]['id'] = ins_cnt
                    store_buffer[loc]['instruction'] = cur_instruction
                    store_buffer[loc]['occupied'] = store_buffer[loc]['busy'] = True
                    store_buffer[loc]['fu'] =
register_status_dict[cur_instruction['rt']]
                    final_state_list[cur_instruction['id']]['issue_cycle'] = cycle
                    if ins_cnt not in dependency.keys():
                            dependency[ins_cnt] = {
                            'rt':register_status_dict[cur_instruction['rt']],

                        }
```

代码描述：

在这里一共完成了算法的三个阶段，分别是IF，ID和IS，其中，IS指令发射取决于当前的队列是否未空，该指令对应的RS是否已满， 如果RS已满则将其stall，放入stalled_instruction列表，直到该RS中的指令已经执行完成，并从RS中移除后，该指令才能被发射并进入RS。

在发射时，为了解决之后可能会发生的冲突，还会记录该指令相关的信息，`register_status_dict`字典记录了结果操作数状态，在`operand_from_load_add`,`operand_from_load_mult`,`operand_from_load`字典中记录了该指令的结果操作数将会从哪个RS中的哪一项产生（之所以不使用`register_status_dict`是因为这个字典再操作数写回寄存器后会记录其从内存中具体位置所得的操作数的运算），此外，还使用了`dependency`字典，来记录该指令的源操作数由哪个RS中的哪一项所产生，如此可以解决数据依赖。

EX, WB阶段

```
 ### Execute instructions ###
        for i, load_ins in enumerate(load_buffer):
            if load_ins['busy']:
                if final_state_list[load_ins['id']]['issue_cycle'] + 2 == cycle:
                    final_state_list[load_ins['id']]['exec_comp_cycle'] = cycle

                # Write Back
                if final_state_list[load_ins['id']]['exec_comp_cycle'] != '' \
                        and final_state_list[load_ins['id']]['exec_comp_cycle'] + 1 ==
cycle:

                    op = 'M(' + load_ins['instruction']['rs1'] +
 load_ins['instruction']['rs2'] + ')'
                    if load_ins['instruction']['rs1'] == '0':
                        op = 'M(' + load_ins['instruction']['rs2'] + ')'
                    register_status_dict[load_ins['instruction']['rt']] = \
                        operand_from_load[load_ins['instruction']['rt']] = op
                    final_state_list[load_ins['id']]['write_result_cycle'] = cycle

                    func_unit = 'Load' + str(i + 1)
                    register_file[func_unit+load_ins['instruction']['rt']] = True
                    # 清除表项
                    load_ins['busy'] = load_ins['occupied'] = False
                    load_ins['address'] = ''
                    load_ins['id'] = -1


        for i, res_add in enumerate(reservationStation_Add):
            # 在每一个cycle都检查处于busy状态的指令需要的操作数是否已经Ready
            if res_add['busy']:
                if res_add['id'] in dependency.keys():
                    rs1_reg = dependency[res_add['id']]['rs1'] + res_add['instruction']
['rs1'] # FU + 源寄存器1

                    if register_file[rs1_reg]:  # 若源操作数rs1已经被写入寄存器堆
                        res_add['qj'] = ''
                        res_add['vj'] = register_status_dict[res_add['instruction']
['rs1']]
                else:
                    if register_status_dict[res_add['instruction']['rs1']] != '':
                        res_add['qj'] = register_status_dict[res_add['instruction']
['rs1']]

                if res_add['id'] in dependency.keys():
                    rs2_reg = dependency[res_add['id']]['rs2'] + res_add['instruction']
['rs2'] # FU + 源寄存器2
                    if register_file[rs2_reg]:  # 若源操作数rs2已经被写入寄存器堆
                        res_add['qk'] = ''
                        res_add['vk'] = register_status_dict[res_add['instruction']
['rs2']]

                else:
                    if register_status_dict[res_add['instruction']['rs2']] != '':
                        res_add['qk'] = register_status_dict[res_add['instruction']
['rs2']]

                # 如果两个源操作数都已经Ready，那么可以进入EX Stage，add，sub都需要2个cycle来执
行
                if res_add['vj'] != '' and res_add['vk'] != '' and
final_state_list[res_add['id']]['exec_comp_cycle'] == '':
                    final_state_list[res_add['id']]['exec_comp_cycle'] = cycle + 2

                # Write Back, update FU
                if final_state_list[res_add['id']]['exec_comp_cycle'] != '' and \
                        final_state_list[res_add['id']]['exec_comp_cycle'] + 1 ==
cycle:
                    final_state_list[res_add['id']]['write_result_cycle'] = cycle


                    func_unit = 'Add' + str(i + 1)
                    rt = func_unit+res_add['instruction']['rt']

                    register_file[rt] = True

                    op1 = register_status_dict[res_add['instruction']['rs1']]
                    op2 = register_status_dict[res_add['instruction']['rs2']]
                    if res_add['instruction']['rs1'] in operand_from_load.keys():
                        op1 = operand_from_load[res_add['instruction']['rs1']]
                    if res_add['instruction']['rs2'] in operand_from_load.keys():
                        op2 = operand_from_load[res_add['instruction']['rs2']]

                    if res_add['opcode'] == "SUBD":
                        register_status_dict[res_add['instruction']['rt']] = \
                            op1 + '-' + op2
                        if res_add['instruction']['rt'] not in
dependency_res_status_dict.keys():
                            dependency_res_status_dict[rt] = op1 + '-' + op2

                    else:
                        register_status_dict[res_add['instruction']['rt']] = \
                            op1 + '+' + op2
                        if res_add['instruction']['rt'] not in
dependency_res_status_dict.keys():
```

```
                                 dependency_res_status_dict[rt] = op1 + '+' + op2


                        remove_reservation_station(reservationStation_Add, i)



            for i, res_mult in enumerate(reservationStation_Mult):
                if res_mult['busy']:
                    if res_mult['id'] in dependency.keys():
                        rs1_reg = dependency[res_mult['id']]['rs1']+res_mult['instruction']
['rs1'] # FU + 源寄存器1
                        if register_file[rs1_reg]:  # 若源操作数rs1已经被写入寄存器堆
                            res_mult['qj'] = ''
                            res_mult['vj'] = register_status_dict[res_mult['instruction']
['rs1']]
                            if rs1_reg in dependency_res_status_dict.keys():
                                res_mult['vj'] = dependency_res_status_dict[rs1_reg] # 结果
状态避免被其他写相同寄存器的指令覆盖

                    else:
                        if register_status_dict[res_mult['instruction']['rs1']] != '':
                            res_mult['qj'] = register_status_dict[res_mult['instruction']
['rs1']]

                    if res_mult['id'] in dependency.keys():
                        rs2_reg = dependency[res_mult['id']]['rs2']+res_mult['instruction']
['rs2'] # FU + 源寄存器2

                        if register_file[rs2_reg]:  # 若源操作数rs2已经被写入寄存器堆

                            res_mult['qk'] = ''
                            res_mult['vk'] = register_status_dict[res_mult['instruction']
['rs2']]

                    else:
                        if register_status_dict[res_mult['instruction']['rs2']] != '':
                            res_mult['qk'] = register_status_dict[res_mult['instruction']
['rs2']]

                    # mult需要10个cycle, div需要20个cycle
                    if res_mult['vj'] != '' and res_mult['vk'] != '' and
final_state_list[res_mult['id']]['exec_comp_cycle'] == '':
                        if res_mult['opcode'] == 'MULTD':
                            final_state_list[res_mult['id']]['exec_comp_cycle'] = cycle +
10
                        else:
                            final_state_list[res_mult['id']]['exec_comp_cycle'] = cycle +
20
```

```
                # Write Back
                if final_state_list[res_mult['id']]['exec_comp_cycle'] != '' and \
                        final_state_list[res_mult['id']]['exec_comp_cycle'] + 1 ==
cycle:
                    final_state_list[res_mult['id']]['write_result_cycle'] = cycle

                    func_unit = 'Mult' + str(i + 1)
                    register_file[func_unit+res_mult['instruction']['rt']] = True

                    op1 = register_status_dict[res_mult['instruction']['rs1']]
                    op2 = register_status_dict[res_mult['instruction']['rs2']]
                    if res_mult['instruction']['rs1'] in operand_from_load.keys():
                        op1 = operand_from_load[res_mult['instruction']['rs1']]
                    if res_mult['instruction']['rs2'] in operand_from_load.keys():
                        op2 = operand_from_load[res_mult['instruction']['rs2']]

                    if res_mult['opcode'] == "MULTD":
                        register_status_dict[res_mult['instruction']['rt']] = op1 + '*'
+ op2
                    else:
                        register_status_dict[res_mult['instruction']['rt']] = op1 + '/'
+ op2

                    remove_reservation_station(reservationStation_Mult, i)


        for i, store_ins in enumerate(store_buffer):
            if store_ins['busy']:
                if store_ins['id'] in dependency.keys():
                    # Write Back
                    rt_reg = dependency[store_ins['id']]['rt'] +
store_ins['instruction']['rt']# FU + 目的寄存器
                    if register_file[rt_reg]:  # 若目的寄存器rt已经被写入寄存器堆
                        final_state_list[store_ins['id']]['exec_comp_cycle'] = cycle +
2
                        register_file[rt_reg] = False # 这里置成False,表示已经取到rt且避免因
为下一个周期又将EX Comp的值再增加2

                    if final_state_list[store_ins['id']]['exec_comp_cycle'] != '' and \
                    final_state_list[store_ins['id']]['exec_comp_cycle'] + 1 == cycle:
                        final_state_list[store_ins['id']]['write_result_cycle'] = cycle
                        # 清除表项
                        store_ins['busy'] = store_ins['occupied'] = False
                        store_ins['address'] = ''
                        store_ins['fu'] = ''
                        store_ins['id'] = -1
```

代码描述：

这部分代码完成了指令的EX和WB阶段的操作，`final_state_list`字典用于记录指令的状态（包括issue_cycle、exec_comp_cycle、write_result_cycle的具体周期数），通过遍历每一个RS，可以根据指令对应的源操作数来判断是否执行该指令，在`register_file`字典中（寄存器堆），保存了所有寄存器的状态（格式为：FU+目的寄存器: True/False，如：'Mult1F0': True'），如果某RS中所产生的操作数已经被写入寄存器堆则置为True，指令可以据此来判断是否可以执行。在执行完成后，进入WB阶段，指令可以判断当前Cycle是否可以写回，写回寄存器堆后，同时更新`register_status_dict`，`register_file` 和`final_state_list`，并将该指令从对应的RS中移除。

3.Result

> Output1.txt

```
LD F6 34+ R2:1,3,4;
LD F2 45+ R3:2,4,5;
MULTD F0 F2 F4:3,15,16;
SUBD F8 F6 F2:4,7,8;
DIVD F10 F0 F6:5,36,37;
ADDD F6 F8 F2:6,10,11;
```

结果分析：

从结果来看，这六条指令分别在前6个cycle依次issue，并且两条LD指令在第3，4个cycle和第4，5个cycle完成了执行和写回阶段，MULTD指令由于需要10个cycle来执行（在第6个cycle开始时可以执行）到第15个cycle执行结束，SUBD需要2个cycle执行（在第6个cycle开始时可以执行）到第7个cycle执行结束，DIVD指令依赖于MULTD指令的结果F0，所以等到第17个cycle才开始执行，需要执行20个cycle，因此在第36个cycle执行结束，ADDD指令依赖于SUBD指令的结果F8，所以在第9个cycle才开始执行，需要2个cycle执行。

> Output2.txt

```
LD F2 0 R2:1,3,4;
LD F4 0 R3:2,4,5;
DIVD F0 F4 F2:3,25,26;
MULTD F6 F0 F2:4,36,37;
ADDD F0 F4 F2:5,7,8;
SD F6 0 R3:6,39,40;
MULTD F6 F0 F2:27,37,38;
SD F6 0 R1:28,40,41;
```

结果分析：

从结果来看，前六条指令分别在前6个cycle依次issue，并且两条LD指令在第3，4个cycle和第4，5个cycle完成了执行和写回阶段，第7条的MULTD F6 F0 F2指令由于RS已满（被DIVD F0 F4 F2和MULTD F6 F0 F2指令占用）所以需要等到它们其中之一的指令执行完才能被issue，可以看到，DIVD指令在26个cycle写回，MULTD F6 F0 F2指令在第37个cycle写回（因为这条指令依赖于DIVD指令的结果F0），所以到第27个cycle，MULTD F6 F0 F2指令才被issue到对应的RS中，它需要执行10个cycle，所以在第27个cycle时执行完毕，最终，最后一条指令SD F6 0 R1将它的上一条指令MULTD F6 F0 F2所产生的结果F6保存到内存的对应位置中。