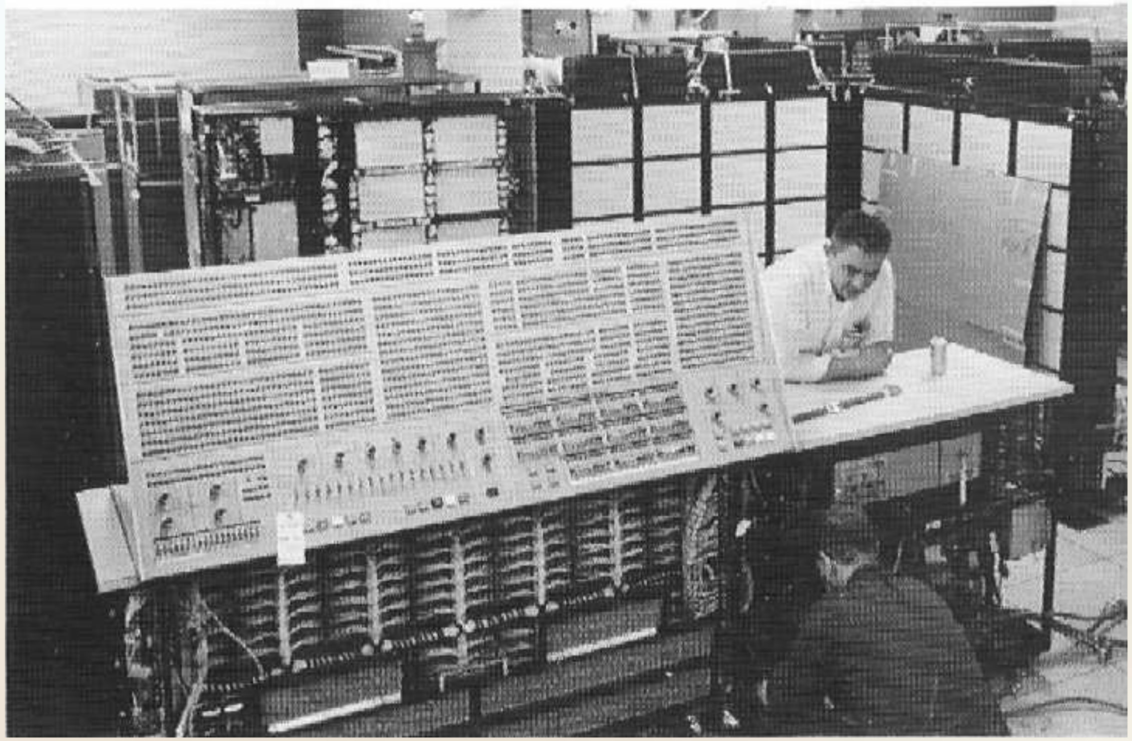
Instruction-level Parallelism

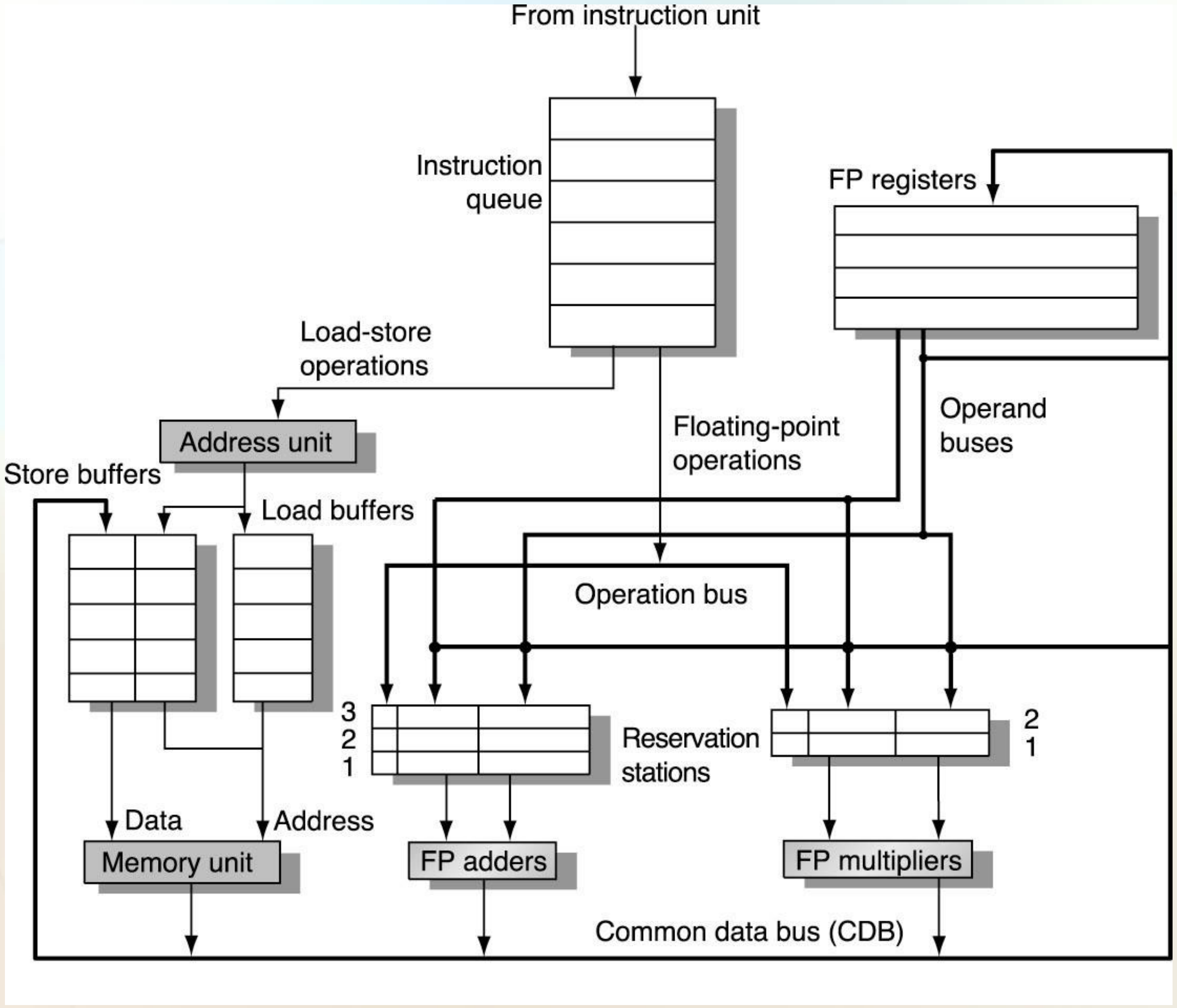
Tomasulo's Algorithm

- Developed for architecture of IBM 360/91 (1967)
  - 360/91 system's goal was to significantly improve performance (especially floating-point) without requiring people to change their code
    - Sound familiar?



16MHz
2MB Mem
50X faster
Than SOA

Tomasulo Organization



Tomasulo Algorithm

- Consider three input instructions
- Common Data Bus broadcasts results to all FUs

  RS's (FU's), registers, etc. responsible for collecting own data off CDB
- Load and Store Queues treated as FUs as well

Reservation Station Components

Op—Operation to perform in the unit (e.g., + or –)

Qj, Qk—Reservation stations producing source registers

Vj, Vk—Value of Source operands

Rj, Rk—Flags indicating when Vj, Vk are ready

Busy—Indicates reservation station is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station free, the scoreboard issues instr & sends operands (renames registers).

2. Execution—operate on operands (EX)

   When both operands ready then execute;
   if not ready, watch CDB for result

3. Write result—finish execution (WB)

   Write on Common Data Bus to all waiting units; mark reservation station available.

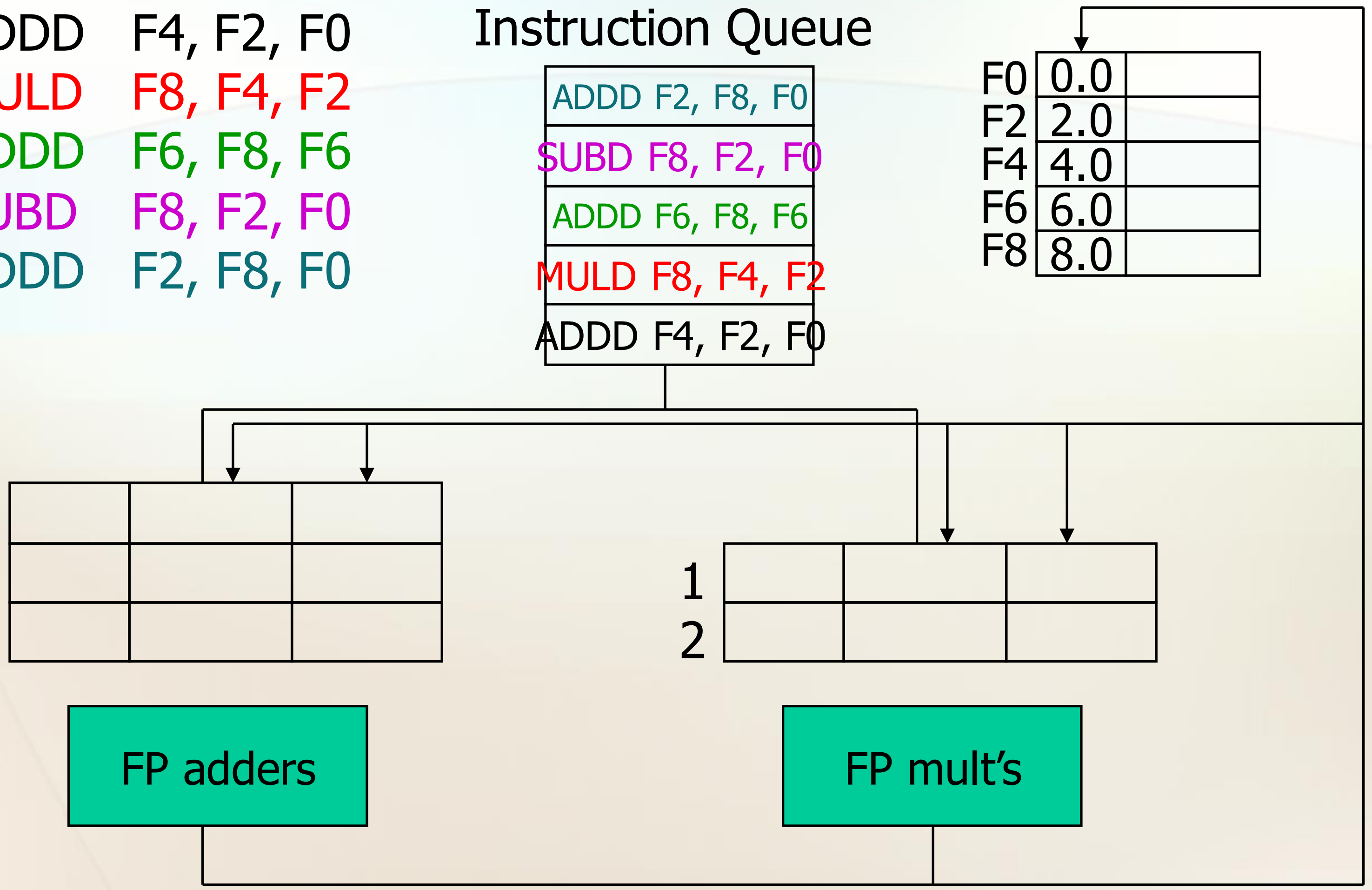
Tomasulo Example

```
ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0
```

Multiply takes 10 clocks, add/sub take 4

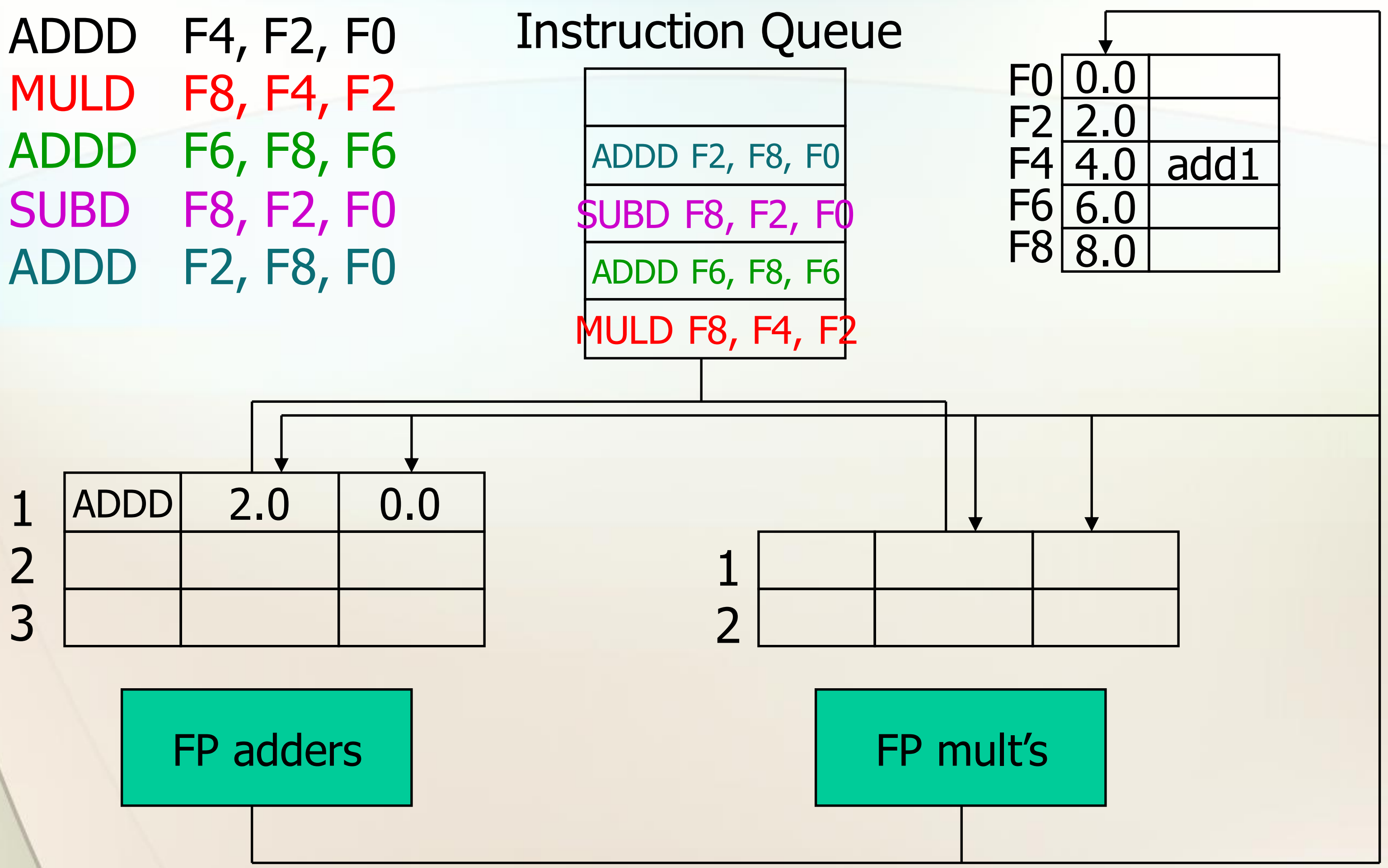
Tomasulo – cycle 0

```
ADDD   F4, F2, F0
MULD   F8, F4, F2
ADDD   F6, F8, F6
SUBD   F8, F2, F0
ADDD   F2, F8, F0
```



Tomasulo – cycle 1

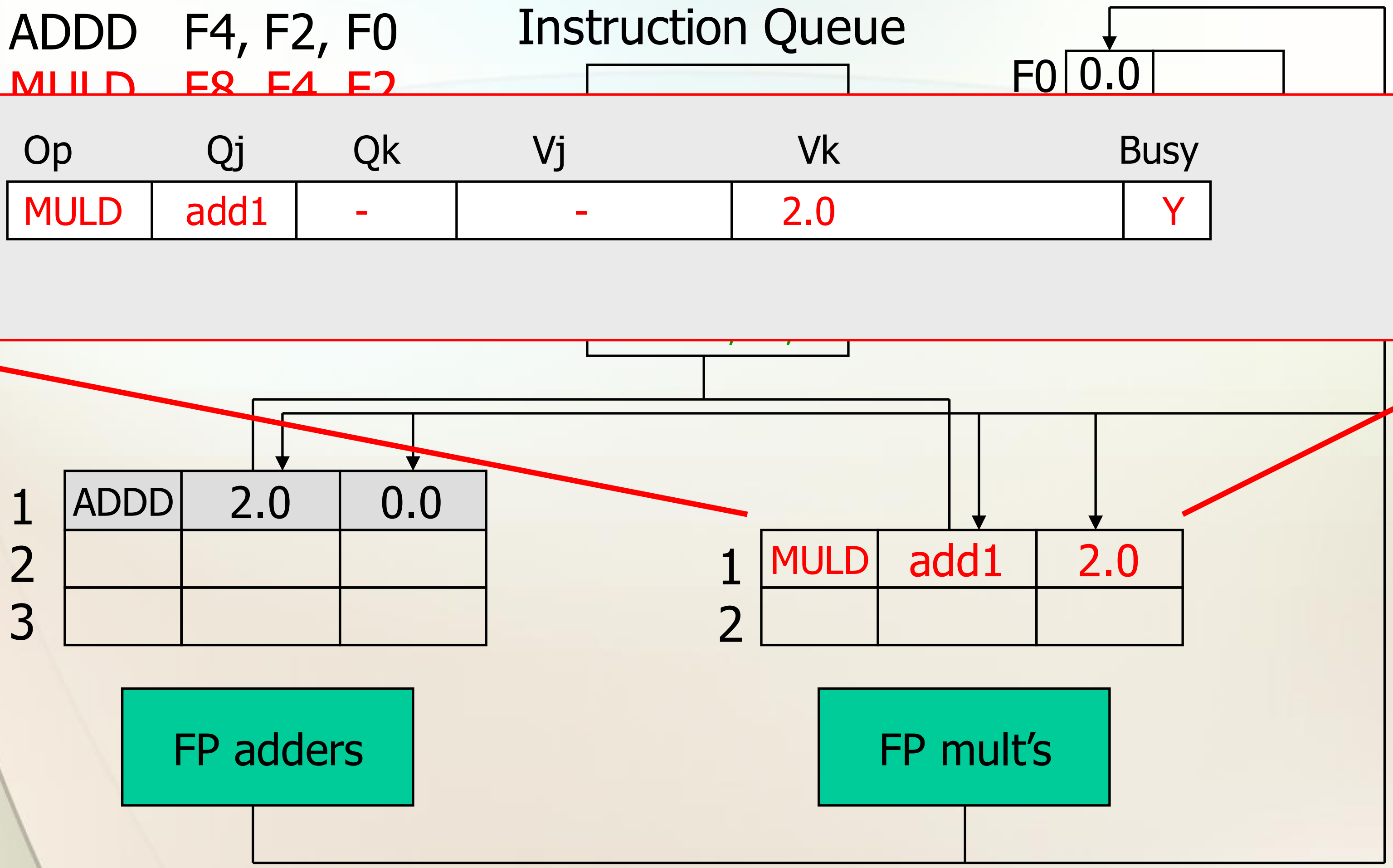
ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



Tomasulo – cycle 2

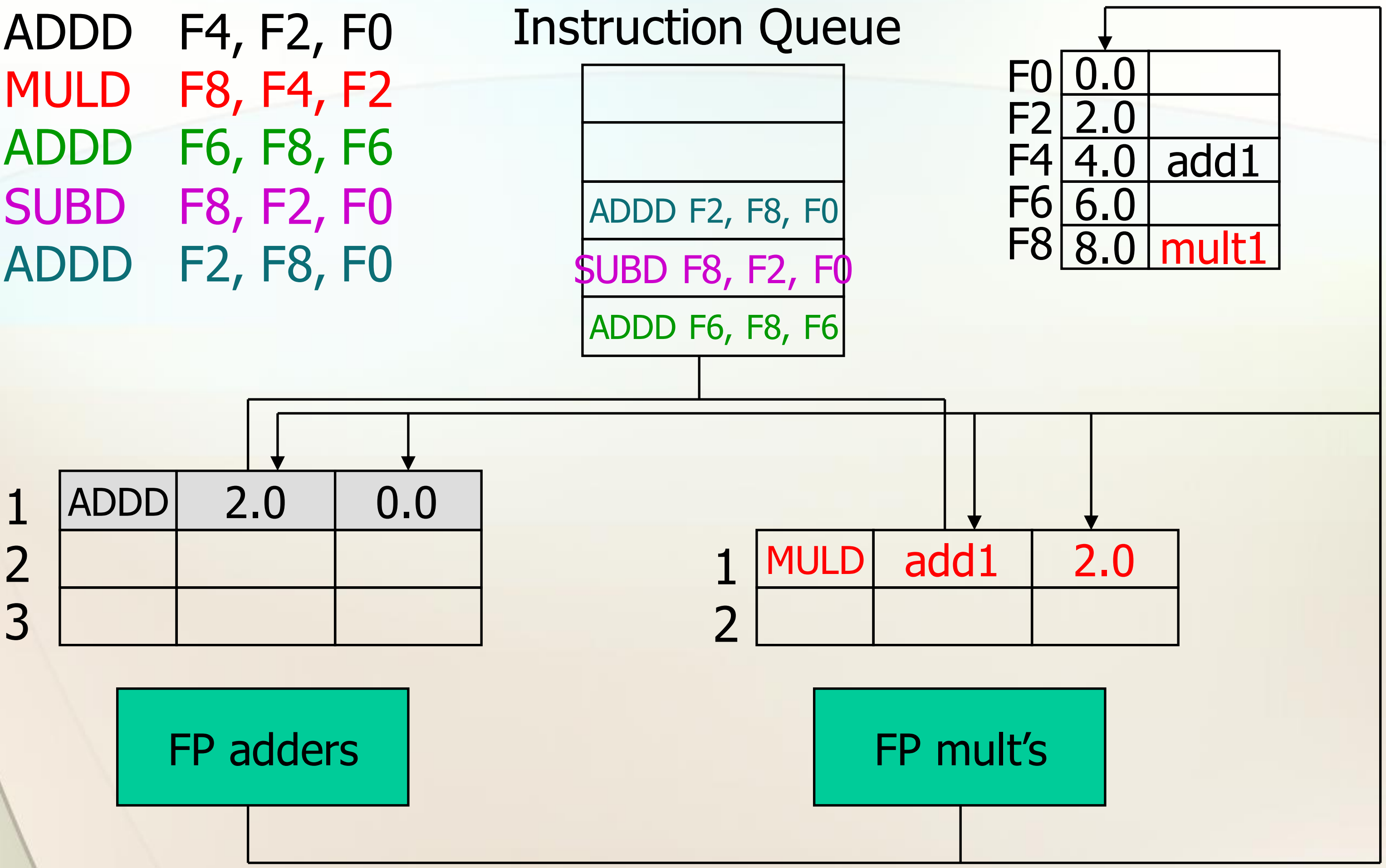
ADDD F4, F2, F0
MULD F8, F4, F2

Op	Qj	Qk	Vj	Vk	Busy
MULD	add1	-	-	2.0	Y



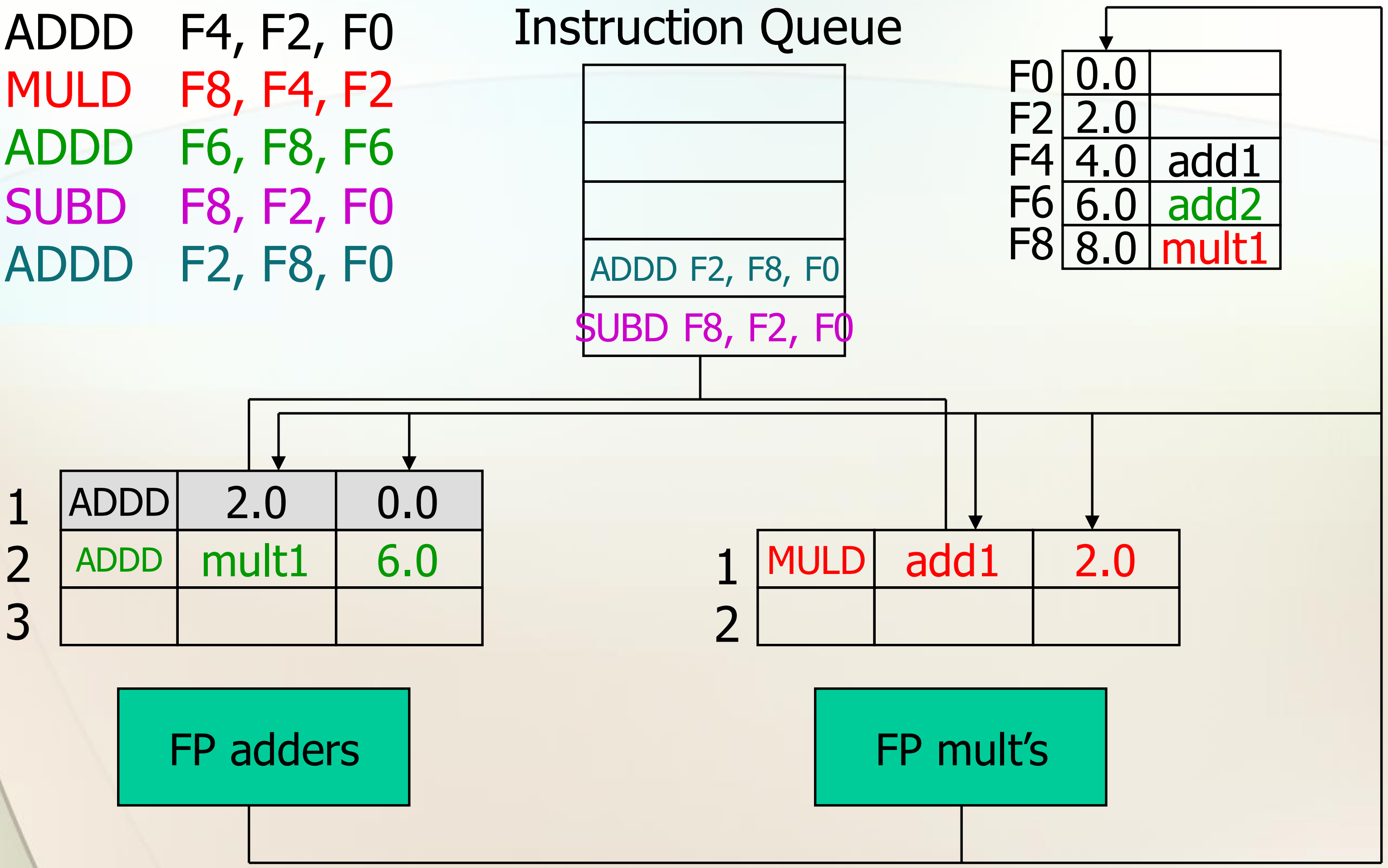
Tomasulo – cycle 2

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Tomasulo – cycle 3

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



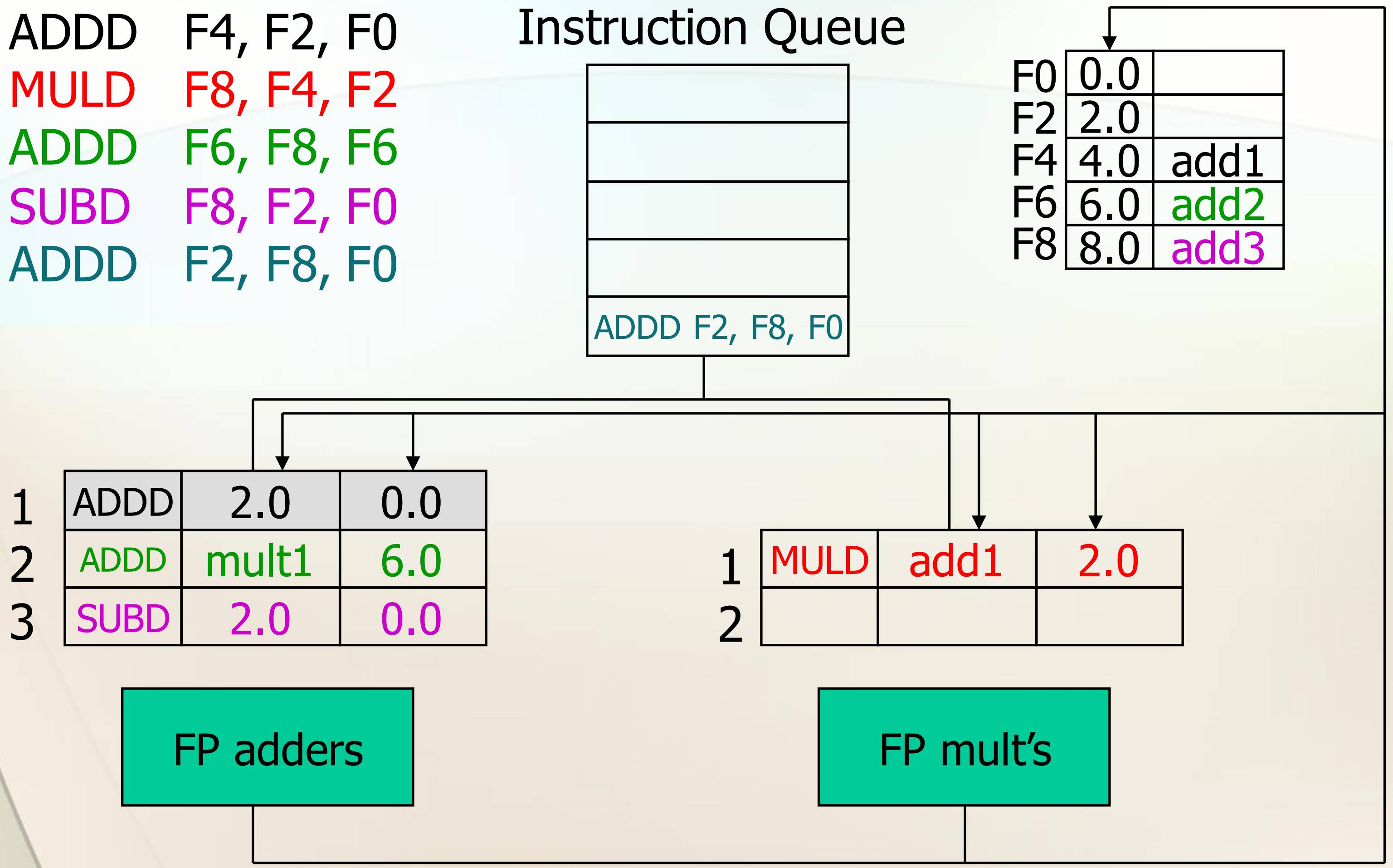
F0	0.0	
F2	2.0	
F4	4.0	add1
F6	6.0	add2
F8	8.0	mult1

1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3			

1	MULD	add1	2.0
2			

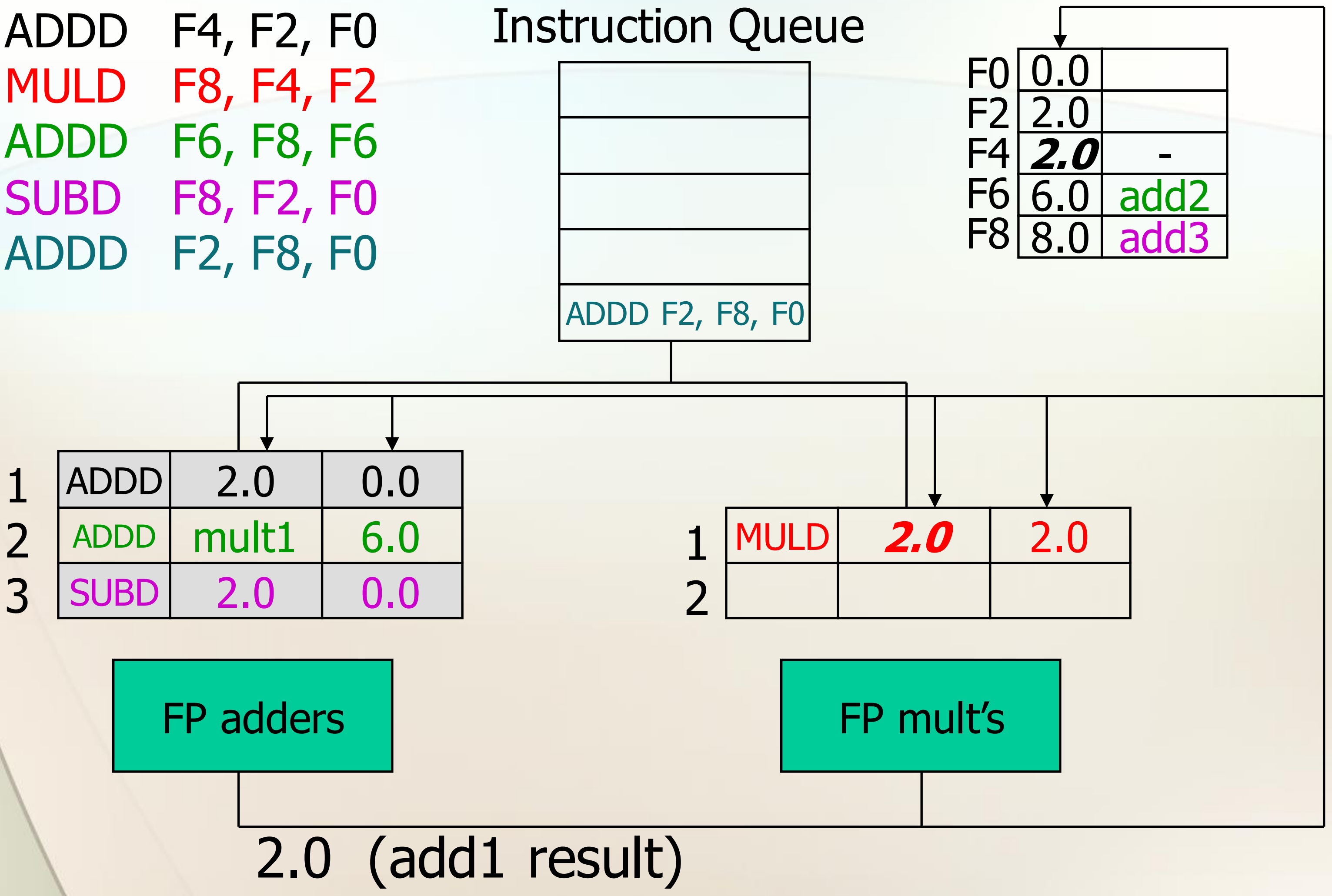
Tomasulo – cycle 4

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



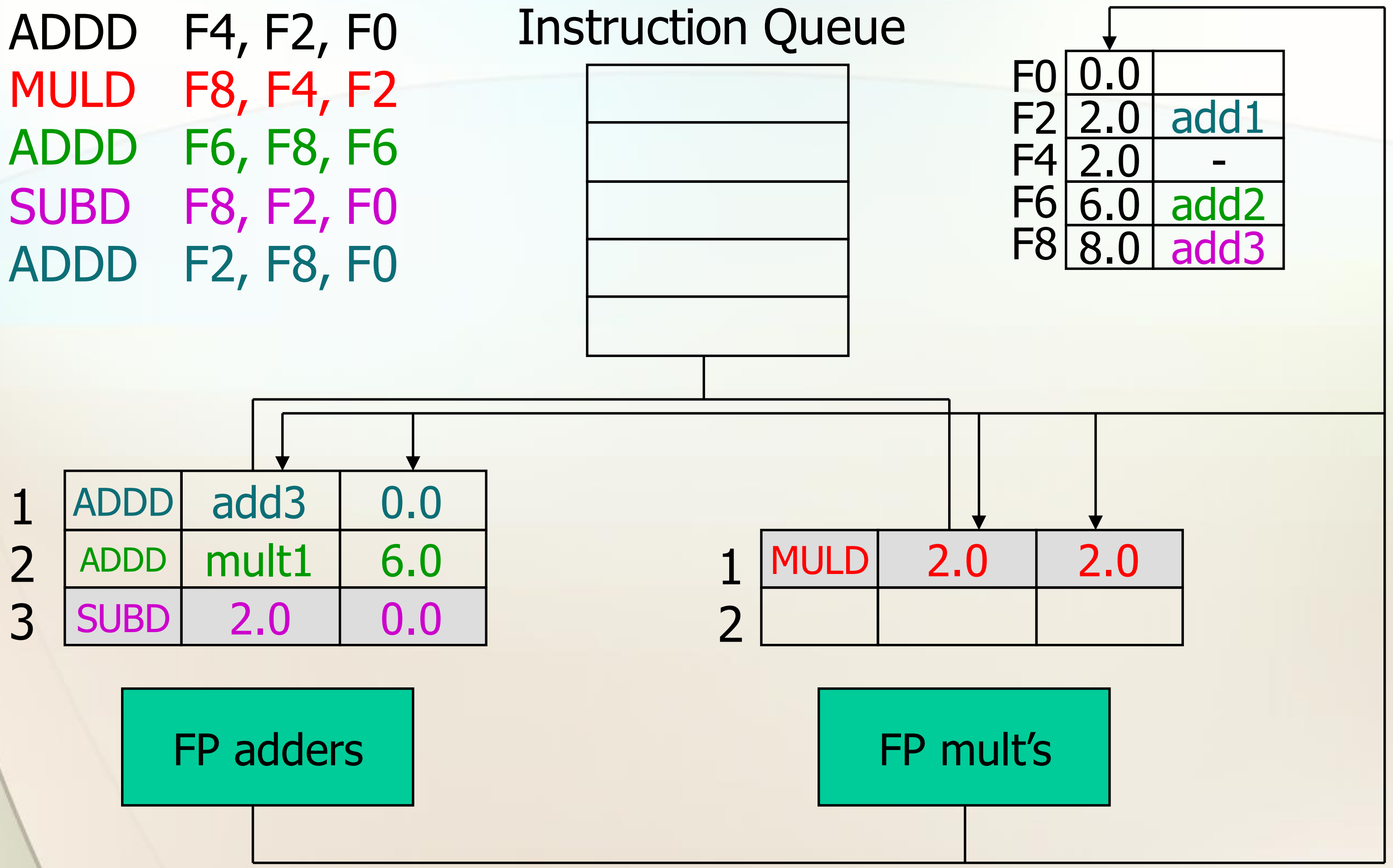
Tomasulo – cycle 5

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



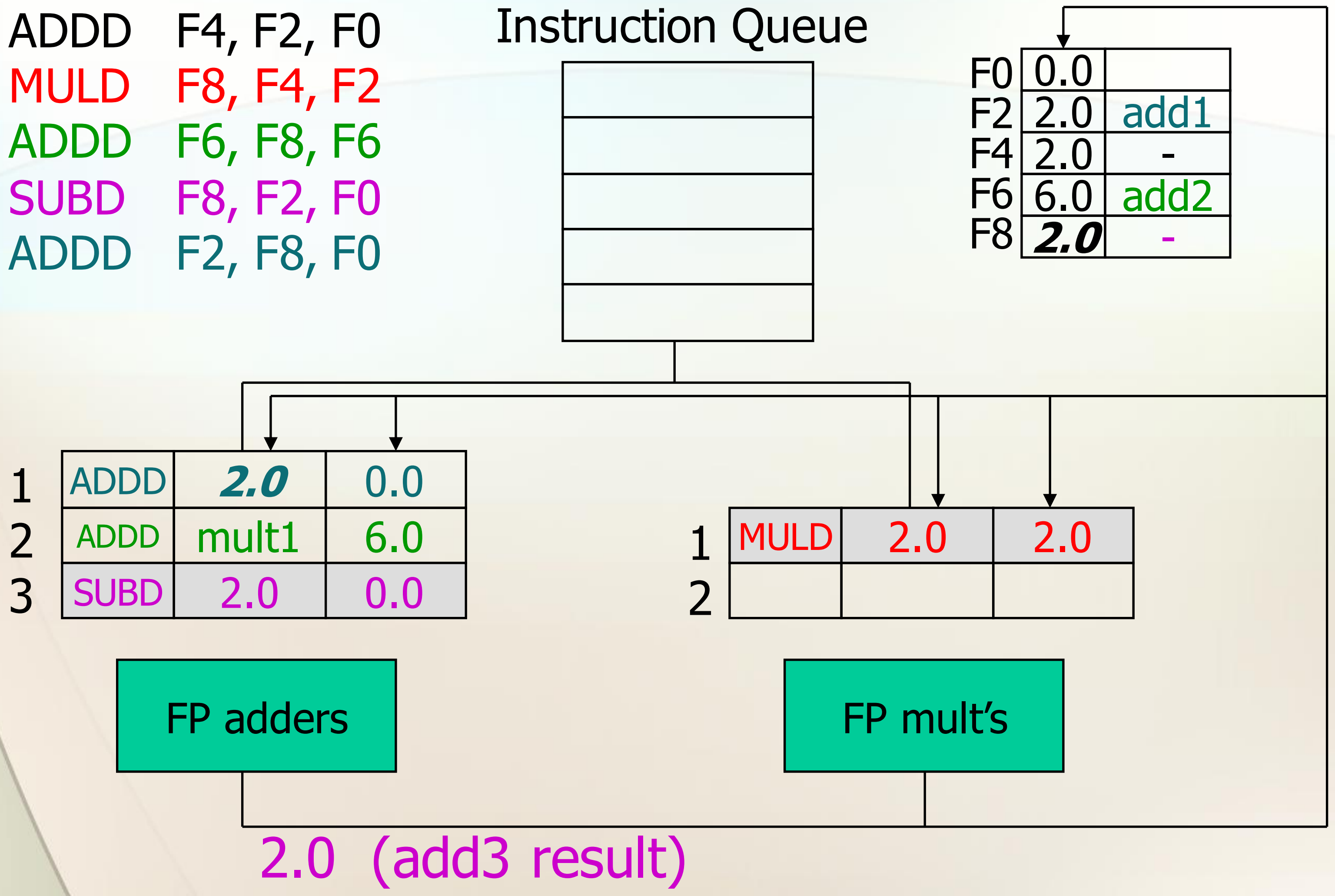
Tomasulo – cycle 6

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



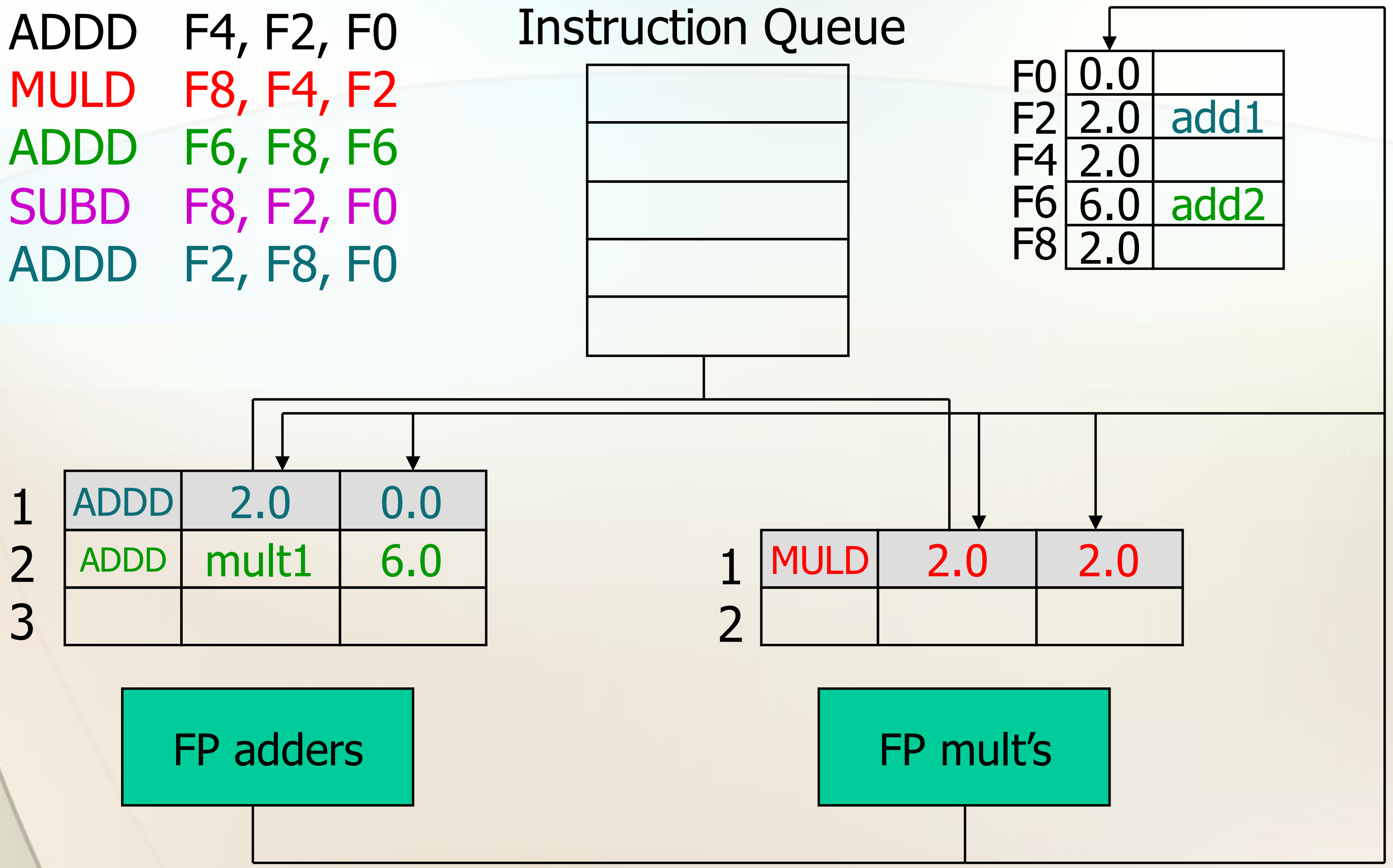
Tomasulo – cycle 8

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



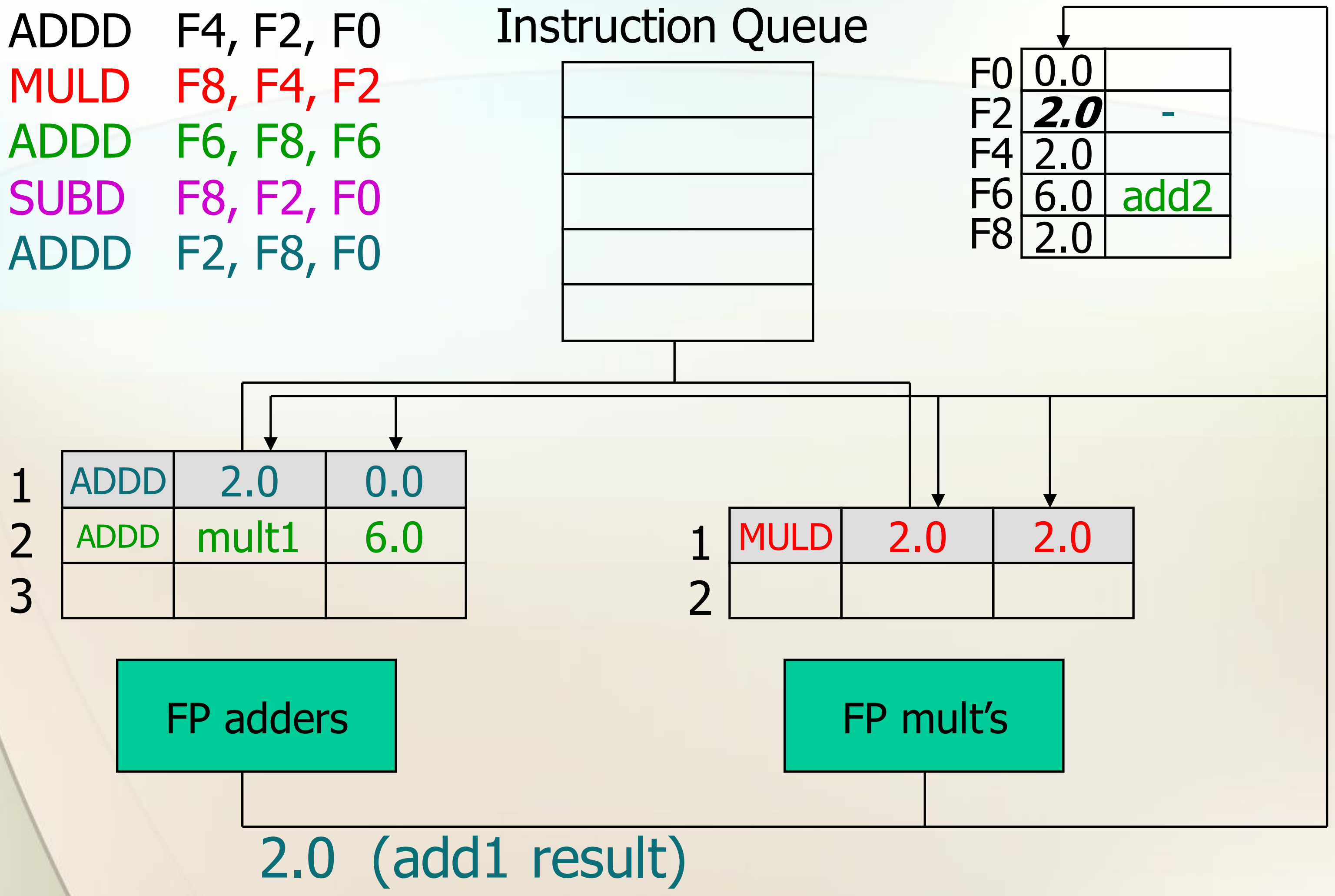
Tomasulo – cycle 9

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



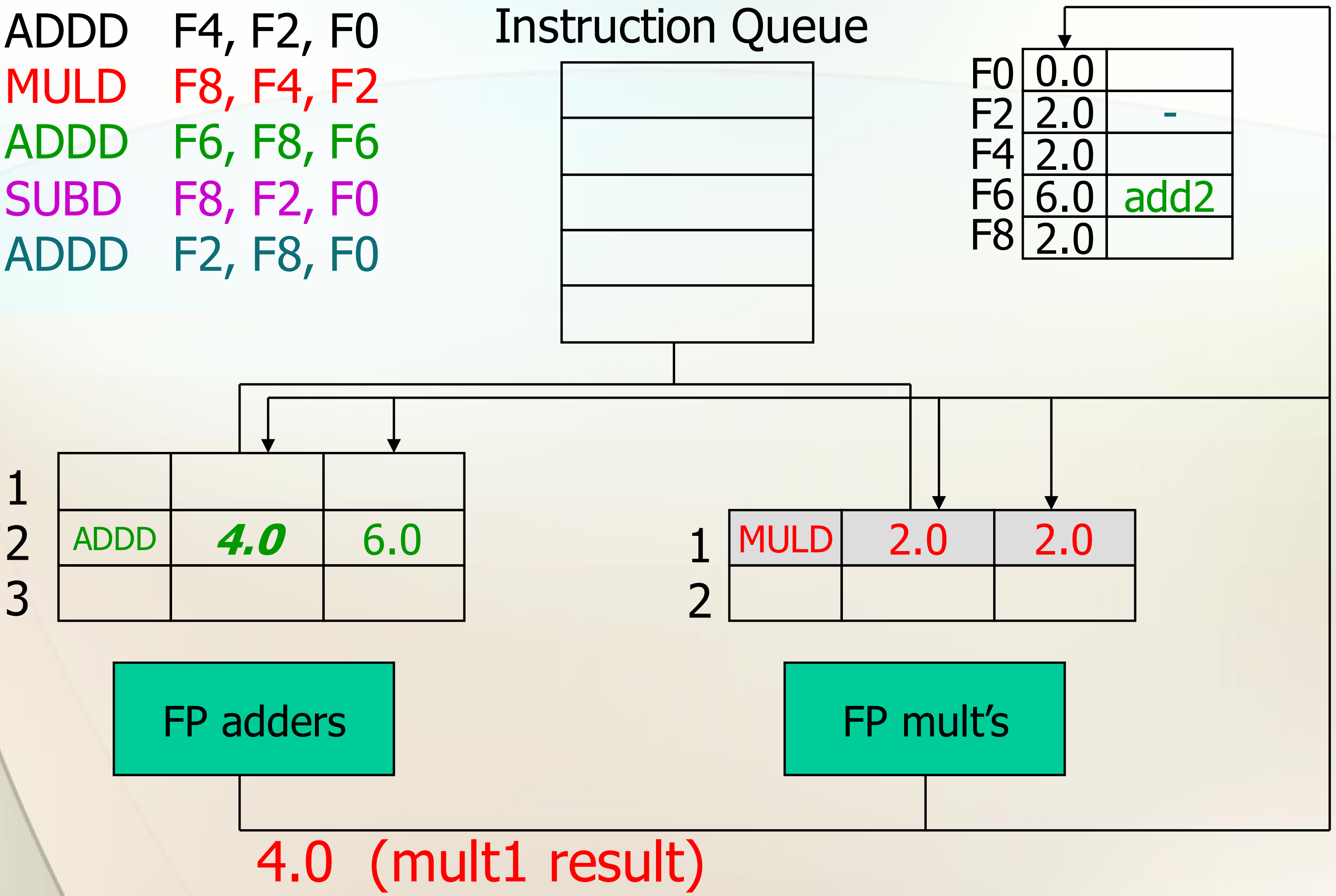
Tomasulo – cycle 12

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



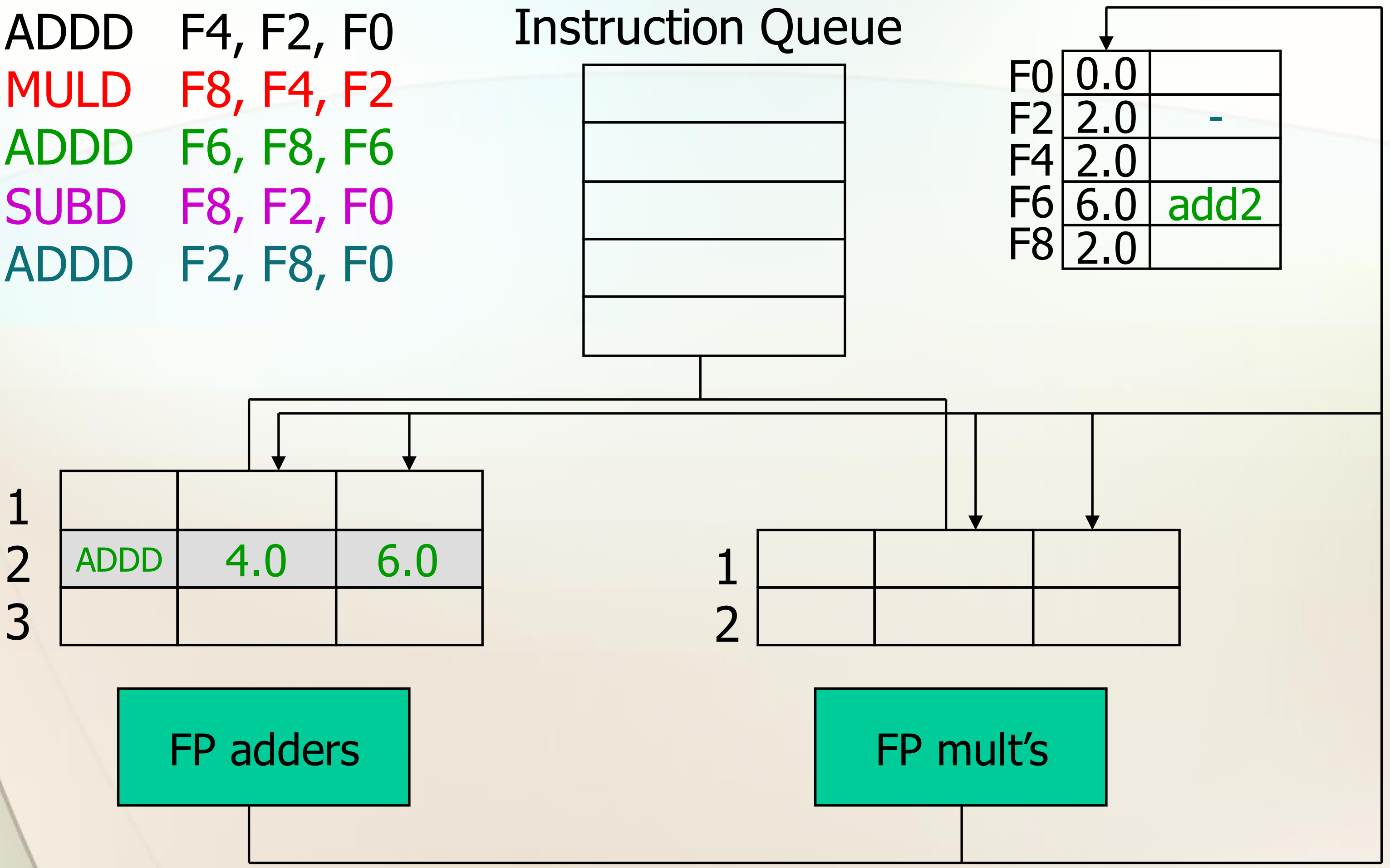
Tomasulo – cycle 15

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



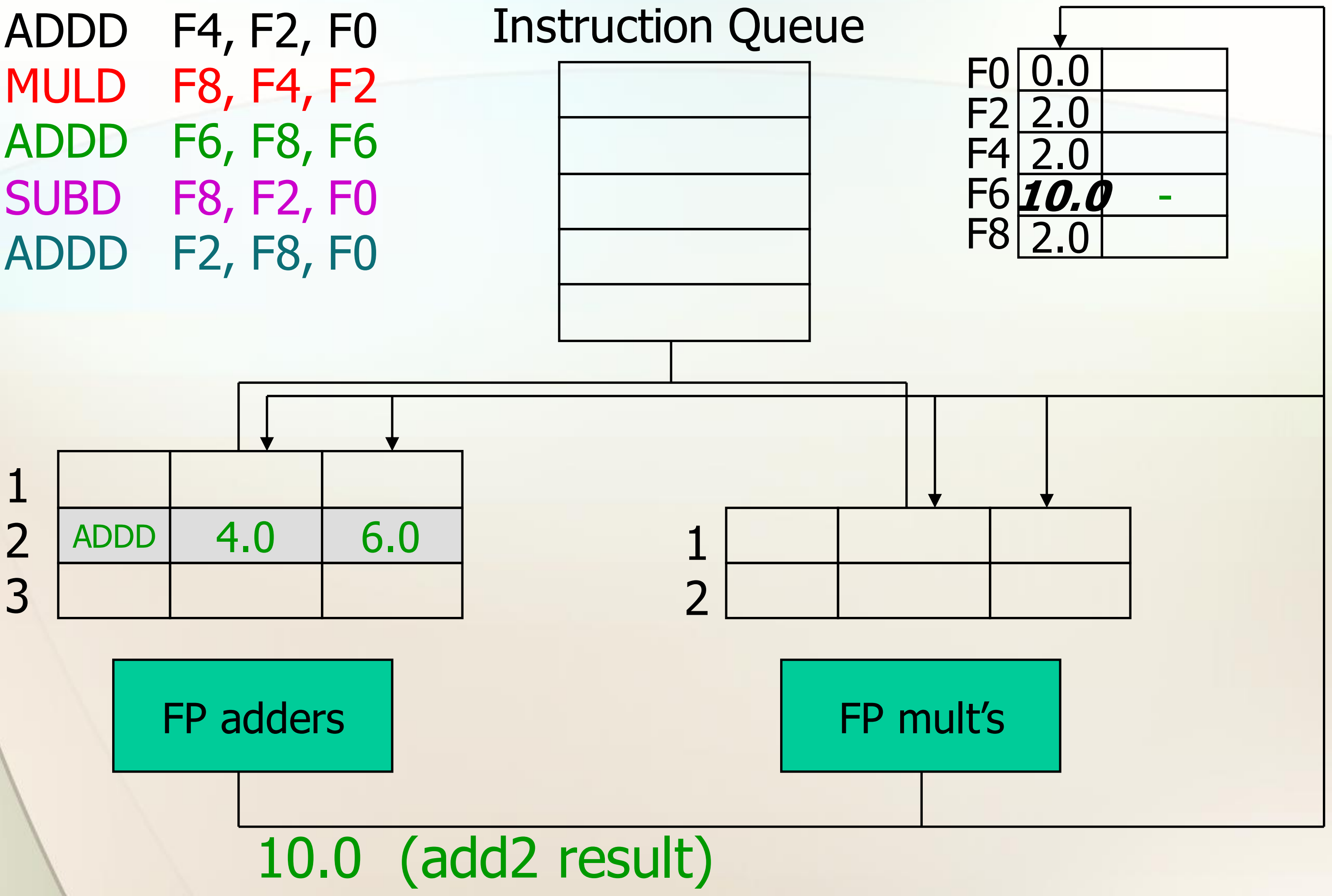
Tomasulo – cycle 16

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Tomasulo – cycle 19

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Tomasulo Summary

- Prevents Register as bottleneck
- Avoids WAR, WAW hazards
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming (in what way does the register name *change*?)
  - Load/store disambiguation

Limitations

- Exceptions/interrupts
  - Can’t identify a particular point in the program at which an interrupt/exception occurs
  - How do you know where to go back to after an interrupt handler completes?
  - OOO completion???
- Interaction with pipelined ALUs
  - Reservation station couldn’t be released until instruction completes, would need many reservation stations.