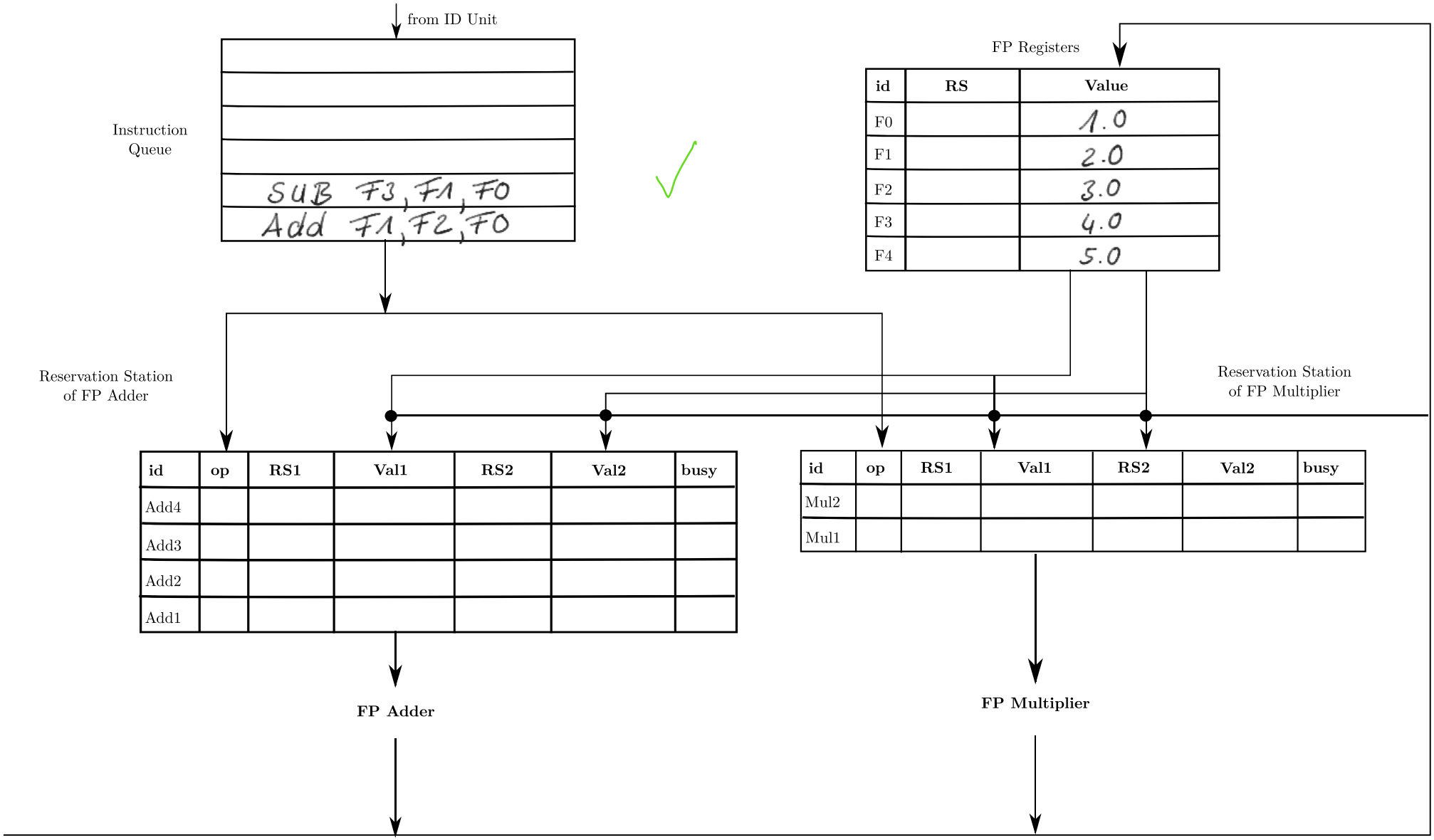
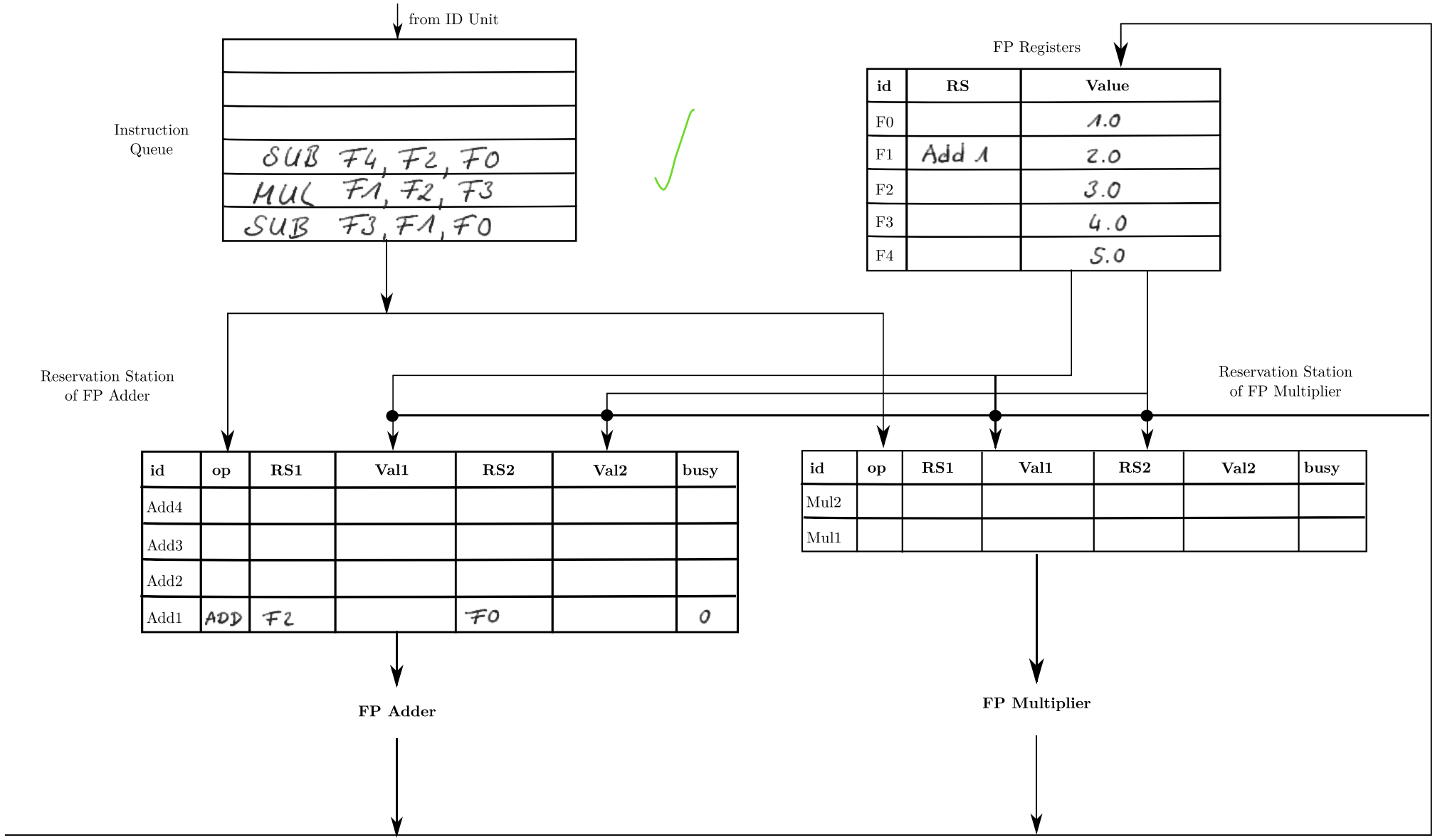
Aufgabe 1: Tomasulos Algorithmus

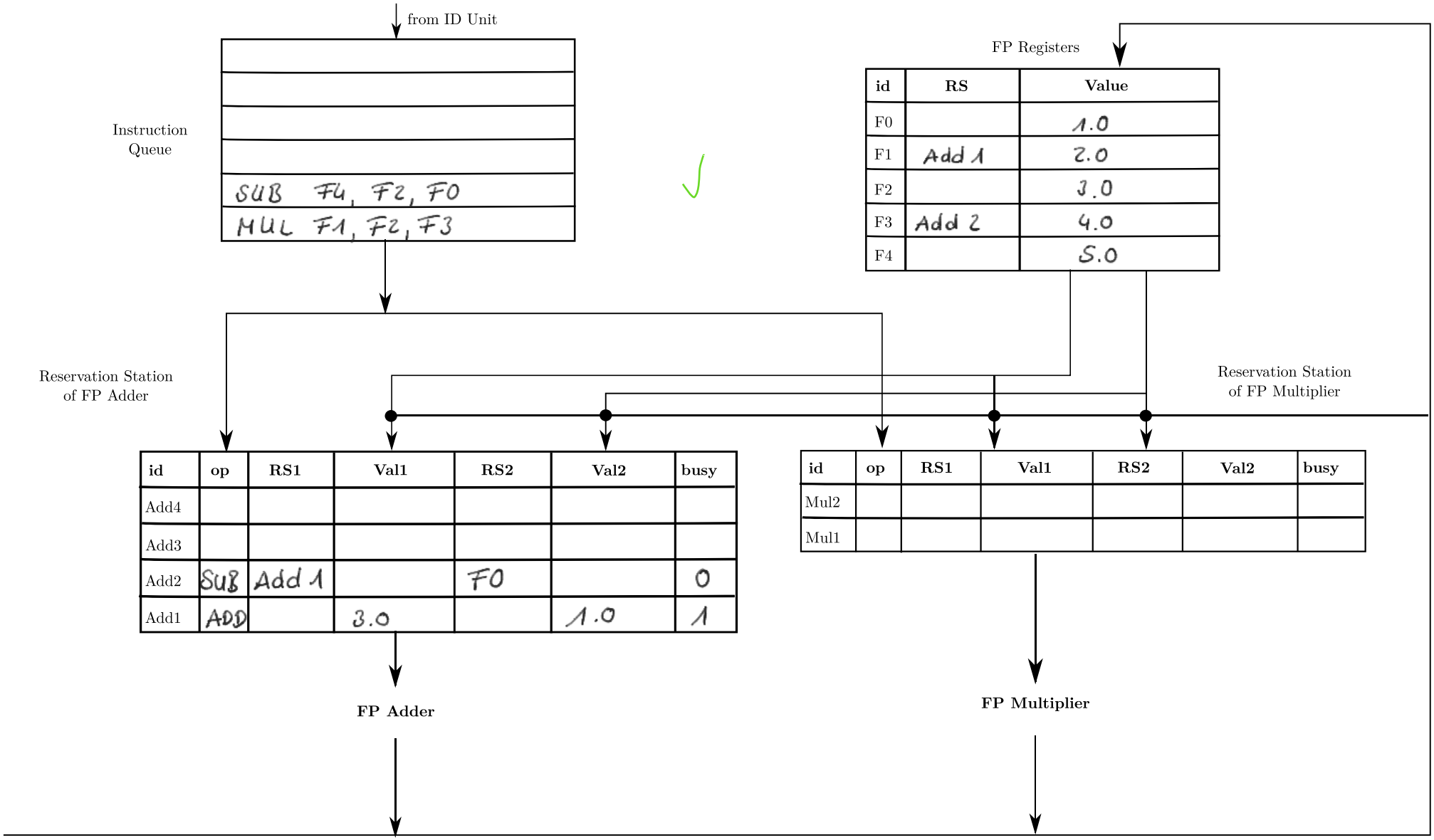
1. Takt


from ID Unit
Instruction Queue
SUB F3, F1, F0
Add F1, F2, F0
FP Registers
id | RS | Value
F0 | 1.0
F1 | 2.0
F2 | 3.0
F3 | 4.0
F4 | 5.0
Reservation Station of FP Adder
id | op | RS1 | Val1 | RS2 | Val2 | busy
Add4
Add3
Add2
Add1
Reservation Station of FP Multiplier
id | op | RS1 | Val1 | RS2 | Val2 | busy
Mul2
Mul1
FP Adder
FP Multiplier

2. Takt


from ID Unit
Instruction Queue
SUB F4, F2, F0
MUL F1, F2, F3
SUB F3, F1, F0
FP Registers
id | RS | Value
F0 | | 1.0
F1 | Add 1 | 2.0
F2 | | 3.0
F3 | | 4.0
F4 | | 5.0
Reservation Station of FP Adder
id | op | RS1 | Val1 | RS2 | Val2 | busy
Add4 | | | | | | |
Add3 | | | | | | |
Add2 | | | | | | |
Add1 | ADD | F2 | | F0 | | 0
Reservation Station of FP Multiplier
id | op | RS1 | Val1 | RS2 | Val2 | busy
Mul2 | | | | | | |
Mul1 | | | | | | |
FP Adder
FP Multiplier

3. Takt

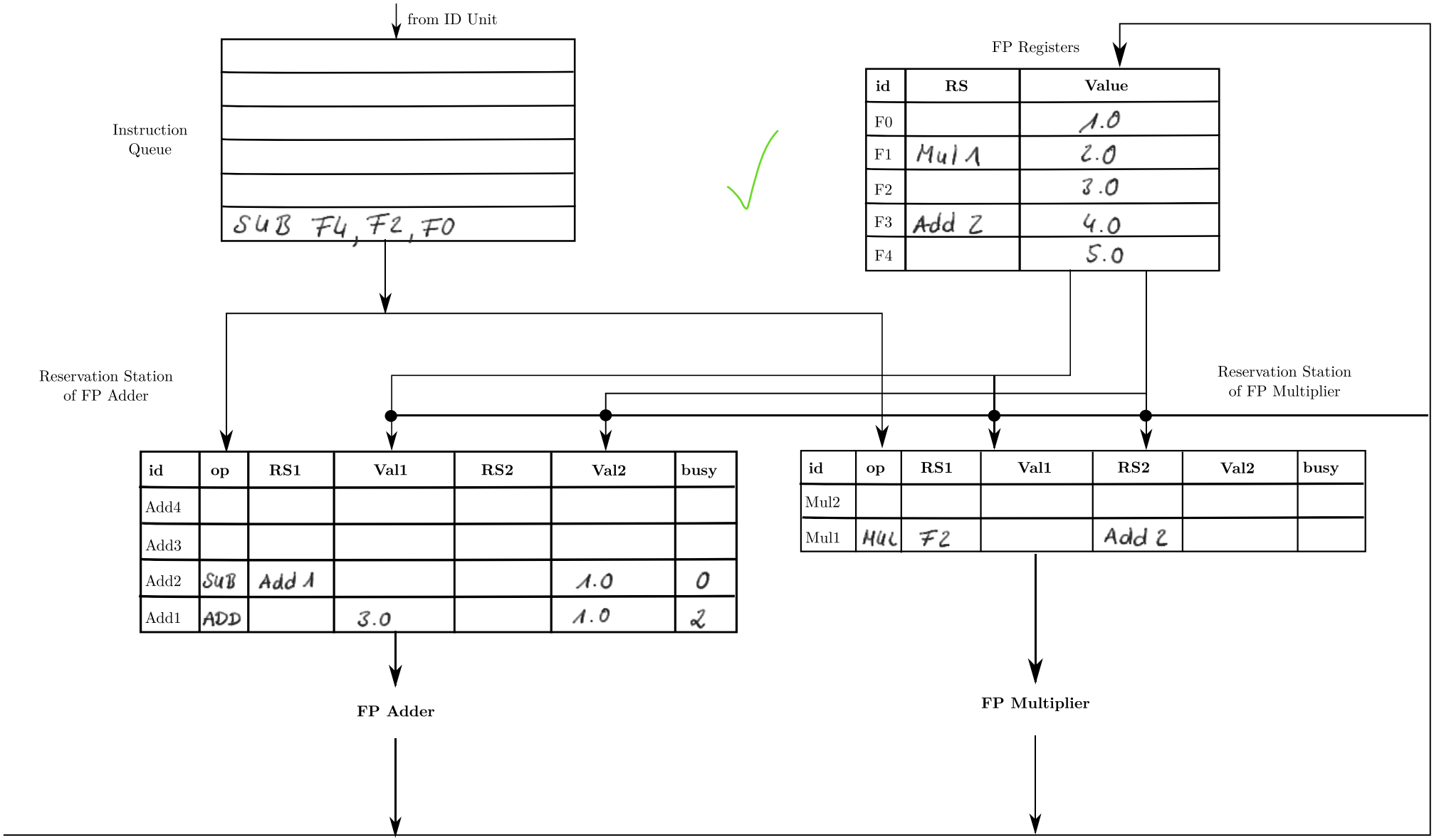


id	RS	Value
F0		1.0
F1	Add 1	2.0
F2		3.0
F3	Add 2	4.0
F4		5.0

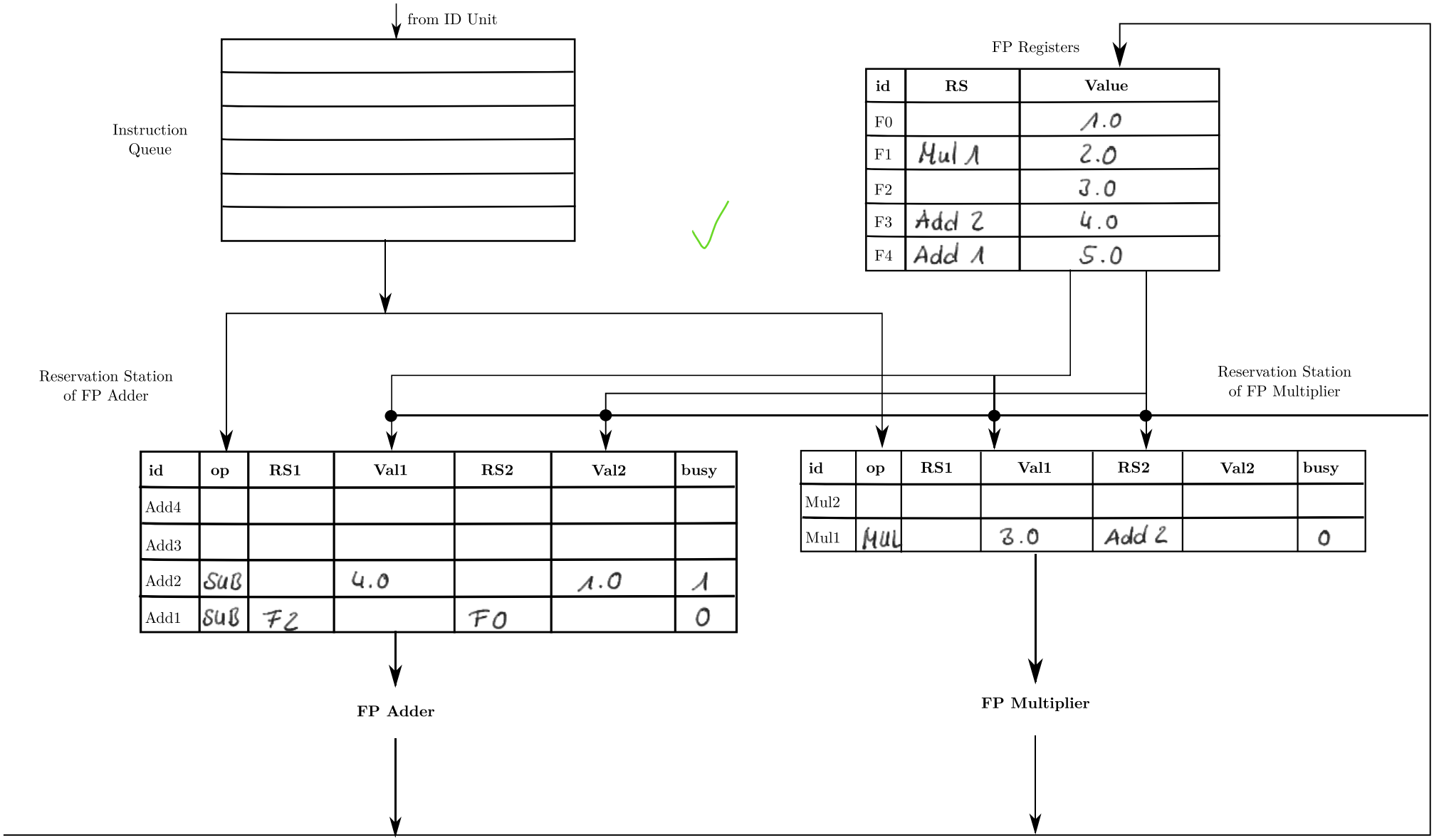
id	op	RS1	Val1	RS2	Val2	busy
Add4						
Add3						
Add2	SUB	Add 1		F0		0
Add1	ADD		3.0		1.0	1

id	op	RS1	Val1	RS2	Val2	busy
Mul2						
Mul1						

4. Takt


from ID Unit
Instruction Queue
SUB F4, F2, F0
FP Registers
id | RS | Value
F0 | | 1.0
F1 | Mul 1 | 2.0
F2 | | 3.0
F3 | Add 2 | 4.0
F4 | | 5.0
Reservation Station of FP Adder
id | op | RS1 | Val1 | RS2 | Val2 | busy
Add4 | | | | | |
Add3 | | | | | |
Add2 | SUB | Add 1 | | | 1.0 | 0
Add1 | ADD | | 3.0 | | 1.0 | 2
Reservation Station of FP Multiplier
id | op | RS1 | Val1 | RS2 | Val2 | busy
Mul2 | | | | | |
Mul1 | MUL | F2 | | Add 2 | |
FP Adder
FP Multiplier

5. Takt



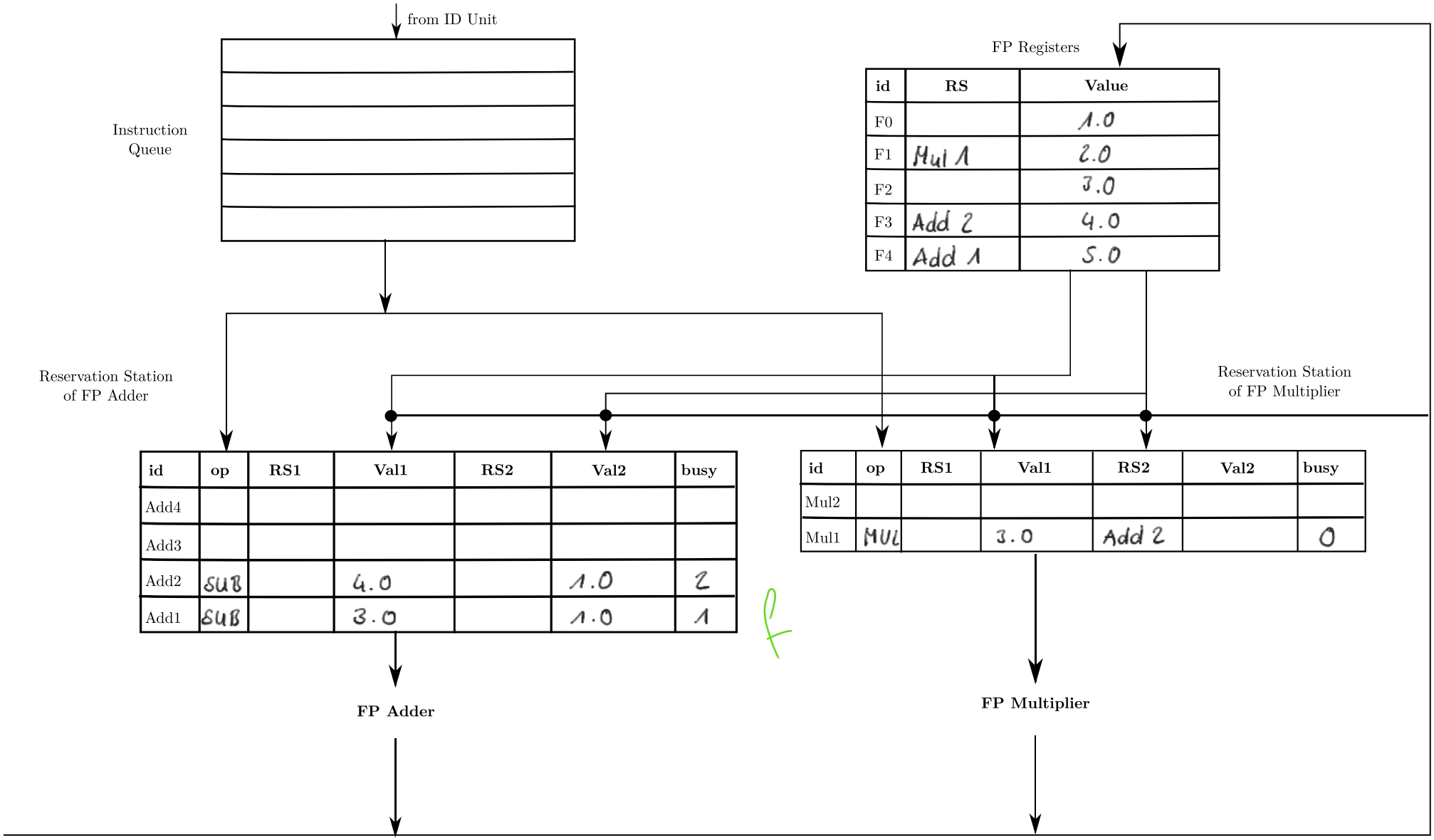
id	RS	Value
F0		1.0
F1	Mul 1	2.0
F2		3.0
F3	Add 2	4.0
F4	Add 1	5.0

id	op	RS1	Val1	RS2	Val2	busy
Add4						
Add3						
Add2	SUB		4.0		1.0	1
Add1	SUB	F2		F0		0

id	op	RS1	Val1	RS2	Val2	busy
Mul2						
Mul1	MUL		3.0	Add 2		0

6. Takt

Da wir nur einen Adder haben kann auch nur eine der Instruktionen gleichzeitig ausgeführt werden; D.h. nur Add2 wird ausgeführt, nicht Add1!



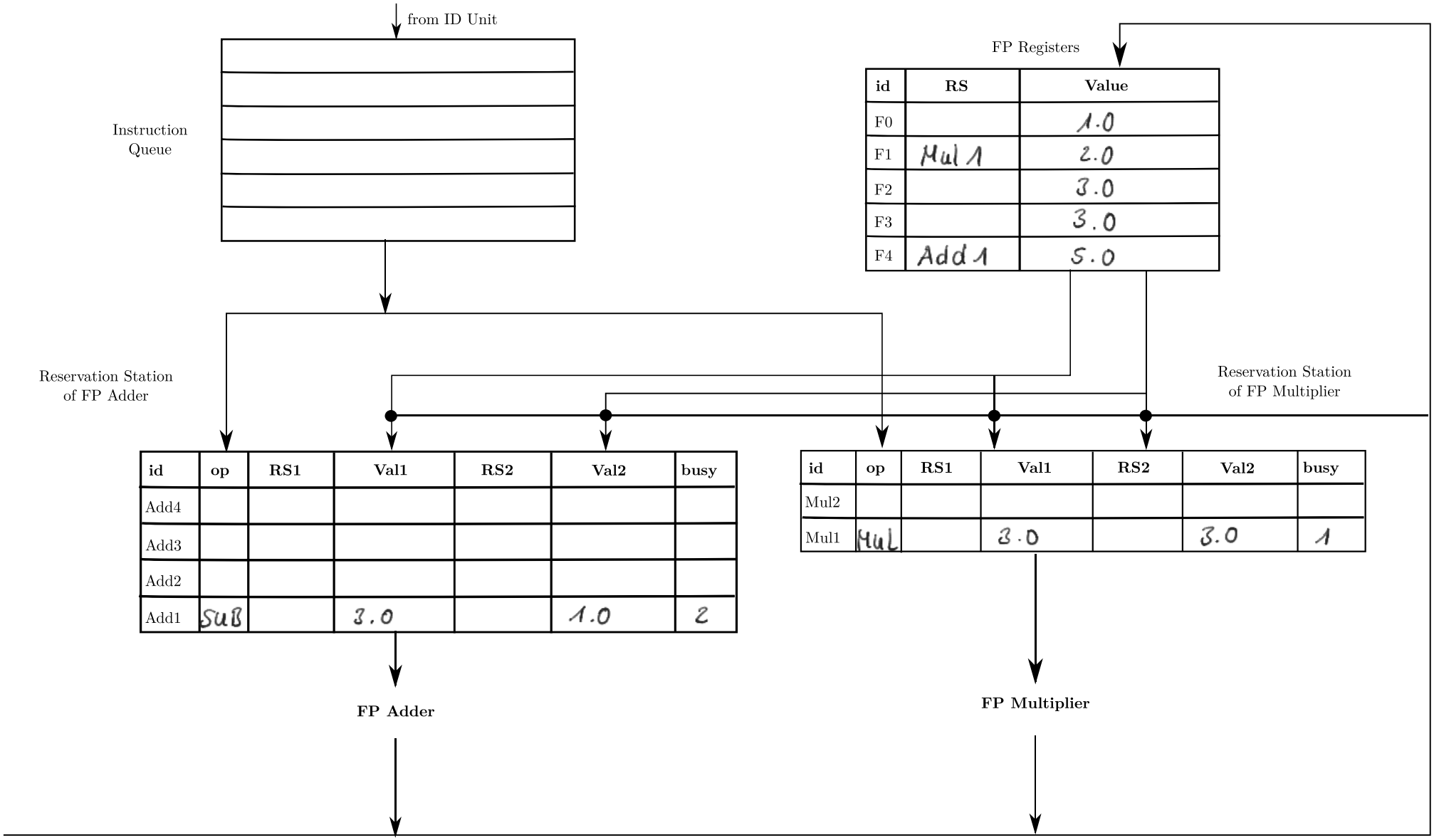
id	RS	Value
F0		1.0
F1	Mul 1	2.0
F2		3.0
F3	Add 2	4.0
F4	Add 1	5.0

id	op	RS1	Val1	RS2	Val2	busy
Add4						
Add3						
Add2	SUB		4.0		1.0	2
Add1	SUB		3.0		1.0	1

id	op	RS1	Val1	RS2	Val2	busy
Mul2						
Mul1	MUL		3.0	Add 2		0

7. Takt

Hier könnte erst Add1 ausgeführt werden!

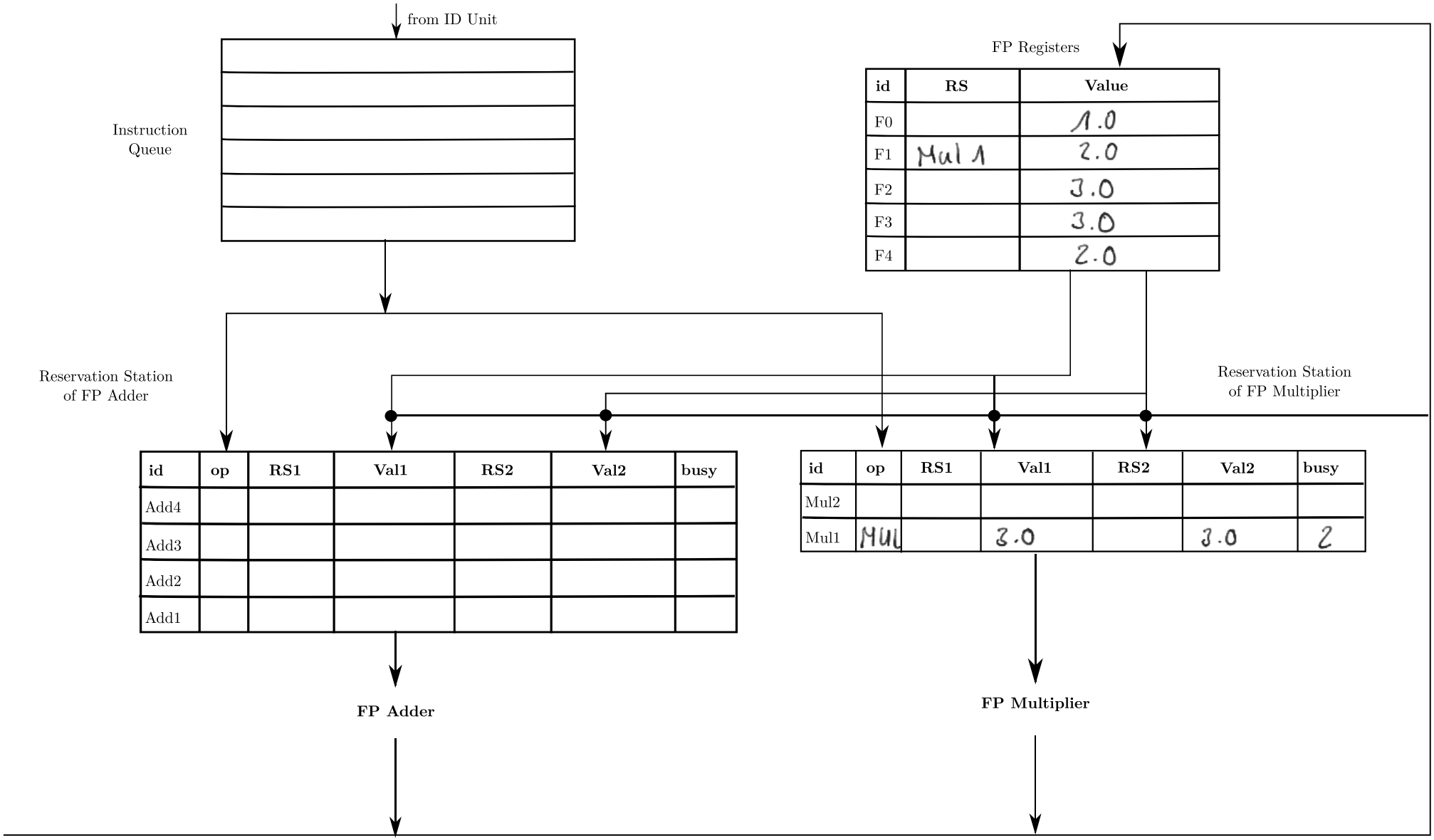


id	RS	Value
F0		1.0
F1	Mul 1	2.0
F2		3.0
F3		3.0
F4	Add 1	5.0

id	op	RS1	Val1	RS2	Val2	busy
Add4						
Add3						
Add2						
Add1	SUB		3.0		1.0	2

id	op	RS1	Val1	RS2	Val2	busy
Mul2						
Mul1	MUL		3.0		3.0	1

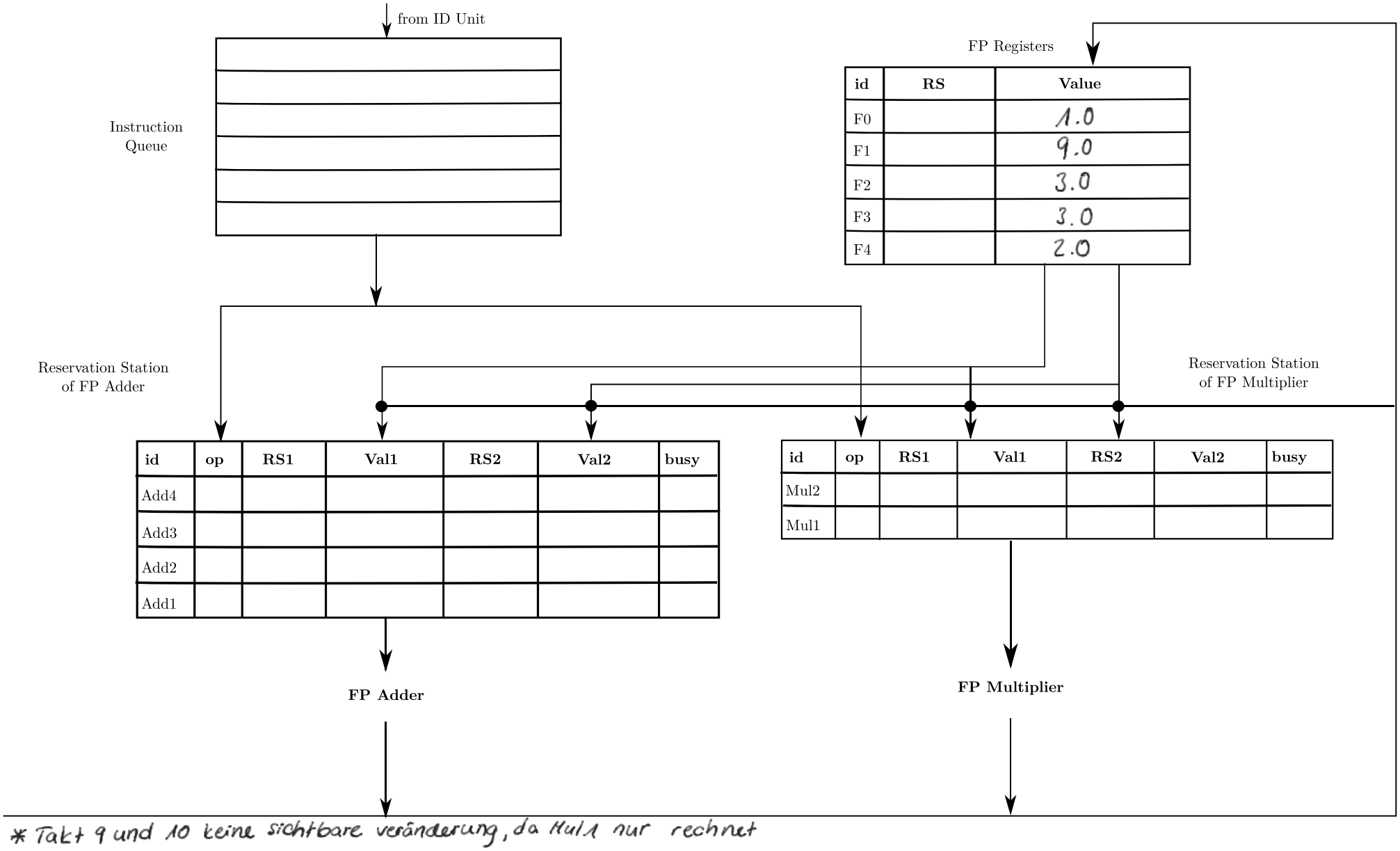
8. Takt


from ID Unit
Instruction Queue
FP Registers
id | RS | Value
F0 | | 1.0
F1 | Mul 1 | 2.0
F2 | | 3.0
F3 | | 3.0
F4 | | 2.0
Reservation Station of FP Adder
Reservation Station of FP Multiplier
id | op | RS1 | Val1 | RS2 | Val2 | busy
Add4
Add3
Add2
Add1
id | op | RS1 | Val1 | RS2 | Val2 | busy
Mul2
Mul1 | MUL | | 3.0 | | 3.0 | 2
FP Adder
FP Multiplier

11. Takt *

b) Es sind 11 Takte nötig, bis die Instruktionsfolge vollständig abgearbeitet ist.

Ok

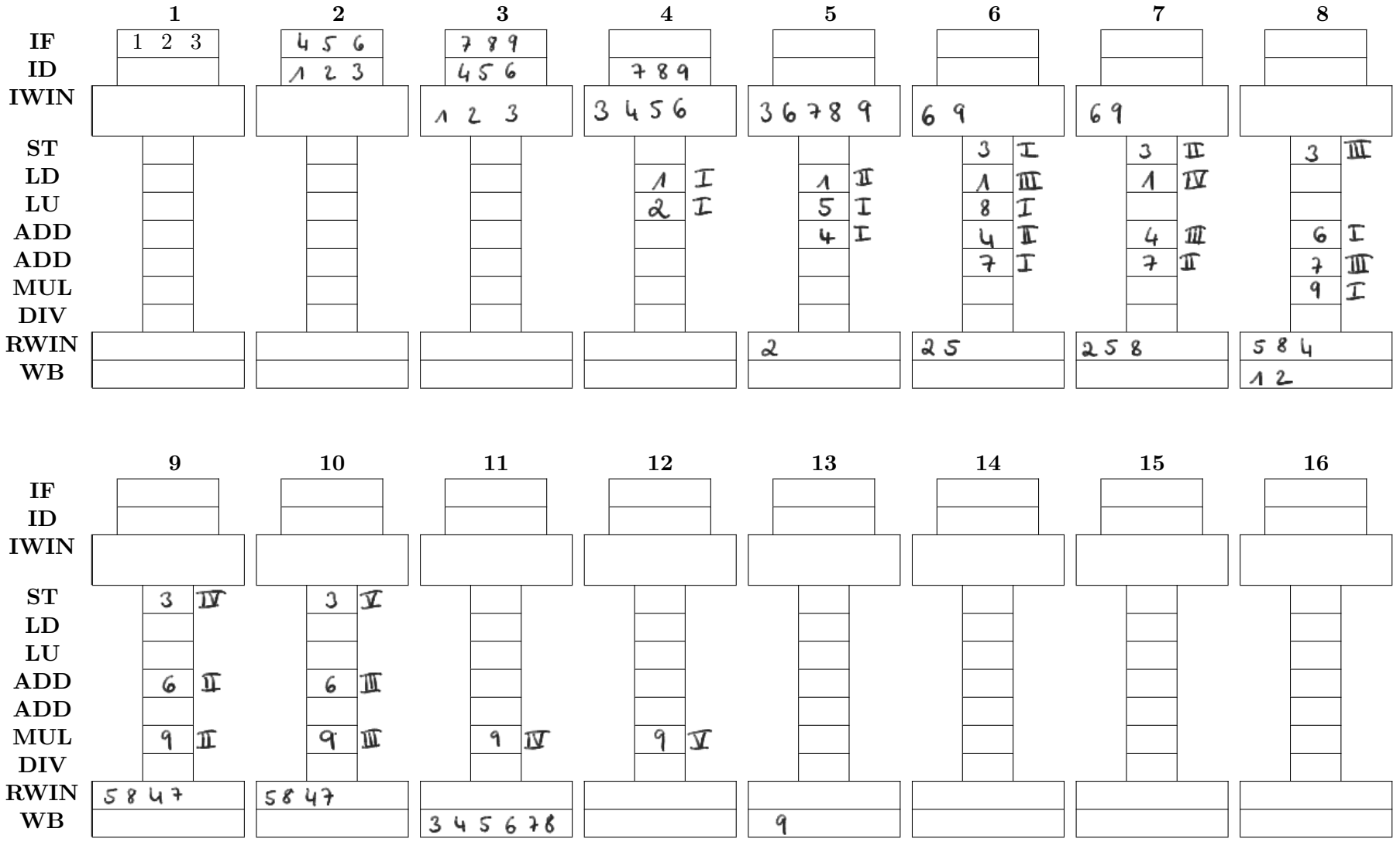


* Takt 9 und 10 keine sichtbare veränderung, da Mul1 nur rechnet

Leider ein kleiner Fehler in der Definition der Pipeline

2/3

Aufgabe 2: Superskalare Pipeline



Musterlösung xD

Superskalare Pipeline (Theoriefragen)

2. Welchen Speed-up haben Sie mit Einführung dieser Pipeline gegenüber einer skalaren Pipeline mit nur einer (mächtigen) EXE Einheit erreicht?

26 (EX addiert) + 3 (IF ID OF für 1. Befehl) + 6 (3 wartet auf 2, 6. auf 5. und 9. auf 8) = 35 Takte

Die Superskalare Pipeline brauchte 14 Takte, um die Folge abzuarbeiten, wenn man zählt, bis die 9 die Write Back Phase verlassen hat. Ansonsten sind es 13 Takte.
Die skalare Pipeline braucht 35 Takte.
Dabei gehen wir davon aus, dass extra Takte eingefüllt werden müssen, damit das Operand Fetch des nächsten Befehls die Ergebnisse aus der Write Back Phase des vorherigen Befehls lesen kann, falls eine Abhängigkeit besteht.
Das betrifft folgende Abhängigkeiten: 3 von 2, 6 von 5 und 9 von 8

Speed up bei euch: 35/11 = 3.18

3. Warum ist Ihre Pipeline nicht optimal ausgelastet? Welche weiteren Maßnahmen führen zu einer (signifikant) größeren Beschleunigung?

Eine mögliche Optimierung wäre die Verwendung von forwarding, damit voneinander abhängige Befehle nicht extra warten müssen.
Befehl 9 (Multiplikation) braucht am längsten und zögert den Abschluss des Codes hinaus. Die Befehle 8 und 9 sind unabhängig von den anderen.
Durch andere Anordnung der Befehle könnte man die Abarbeitungszeit verkürzen, wenn die benötigten Rechenwerke zu der Zeit frei sind.

Ein Forwarding anders als RWIN?

Reordering ist auf jeden Fall richtig

3/3