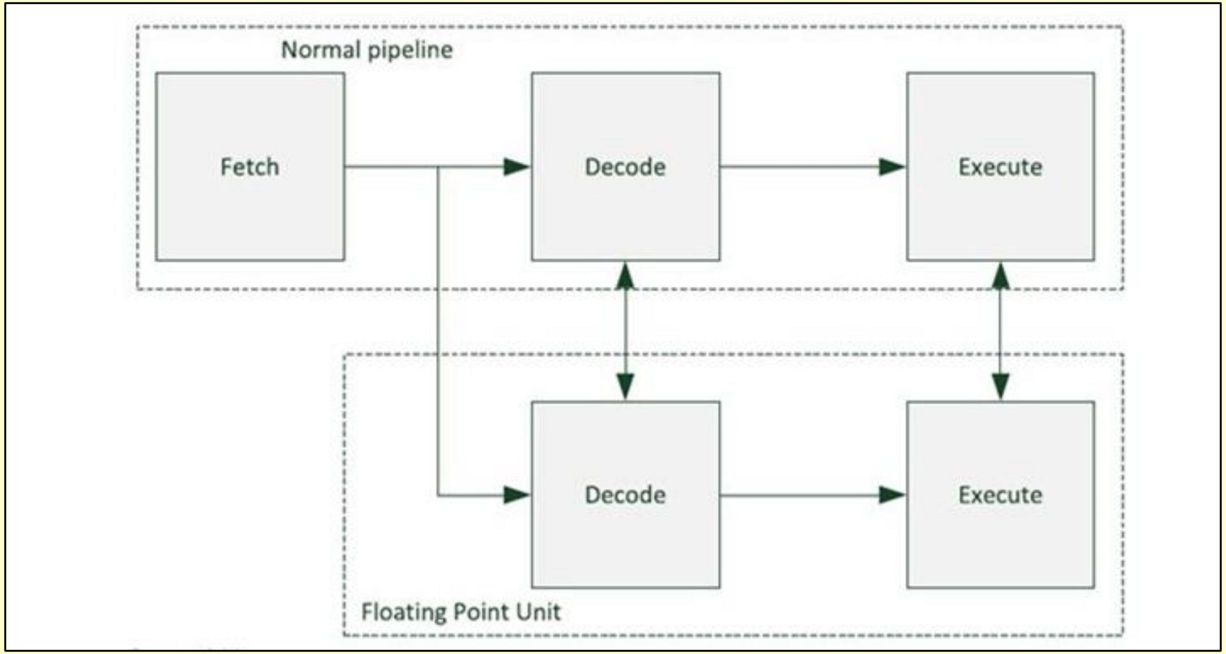
CS-4515
Tomasulo’s Algorithm

Thomas Koker, Christian Tweed,
Eric Arthur, Przemek Gardias,
Milap Patel, Thomas Gibbia, Tessa Garbely

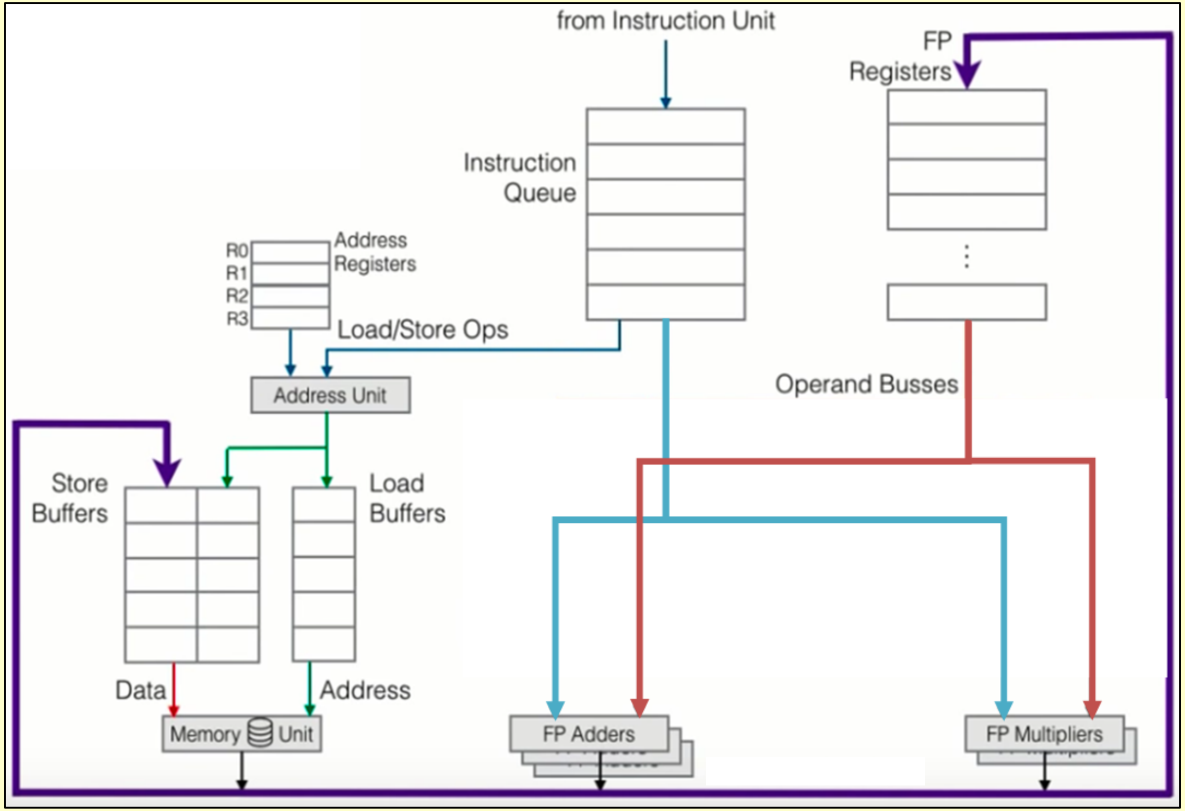
Background

- Tomasulo’s algorithm created by Robert Tomasulo in 1967
- Created to make IBM 360 series computer systems faster
  - Allowing out of order execution of instructions
  - Originally accomplished by complex system specific compilers
- Tomasulo’s algorithm allowed for same benefits on hardware level without need for specific compilers

Original Architecture Setup



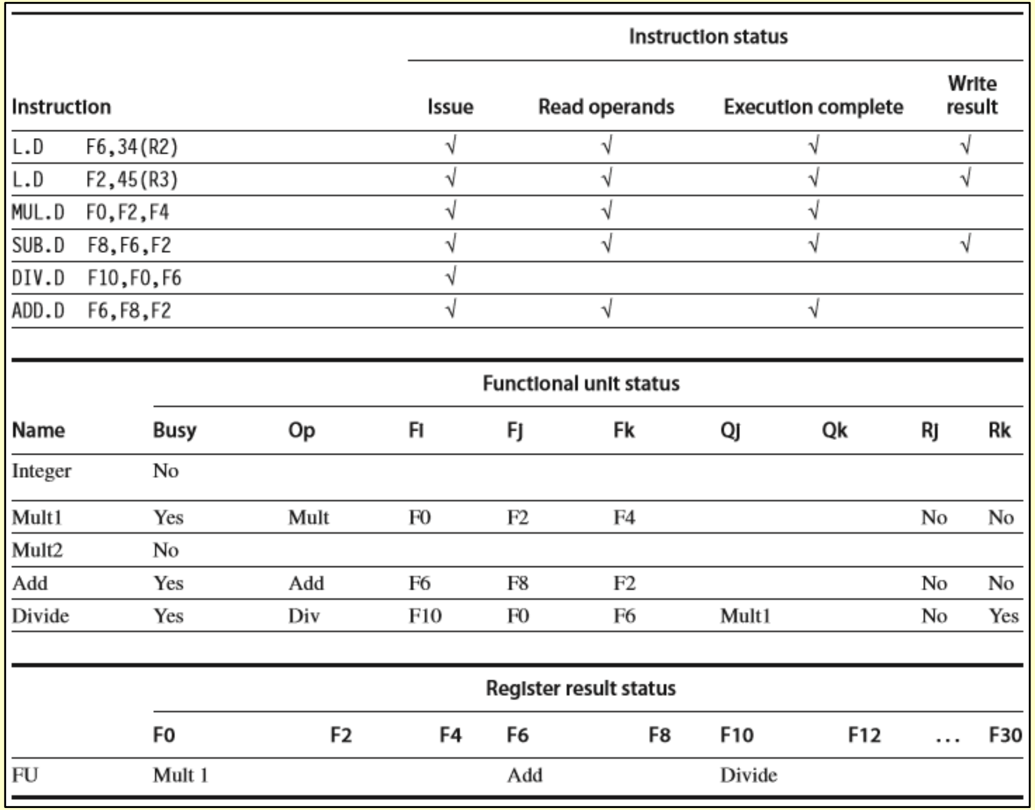
Original Architecture Setup



Tomasulo’s Algorithm

Scoreboard

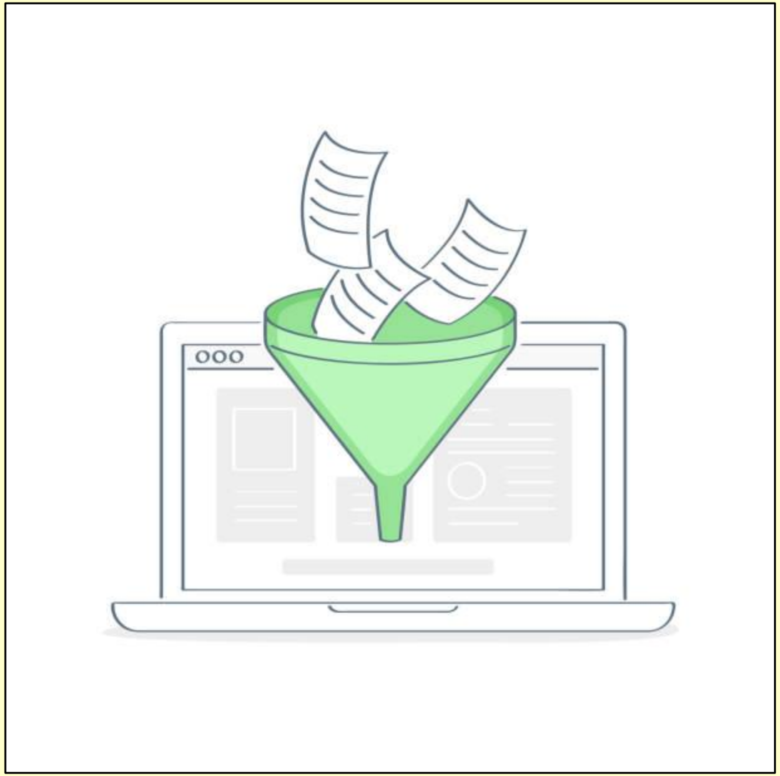
Score Board



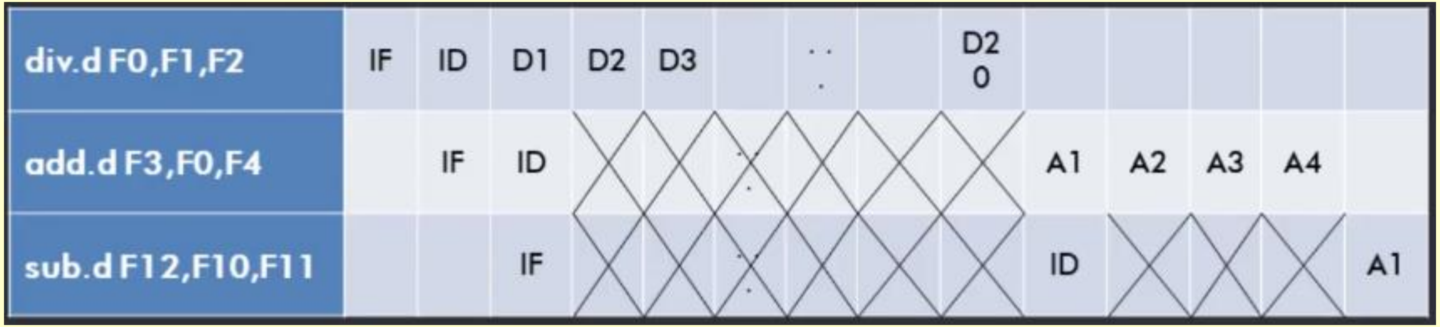
	Instruction status			
Instruction	Issue	Read operands	Execution complete	Write result
L.D F6,34(R2)	√	√	√	√
L.D F2,45(R3)	√	√	√	√
MUL.D F0,F2,F4	√	√	√	
SUB.D F8,F6,F2	√	√	√	√
DIV.D F10,F0,F6	√			
ADD.D F6,F8,F2	√	√	√	

	Functional unit status								
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult 1			Add		Divide			



Execution of instruction



div.d F0,F1,F2	IF	ID	D1	D2	D3		∵		D20					
add.d F3,F0,F4		IF	ID	☒	☒	☒ ∵	☒	☒	☒	A1	A2	A3	A4	
sub.d F12,F10,F11			IF	☒	☒	☒ ∵	☒	☒	☒	ID	☒	☒	☒	A1

Tomasulo’s Algorithm

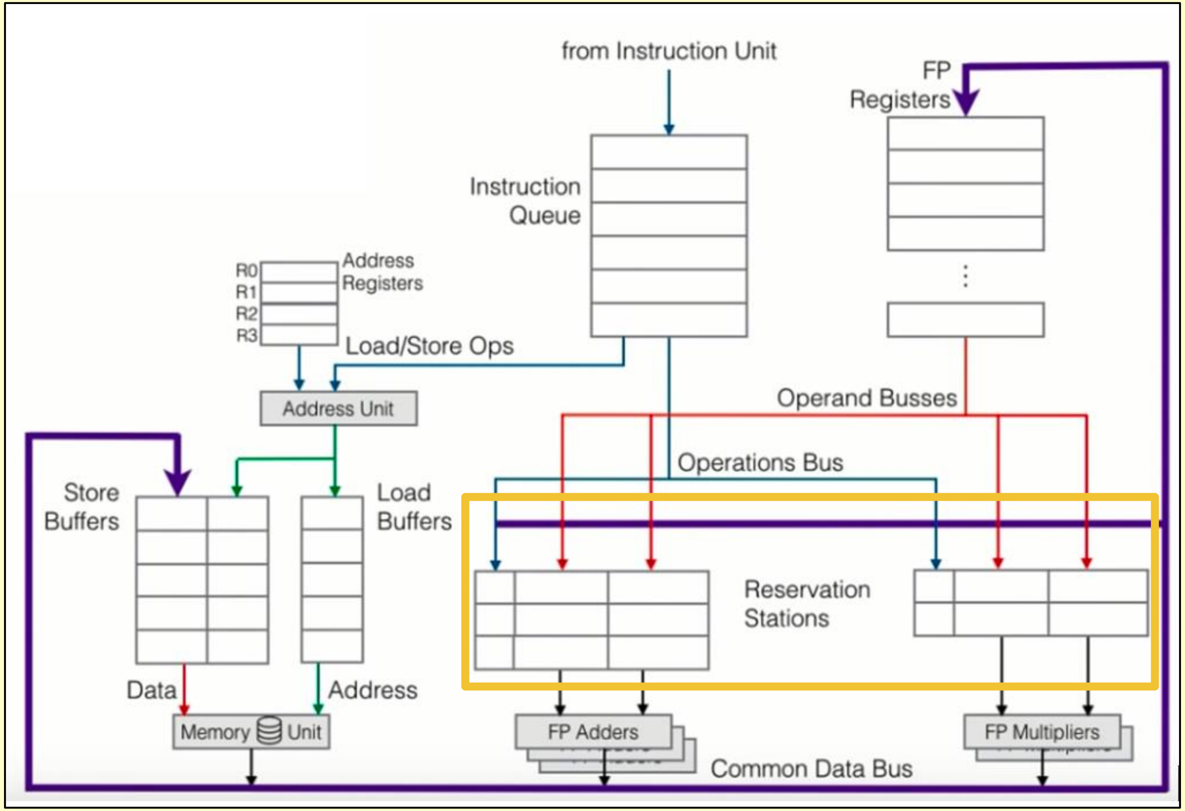
- First implemented in IBM System/360 Model 91’s floating point unit
- Tomasulo’s algorithm allows for out-of-order execution and more efficient use of multiple execution units
- Improvements over scoreboarding:
    - Register renaming
    - Reservation stations for execution units
    - Common Data Bus

Why was Tomasulo’s algorithm developed?

5 major issues before Tomasulo’s algorithm:

- A small number of floating point registers
- Long memory latency (no caching)
- Functional unit were often underutilized.
- Penalties of data dependencies
- Scoreboarding would stall issuing instructions due to hazards

Tomasulo’s Architecture Setup



Tomasulo’s Algorithm

Dependencies

- Data Dependencies
  - Flow Dependencies
- Name Dependencies
  - Anti-dependencies
  - Output Dependencies
- Control Dependencies

Register Renaming

- Two operations use the same register
  - No data flow between them
  - Register as a name
- If there are available registers, rename
  - Change of location removes dependency
  - Can be executed out of order, or in parallel
- Two levels
  - Compiler level
  - Hardware level

Flow Dependencies

- True dependency
  - RAW hazard
- Instruction relies on output of prior

```
Mul    R3, R1, R2
Add     R8, R9, R7
Sub R5, R6, R3
```

- Sub relies on Mul
  - R3 is shared between the two

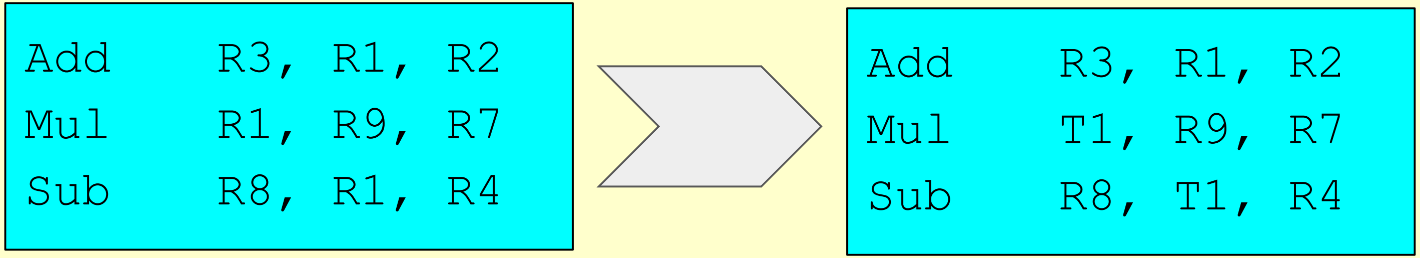
Flow Dependencies - Transitive

- Program order must be preserved
- Other instructions can be executed between the two
- Transitive
  - 3 depends on 2 which depends on 1
  - 3 depends on 1

```
Add     R3, R1, R2
Mul     R8, R3, R7
Sub     R5, R8, R1
```

Anti-Dependence

- False Dependency
  - RAW hazard
- Later instruction writes to a location read by a prior instruction
- Register Renaming



```
Add     R3, R1, R2
Mul     R1, R9, R7
Sub     R8, R1, R4
```

```
Add     R3, R1, R2
Mul     T1, R9, R7
Sub     R8, T1, R4
```

Anti Dependence

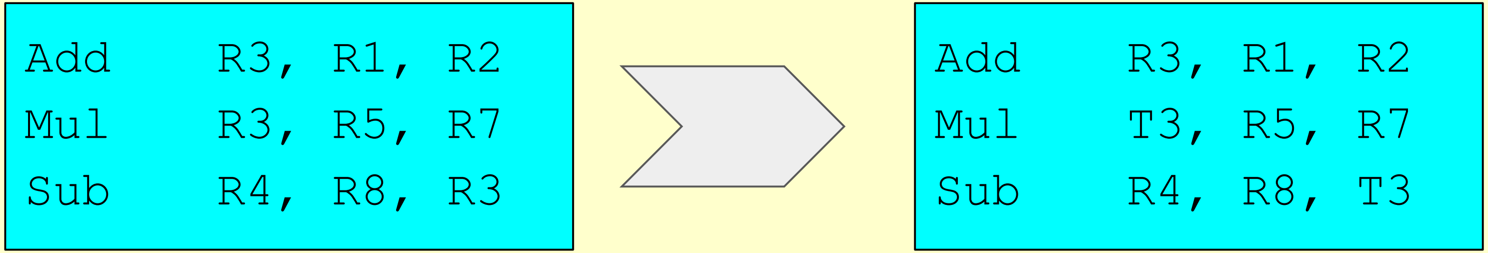
```
1. B = 3
2. A = B + 5
3. B = 7
```

```
1. B = 3
2. B2 = B
3. A = B2 + 5
4. B = 7
```

Introduces flow dependency
Instruction 3 depends on 2, which depends on 1

Output Dependency

- False Dependency
  - WAW hazard
- Later instruction writes to a location written to by earlier instruction
- Ensure that the later write’s result is preserved
  - Later in program order



```
Add     R3, R1, R2
Mul     R3, R5, R7
Sub     R4, R8, R3
```

```
Add     R3, R1, R2
Mul     T3, R5, R7
Sub     R4, R8, T3
```

Questions?

Step 1: Issue

- Get instruction from queue
- Send to reservation station if available
- Stall if no reservation station available
- If operands are not in registers, keep track of which functional units will produce them

Step 2: Execute

- Wait for operands to be computed
  - And sent to CDB
- Execute instruction

Step 3: Write result

- Write result to CDB
- From there it goes to registers and, if necessary, reservation stations
  - If reservation stations need the value
- If a store, write to memory

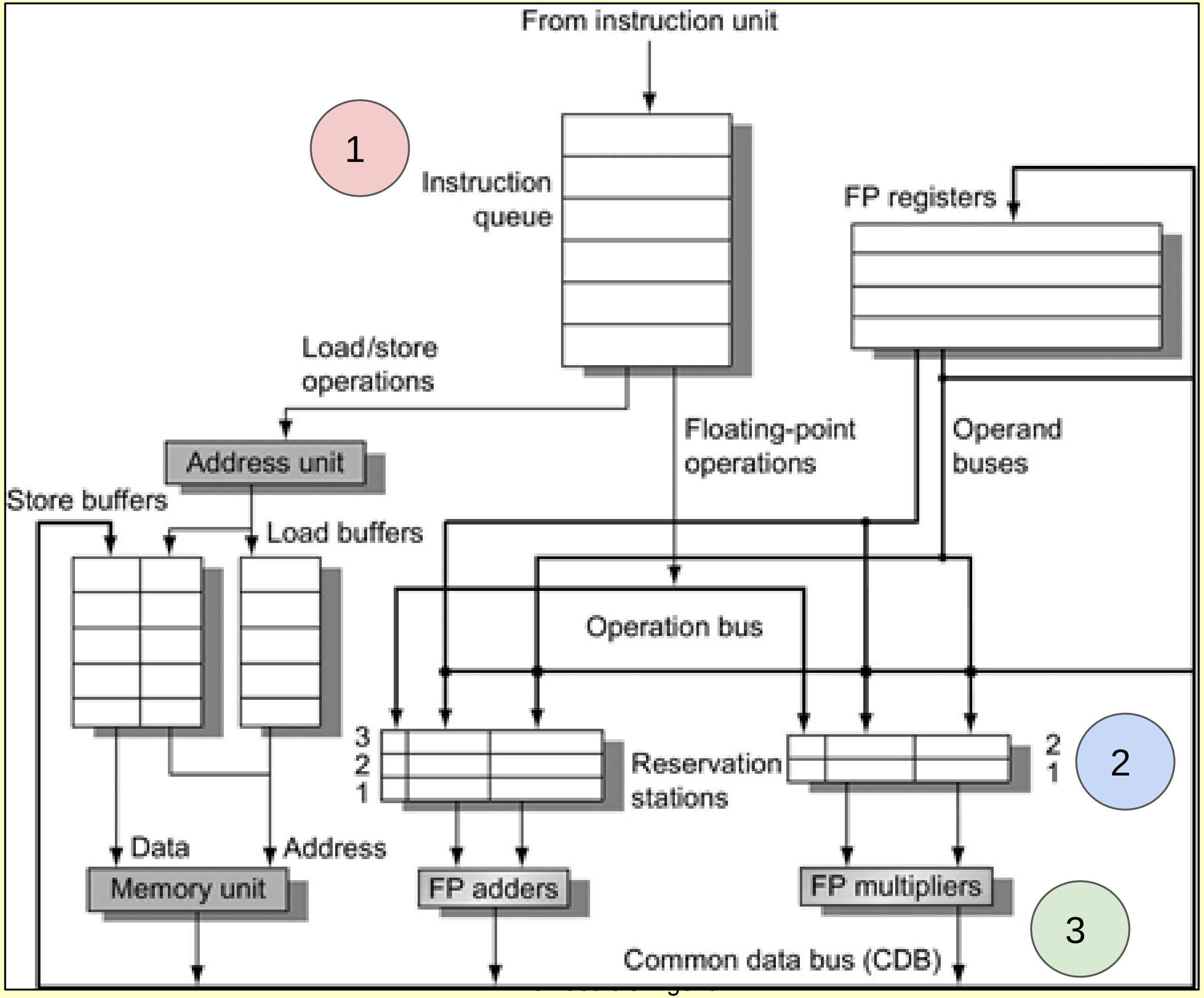

From instruction unit
1
Instruction queue
FP registers
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
3
2
1
Reservation stations
2
1
2
Data
Address
Memory unit
FP adders
FP multipliers
3
Common data bus (CDB)

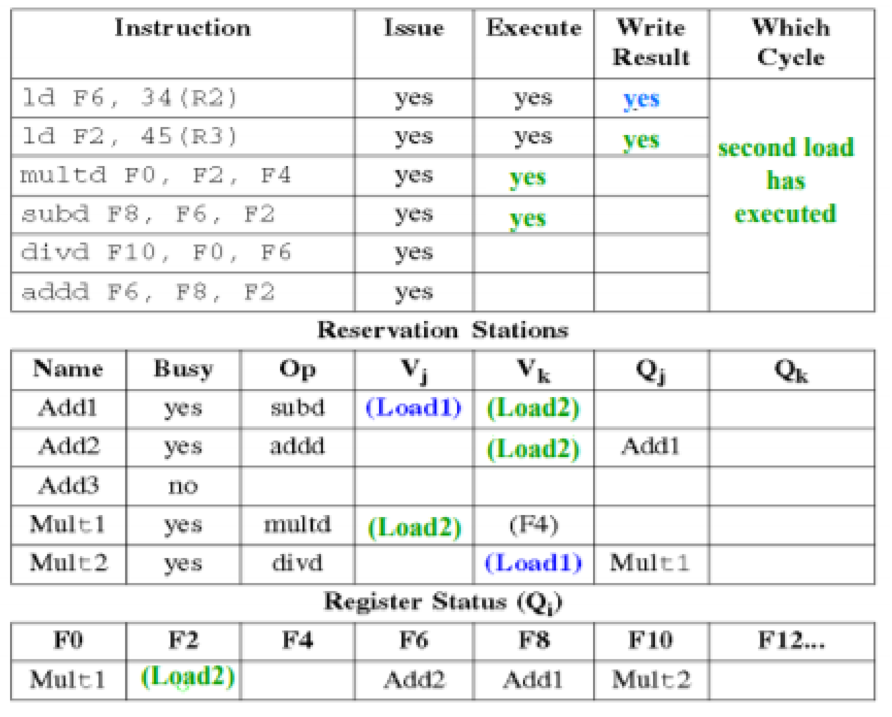
Figure 3.10

Reservation Station

- Buffer operands waiting to issue
- Associated with functional units
- Can obtain operands as soon as they are available so that they do not need to be retrieved from registers
- Provides register renaming; specifiers of the registers that hold operands renamed to those of reservation stations
- Handles data dependencies; the instruction cannot execute until the operands have been retrieved
- Allows for parallel instruction execution, as there are multiple per type of operation

Fields of Reservation Station

- **Op:** the operation to be performed
- **Qj, Qk:** the reservation stations that will produce the relevant operands, or 0 if the operand is already available or not necessary
- **Vj, Vk:** the value of the operands
- **A:** holds information for the memory address calculation for a load or store
- **Busy:** indicates this reservation station is occupied



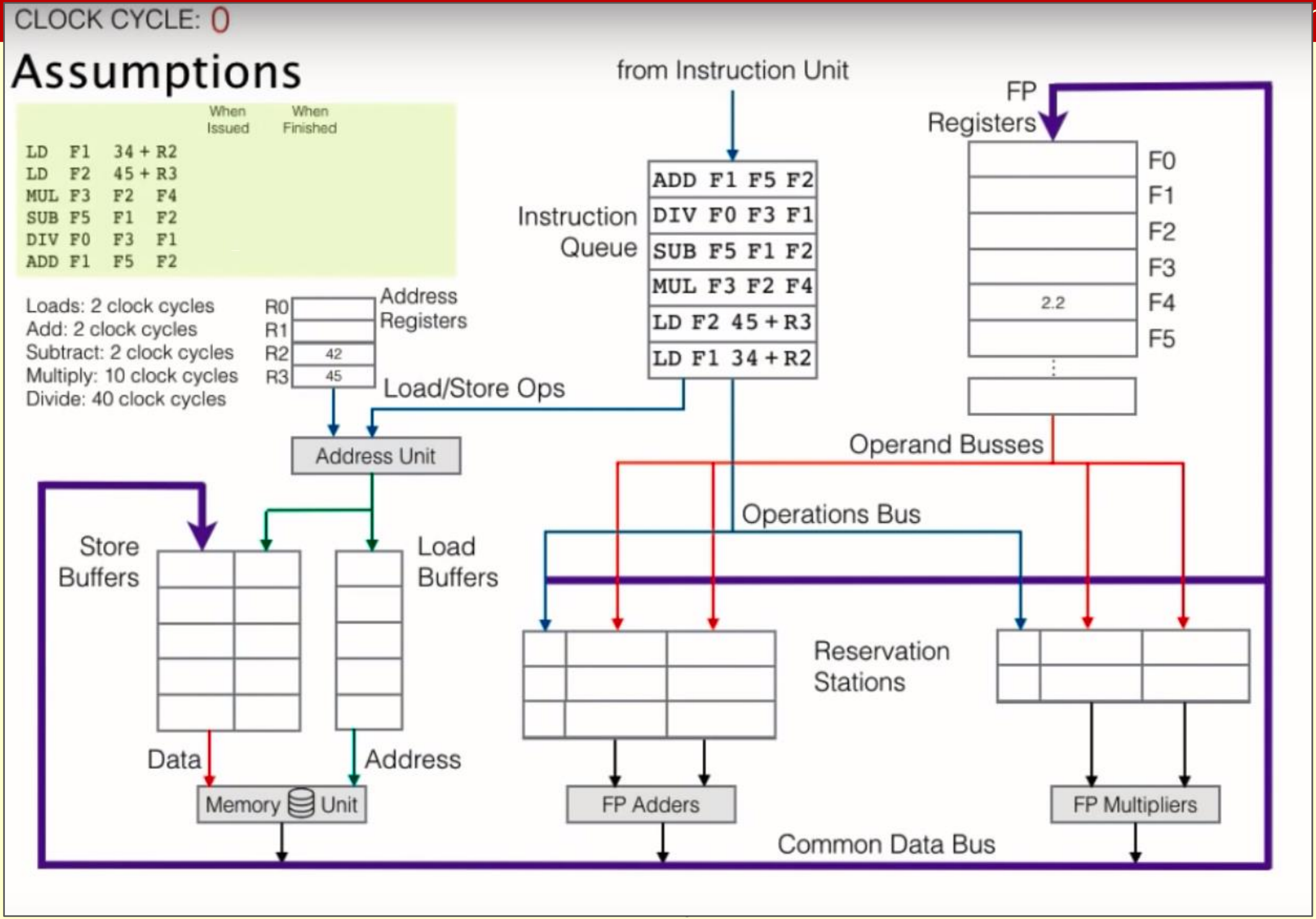
Instruction	Issue	Execute	Write Result	Which Cycle
ld F6, 34(R2)	yes	yes	yes	second load has executed
ld F2, 45(R3)	yes	yes	yes	
multd F0, F2, F4	yes	yes		
subd F8, F6, F2	yes	yes		
divd F10, F0, F6	yes			
addd F6, F8, F2	yes			

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	yes	subd	(Load1)	(Load2)		
Add2	yes	addd		(Load2)	Add1	
Add3	no					
Mult1	yes	multd	(Load2)	(F4)		
Mult2	yes	divd		(Load1)	Mult1	

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
Mult1	(Load2)		Add2	Add1	Mult2	



CLOCK CYCLE: 0

Assumptions

			When Issued	When Finished
LD	F1	34 + R2		
LD	F2	45 + R3		
MUL	F3	F2 F4		
SUB	F5	F1 F2		
DIV	F0	F3 F1		
ADD	F1	F5 F2		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

Address Registers

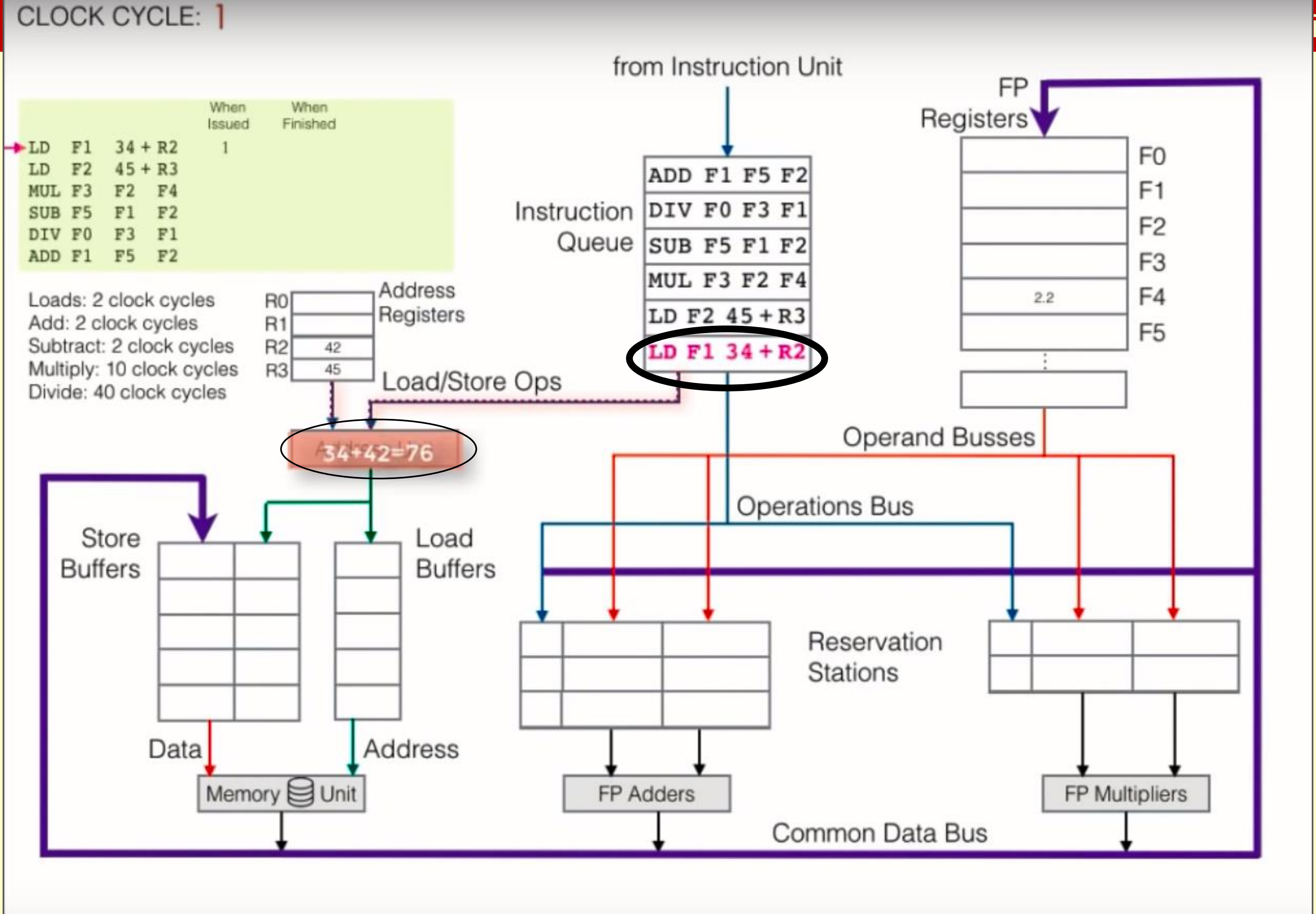
R0	
R1	
R2	42
R3	45

Instruction Queue

```
ADD F1 F5 F2
DIV F0 F3 F1
SUB F5 F1 F2
MUL F3 F2 F4
LD F2 45+R3
LD F1 34+R2
```

FP Registers

F0	
F1	
F2	
F3	
F4	2.2
F5	

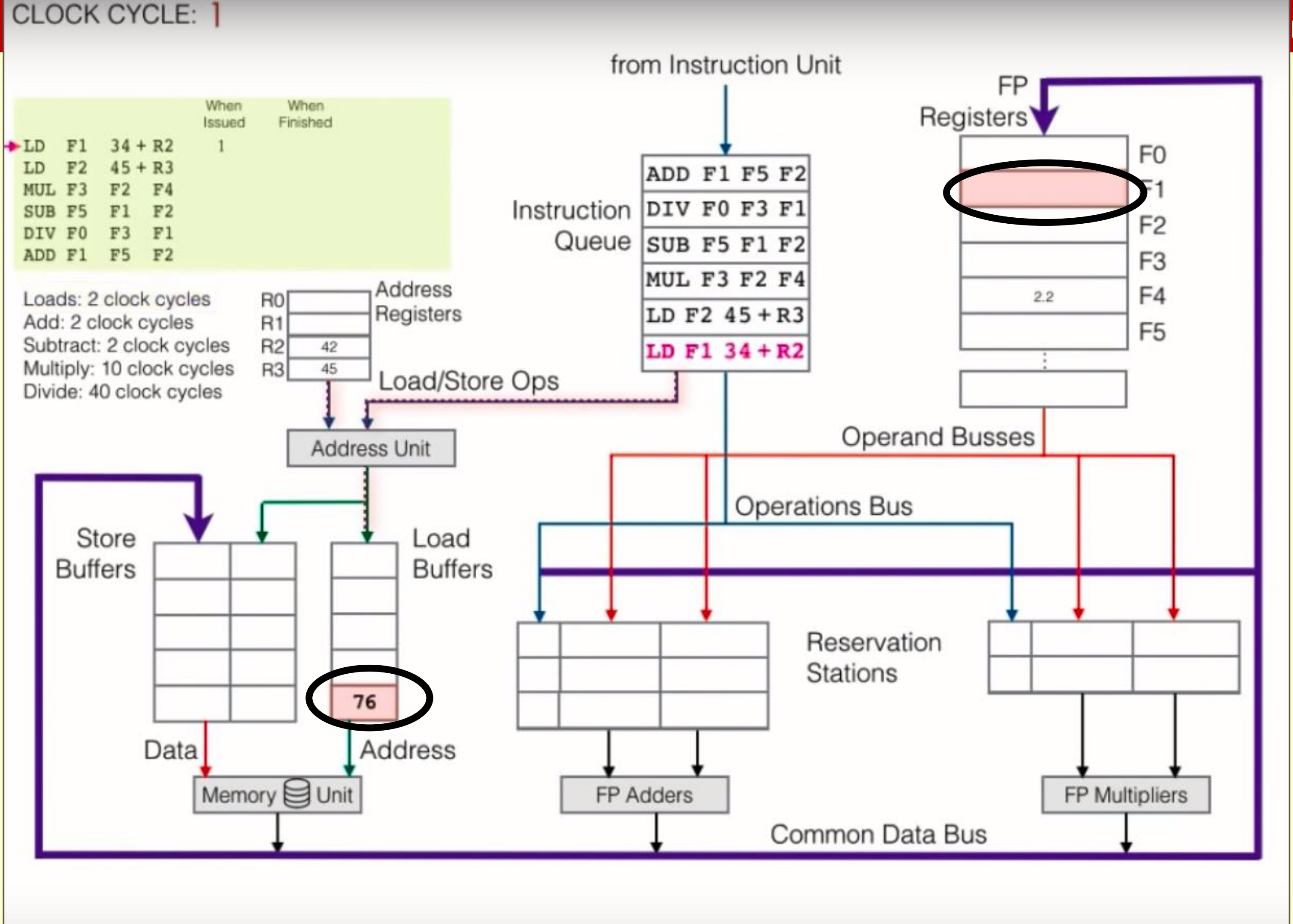


CLOCK CYCLE: 1

				When Issued	When Finished
LD	F1	34 + R2		1	
LD	F2	45 + R3			
MUL	F3	F2	F4		
SUB	F5	F1	F2		
DIV	F0	F3	F1		
ADD	F1	F5	F2		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

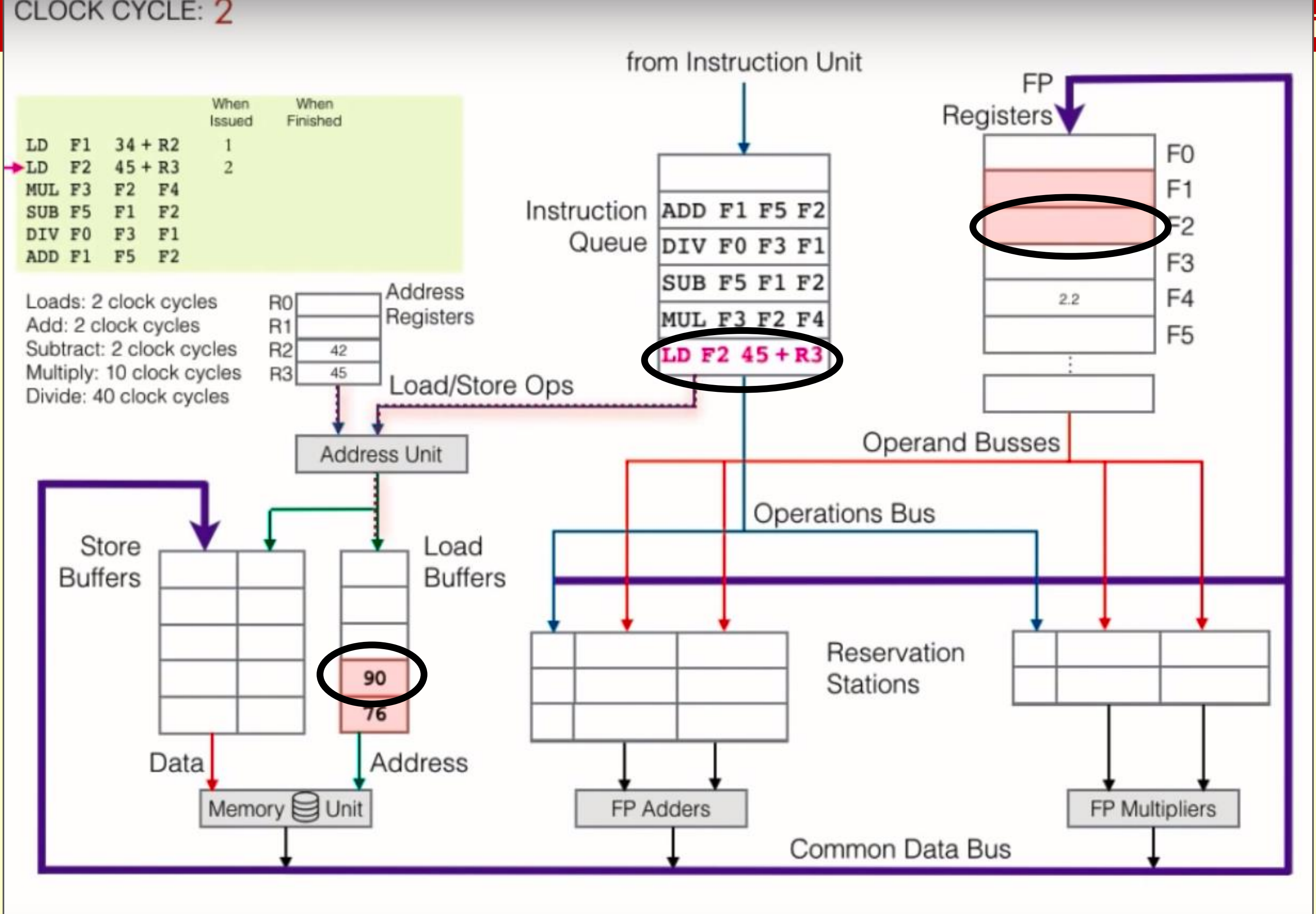
Address Registers	
R0	
R1	
R2	42
R3	45



CLOCK CYCLE: 1

			When Issued	When Finished
LD	F1	34 + R2	1	
LD	F2	45 + R3		
MUL	F3	F2	F4		
SUB	F5	F1	F2		
DIV	F0	F3	F1		
ADD	F1	F5	F2		

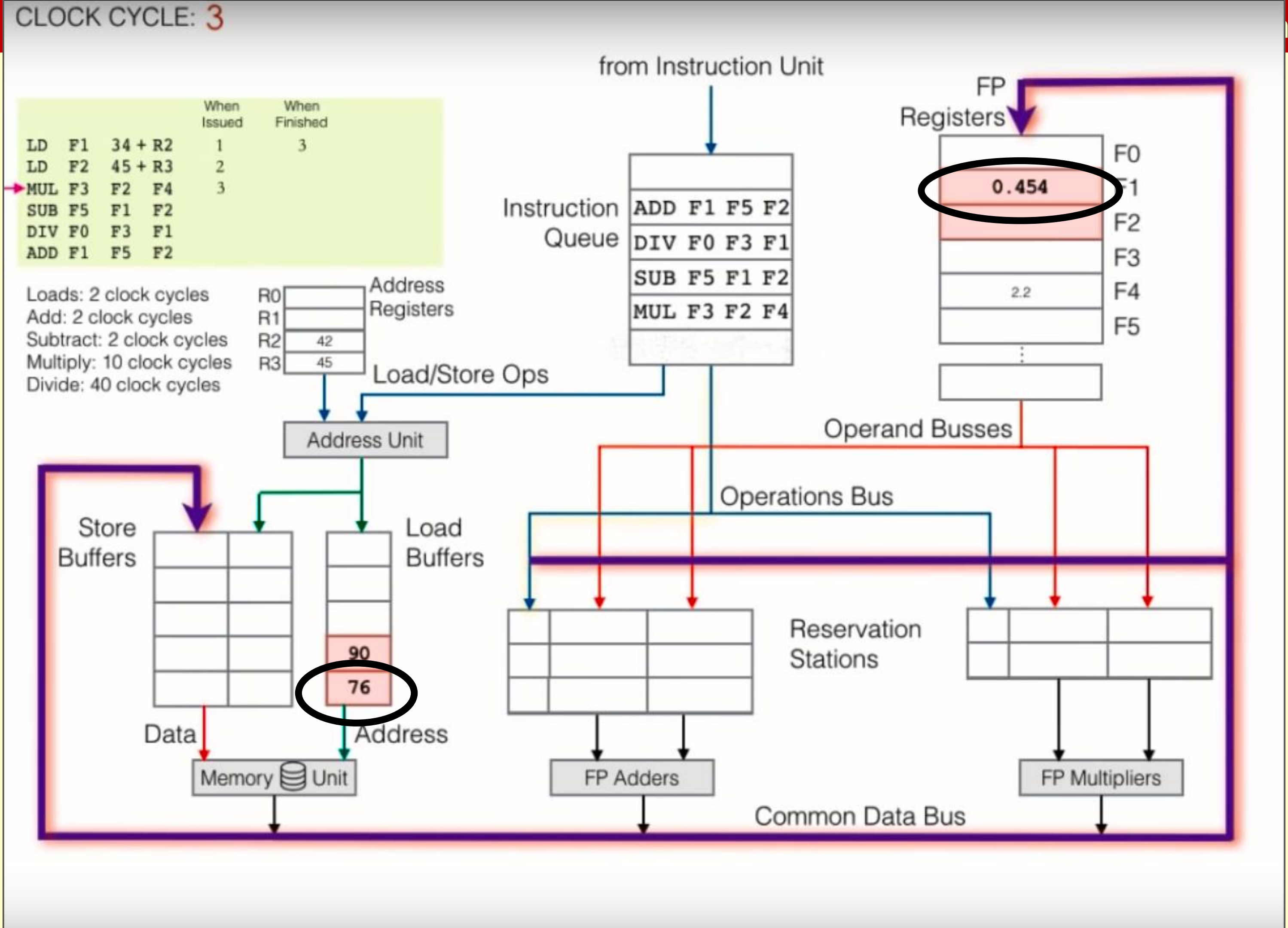
Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles



CLOCK CYCLE: 2

				When Issued	When Finished
LD	F1	34 + R2		1	
LD	F2	45 + R3		2	
MUL	F3	F2	F4		
SUB	F5	F1	F2		
DIV	F0	F3	F1		
ADD	F1	F5	F2		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles



CLOCK CYCLE: 3

Instruction	When Issued	When Finished
LD F1 34 + R2	1	3
LD F2 45 + R3	2	
MUL F3 F2 F4	3	
SUB F5 F1 F2		
DIV F0 F3 F1		
ADD F1 F5 F2		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

Address Registers	
R0	
R1	
R2	42
R3	45

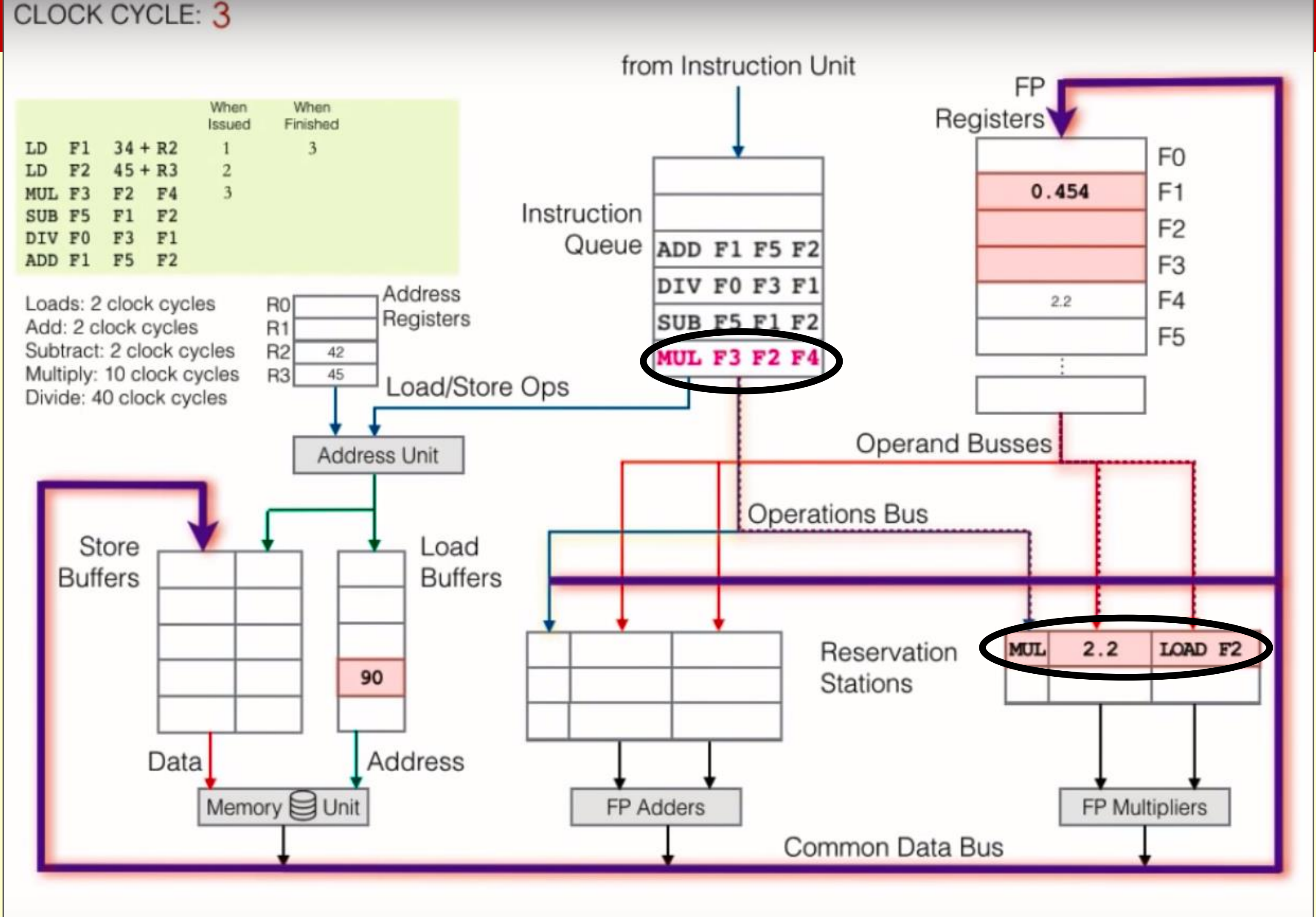
Instruction Queue

ADD F1 F5 F2
DIV F0 F3 F1
SUB F5 F1 F2
MUL F3 F2 F4

FP Registers	
F0	
F1	0.454
F2	
F3	
F4	2.2
F5	

Load Buffers

90
76



CLOCK CYCLE: 3

				When Issued	When Finished
LD	F1	34 + R2		1	3
LD	F2	45 + R3		2	
MUL	F3	F2	F4	3	
SUB	F5	F1	F2		
DIV	F0	F3	F1		
ADD	F1	F5	F2		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

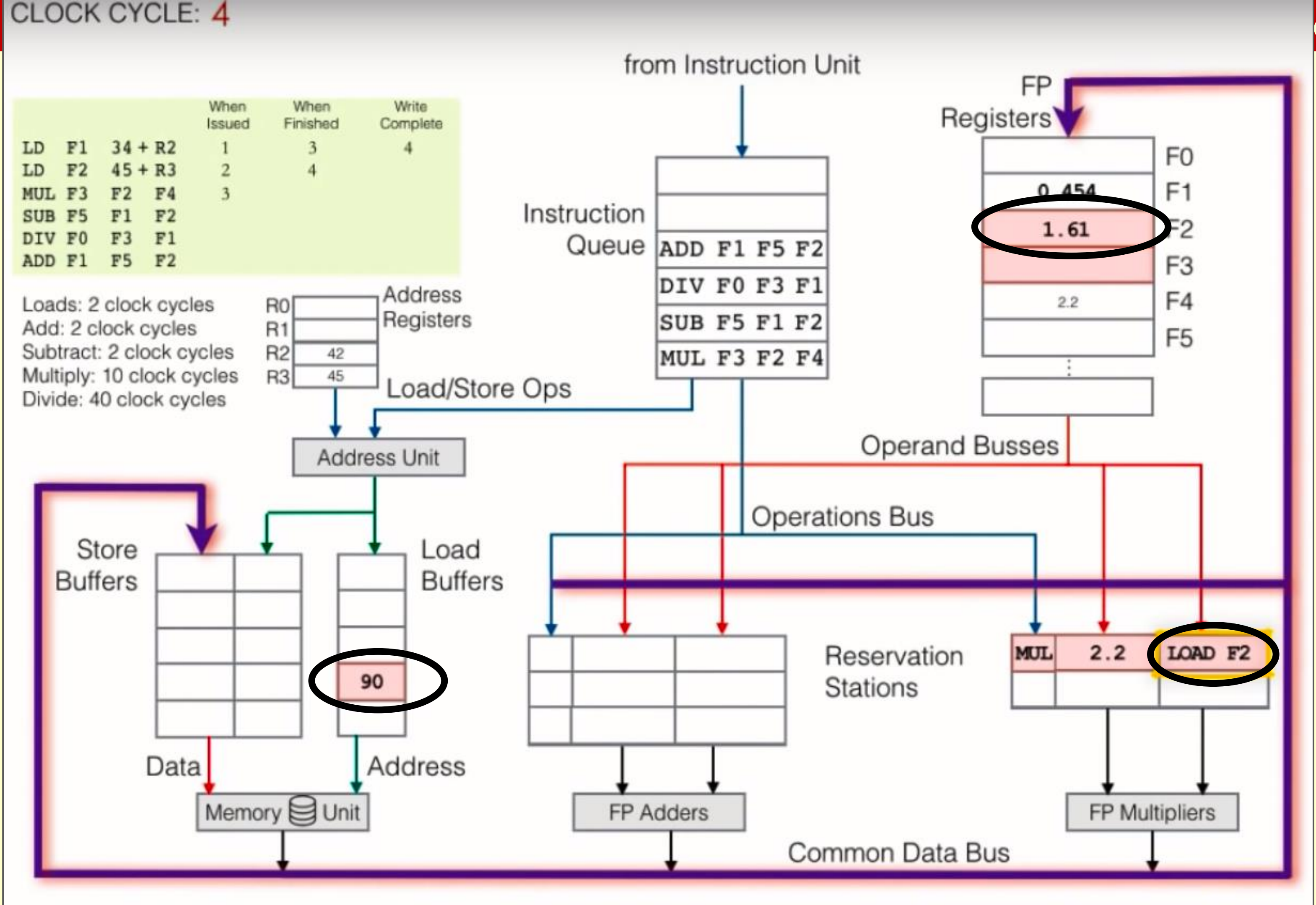
Address Registers	
R0	
R1	
R2	42
R3	45

FP Registers	
F0	
F1	0.454
F2	
F3	
F4	2.2
F5	

Instruction Queue:
ADD F1 F5 F2
DIV F0 F3 F1
SUB F5 F1 F2
MUL F3 F2 F4

Reservation Stations: MUL | 2.2 | LOAD F2

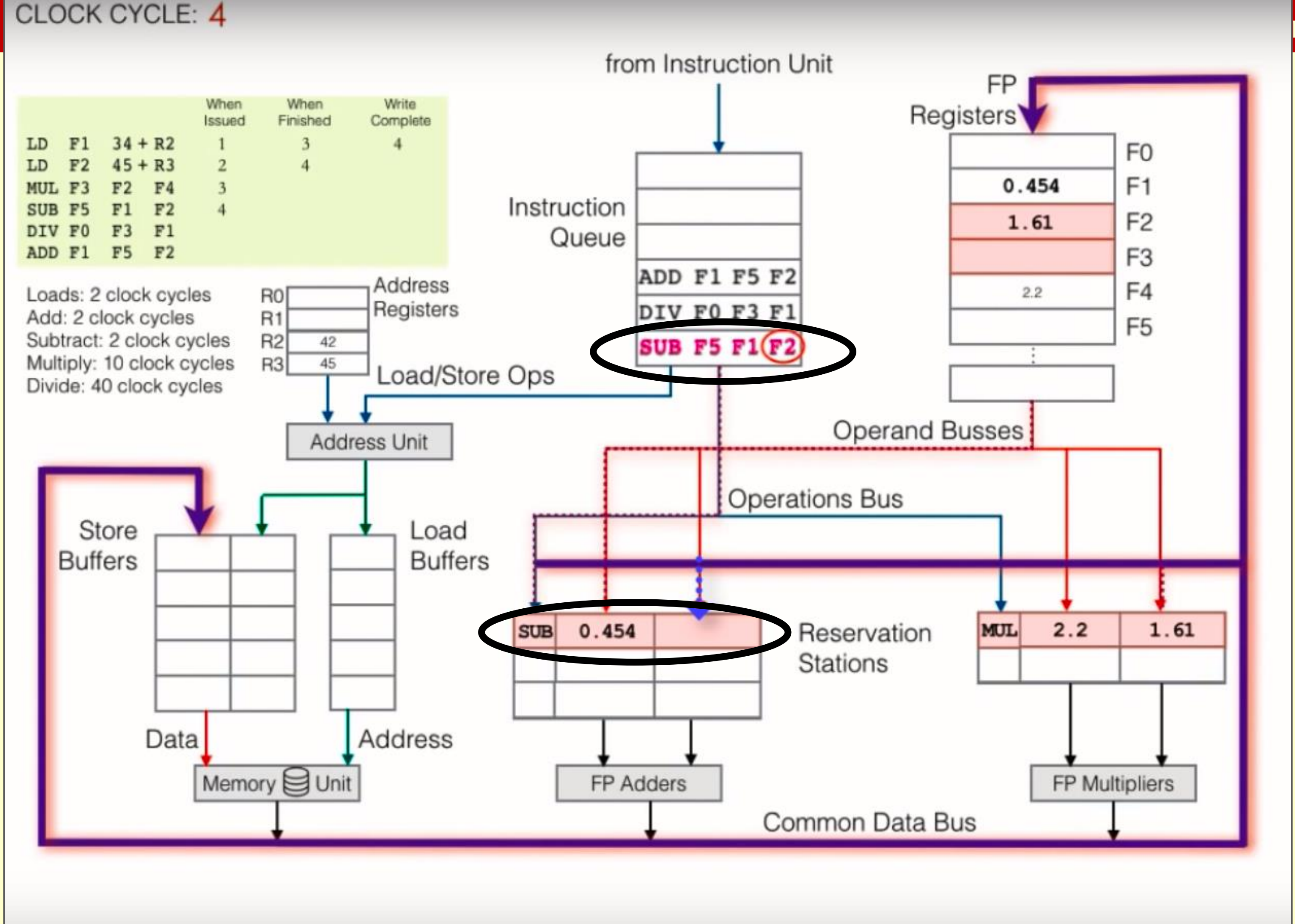
Load Buffers: 90



CLOCK CYCLE: 4

Instruction			When Issued	When Finished	Write Complete
LD	F1	34 + R2	1	3	4
LD	F2	45 + R3	2	4	
MUL	F3	F2 F4	3		
SUB	F5	F1 F2			
DIV	F0	F3 F1			
ADD	F1	F5 F2			

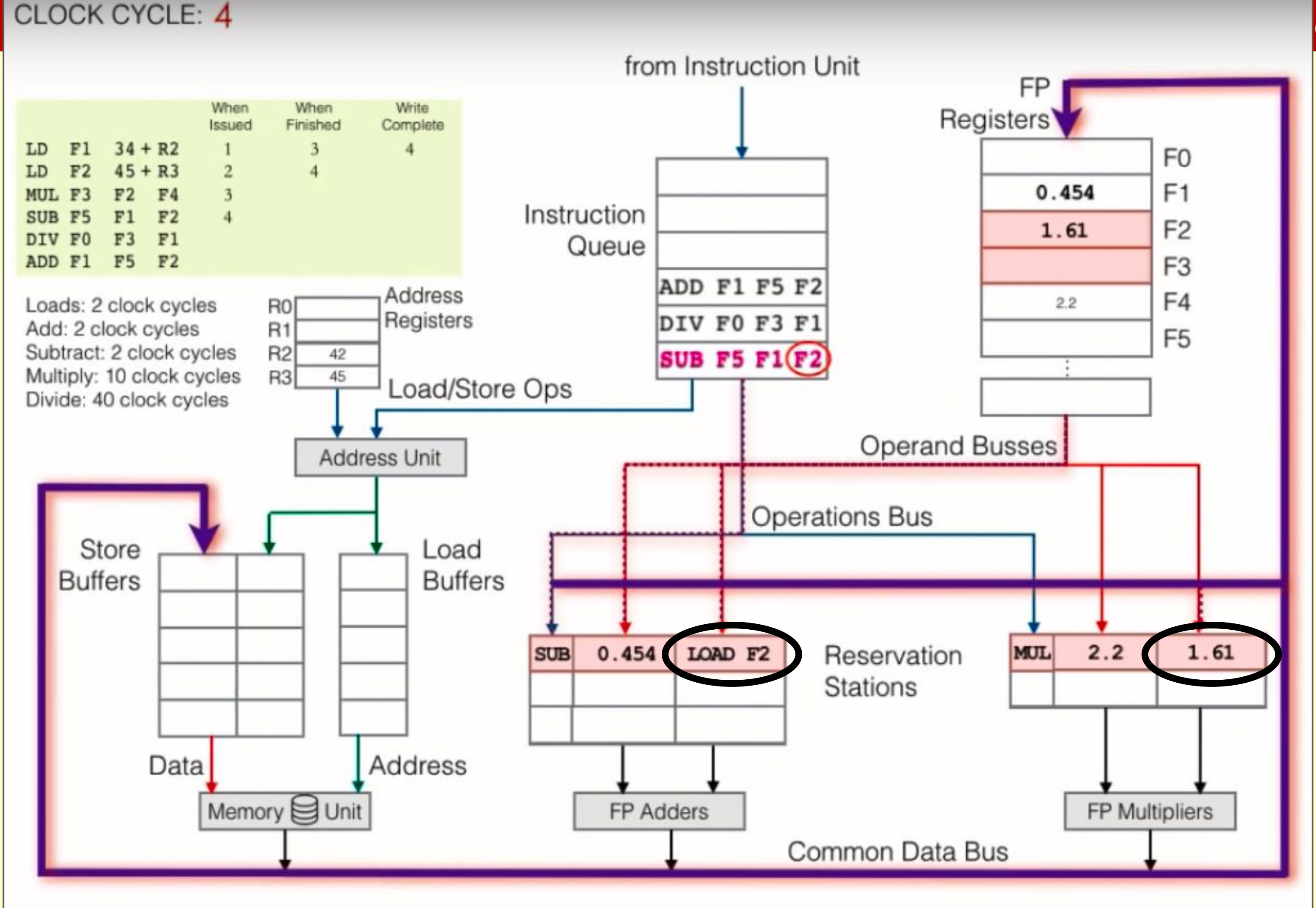
Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles



CLOCK CYCLE: 4

	When Issued	When Finished	Write Complete
LD F1 34 + R2	1	3	4
LD F2 45 + R3	2	4	
MUL F3 F2 F4	3		
SUB F5 F1 F2	4		
DIV F0 F3 F1			
ADD F1 F5 F2			

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles



CLOCK CYCLE: 4

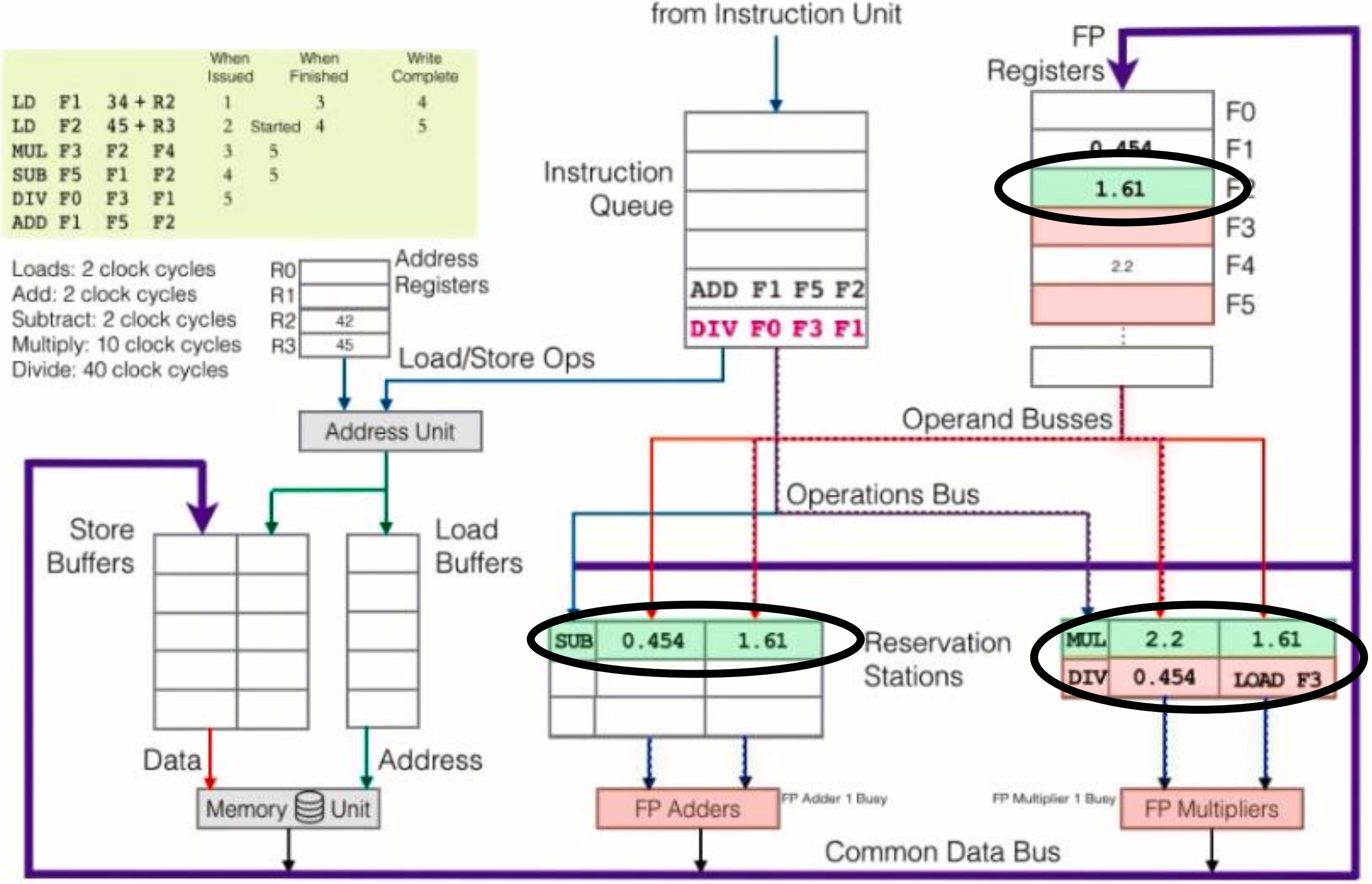
				When Issued	When Finished	Write Complete
LD	F1	34 + R2		1	3	4
LD	F2	45 + R3		2	4	
MUL	F3	F2	F4	3		
SUB	F5	F1	F2	4		
DIV	F0	F3	F1			
ADD	F1	F5	F2			

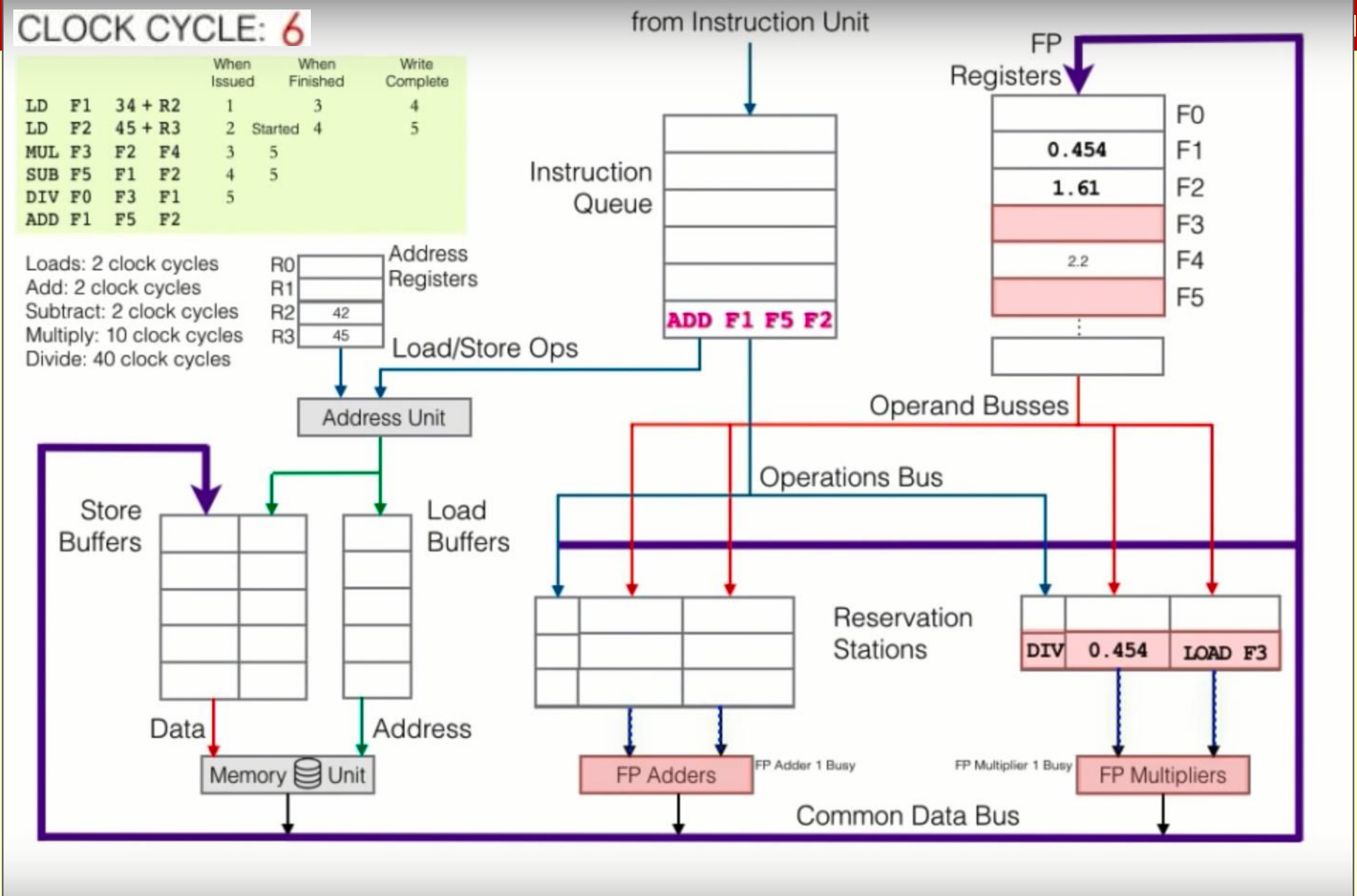
Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

CLOCK CYCLE: 5

	When Issued	When Finished	Write Complete
LD F1 34 + R2	1	3	4
LD F2 45 + R3	2	Started 4	5
MUL F3 F2 F4	3	5	
SUB F5 F1 F2	4	5	
DIV F0 F3 F1	5		
ADD F1 F5 F2			

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

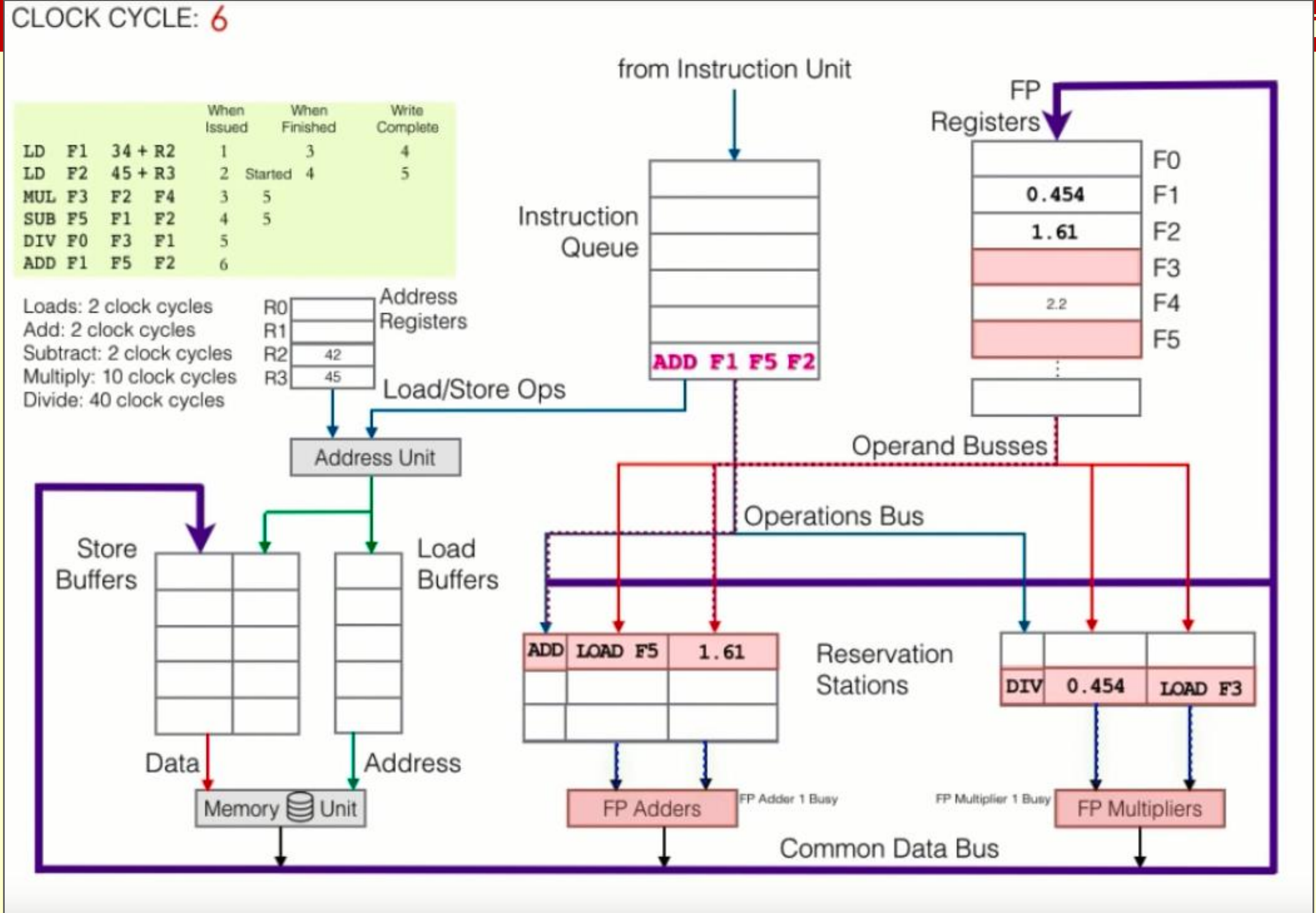




CLOCK CYCLE: 6

				When Issued	When Finished	Write Complete
LD	F1	34 + R2		1	3	4
LD	F2	45 + R3		2	Started 4	5
MUL	F3	F2	F4	3	5	
SUB	F5	F1	F2	4	5	
DIV	F0	F3	F1	5		
ADD	F1	F5	F2			

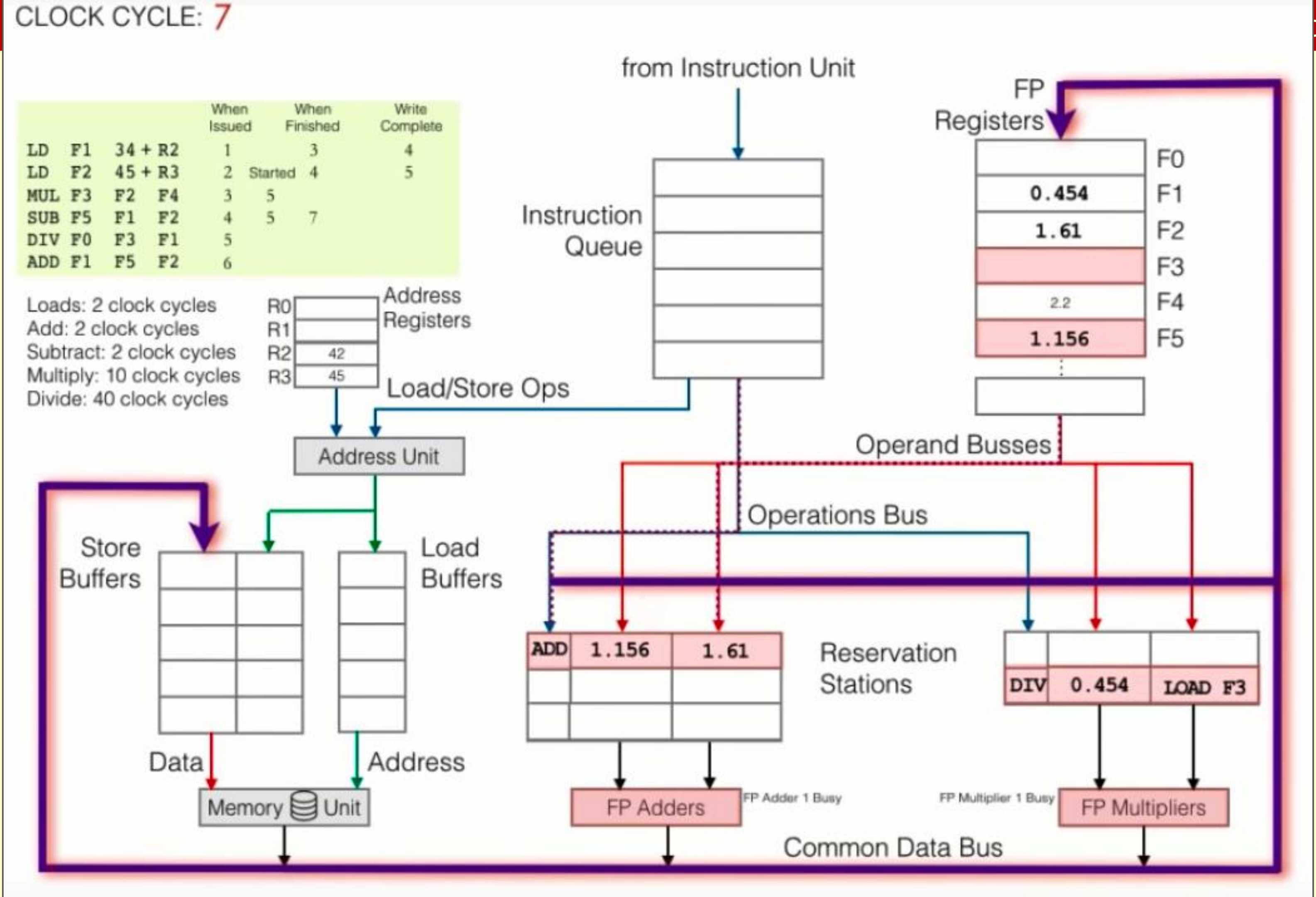
Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles



CLOCK CYCLE: 6

				When Issued	When Finished	Write Complete
LD	F1	34 + R2		1	3	4
LD	F2	45 + R3		2	Started 4	5
MUL	F3	F2	F4	3	5	
SUB	F5	F1	F2	4	5	
DIV	F0	F3	F1	5		
ADD	F1	F5	F2	6		

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

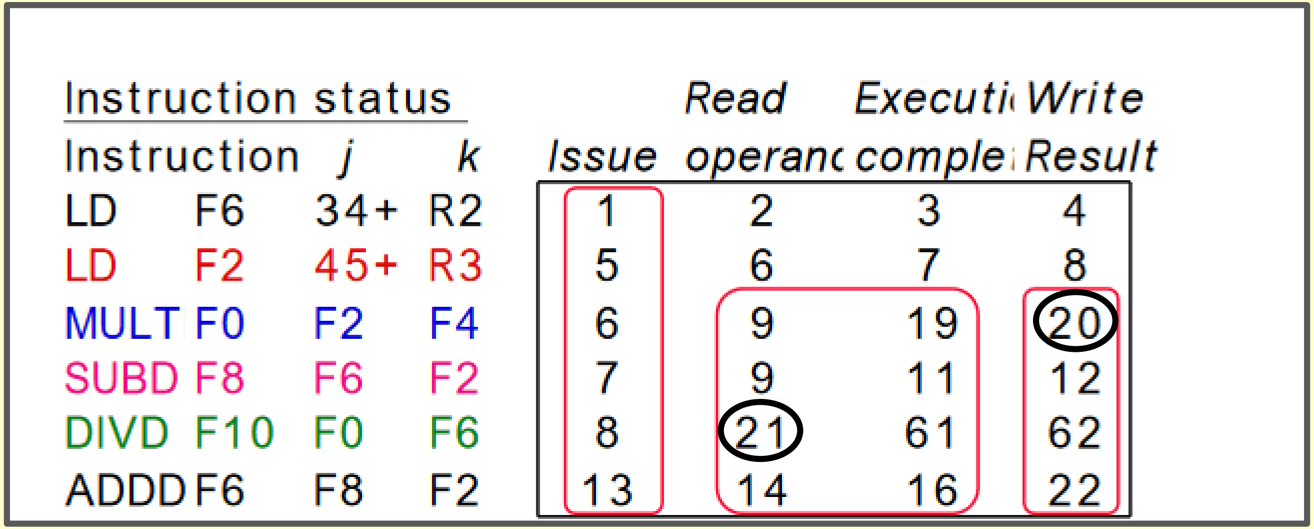


CLOCK CYCLE: 7

				When Issued		When Finished	Write Complete
LD	F1	34 + R2		1		3	4
LD	F2	45 + R3		2	Started	4	5
MUL	F3	F2	F4	3	5		
SUB	F5	F1	F2	4	5	7	
DIV	F0	F3	F1	5			
ADD	F1	F5	F2	6			

Loads: 2 clock cycles
Add: 2 clock cycles
Subtract: 2 clock cycles
Multiply: 10 clock cycles
Divide: 40 clock cycles

62 Clock Cycle Example



Instruction status					Read	Executi	Write
Instruction		j	k	Issue	operanc	comple	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Questions?

Modern High-Performance Computing

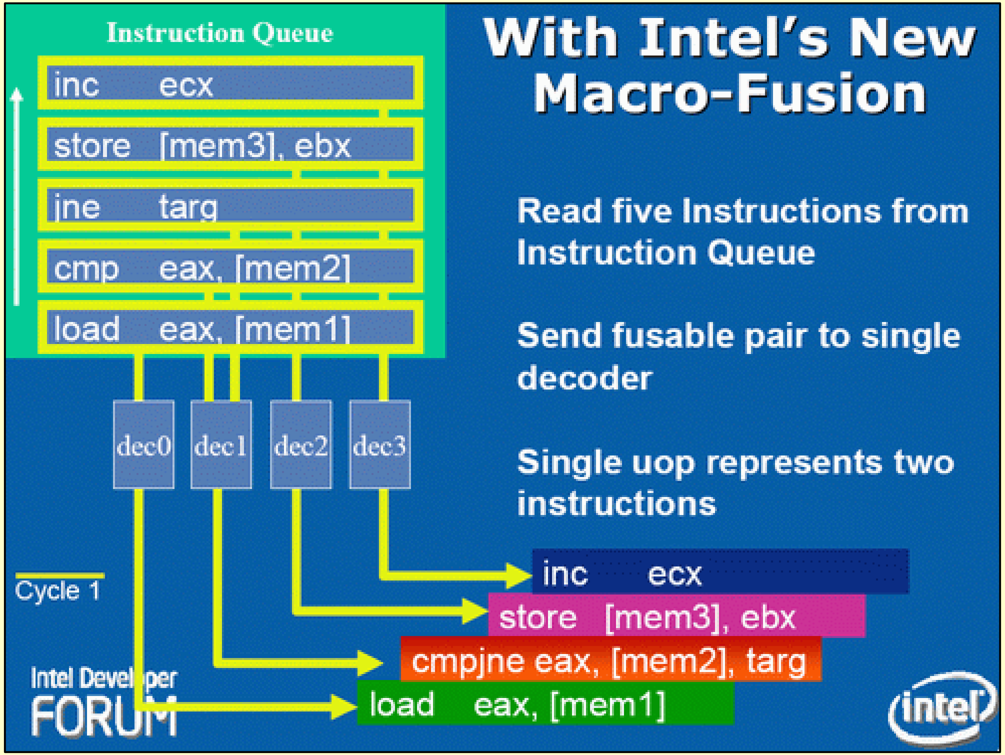
- IBM System/360 Model 91
    - Unit at NASA Goddard was most powerful computer in operation in 1968
    - 16.6 million instructions/second
- Nowadays, Tomasulo’s algorithm is no longer used without heavy modifications and improvements
    - Intel Core Microarchitecture

Intel Core Microarchitecture

- Tomasulo’s algorithm used as a basis
- Intel’s pipeline contains three stages:
  - **In-order issue front end:** provides decoded pre-fetched instructions to core
  - **Out-of-order execution core:** re-orders micro-operations to minimize loss of cycles
  - **In-order retirement unit:** updates the in-order of original instructions after micro-operations finish
- Still retains focus on supporting dynamic scheduling and branch prediction while avoiding hazards

Intel Core: Decoding Optimizations

- Intel Core has dedicated hardware which performs instruction pre-decoding
  - Determines length of operation
  - Decodes prefixes associated with instruction
  - Tags instruction with properties that can be associated (e.g. if branch)
- **Macro-fusion:** common instruction pairs are fused into a single instruction during decoding to reduce overall work
- **Micro-fusion:** operations derived from the same macro-fusion operation are combined to reduce the number of micro-operations, effectively minimizing power usage and re-order buffer memory usage



With Intel's New Macro-Fusion

Read five Instructions from Instruction Queue

Send fusable pair to single decoder

Single uop represents two instructions

On average in a typical x86 program, for every 10 instructions 2 are fused together.

Intel Core: Register Renaming

- Intel employs register renaming to avoid WAR and WAW hazards
- Execution core unit has a renamer component which renames architectural registers to a larger set of micro-architectural registers
- Allows recovery from imprecise speculations since micro-architectural register which holds the destination does not become the architectural register until the instruction commits

Intel Core: Other Optimizations

- Intel Core microarchitecture uses a separate memory ordering buffer (MOB) to handle hazards due to name dependencies
    - This offers an advantage over Tomasulo’s approach in handling memory accesses to contiguous locations since instructions can be continuously issues without checking for concurrent access to the same location in memory
- **Store forwarding:** while handling loads that follow stores, the microarchitecture can forward the data directly from the store to the load

Issues with Extending Tomasulo’s Algorithm

- Cannot handle precise exceptions in the original implementation (Solutions do exist, such as a ROB)
- With superscalar, the selection logic has an amount of $N^2\log(w)$ connections.
- Also, comes with the issues of other superscalar such as time spent checking for dependencies, branching, and the limitation to how many instructions can be fetched

Superscalar Pipeline Stages with Tomasulo’s

- Fetch - Fetch the instruction from the instruction cache
- Dispatch - Send the instruction to the Reservation Station
- Issue - Send the instruction to the functional unit
- Execute - Use the specified functional unit to do the operation (bypass occurs here)
- Writeback - Write the result to the CDB and to memory if needed.

Extending to Superscalar

For an N-way superscalar processor with W number of reservation stations we will need:

- Reservation Station: N tag/value w-ports (dispatch), N value reservation ports (Issue), 2N tag CAMs (write)
- Select Logic: W -> N priority encoder (Issue)
- Map Table: 2N reservation ports (dispatch), N write ports (dispatch)
- Register File: 2N reservation ports (dispatch), N write ports (dispatch)
- Common Data Bus: N connections (write)

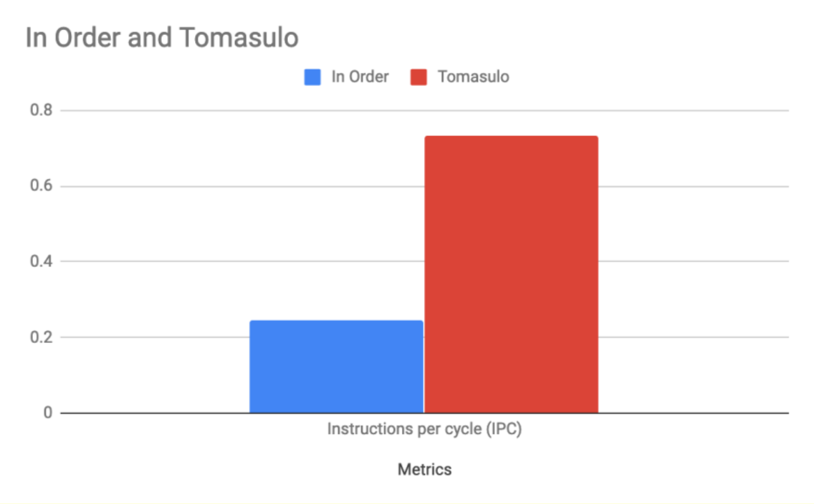
Addressing the Issues

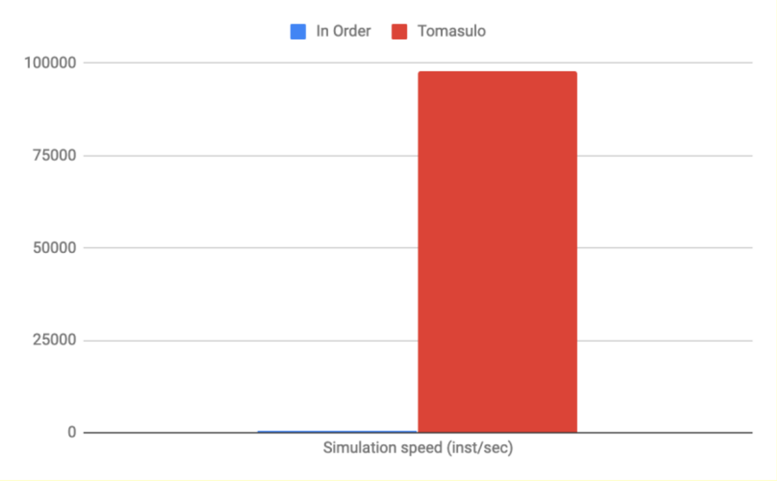
Two common methods exist for fixing some of the previous issues:

- **Split Design**: N banks for the Reservation station
- Implement N (N/W) -> 1 priority encoders
- Simpler design for the multiplexer: N log(W/N) encoders, however, there is less scheduling flexibility
- **FIFO Design**: Stack design where only the first Reservation Station entry per bank can be issued.
- No select logic needed
- However, there is also less scheduling flexibility

Benchmarking

- Study: “An optimizing pipeline stall reduction algorithm for power and performance on multi-core CPUs”
  - Benchmarked In-Order vs Tomasulo’s algorithm performance
  - Sample program run on the SimpleScalar computer architecture simulator

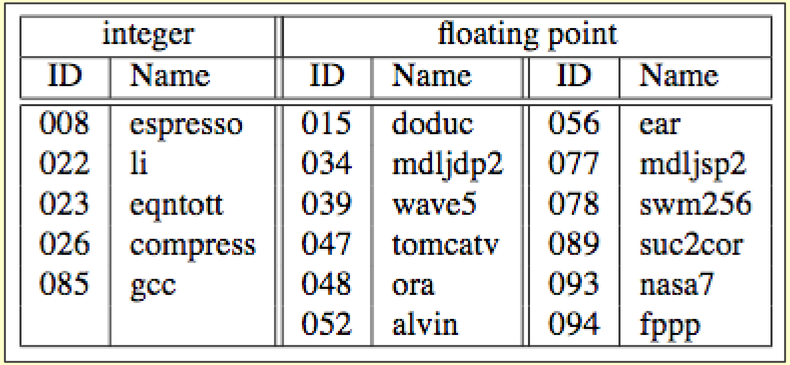




Tomasulo’s Algorithm

Benchmarking

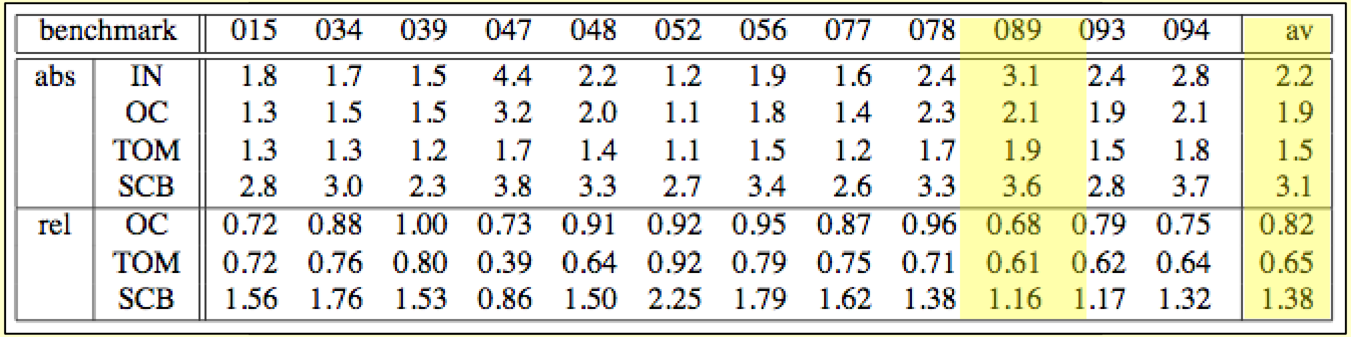
- Study: “The Impact of Hardware Scheduling Mechanisms on the Performance and Cost of Processor Designs”
    - SPEC-92 Benchmark Suite



integer		floating point			
ID	Name	ID	Name	ID	Name
008	espresso	015	doduc	056	ear
022	li	034	mdljdp2	077	mdljsp2
023	eqntott	039	wave5	078	swm256
026	compress	047	tomcatv	089	suc2cor
085	gcc	048	ora	093	nasa7
		052	alvin	094	fppp

Benchmarking

- Study: “The Impact of Hardware Scheduling Mechanisms on the Performance and Cost of Processor Designs”
  - CPI of in-order execution (IN), out-of-order completion (OC), Tomasulo out-of-order dispatch scheduler (TOM), and scoreboard scheduler (SCB)



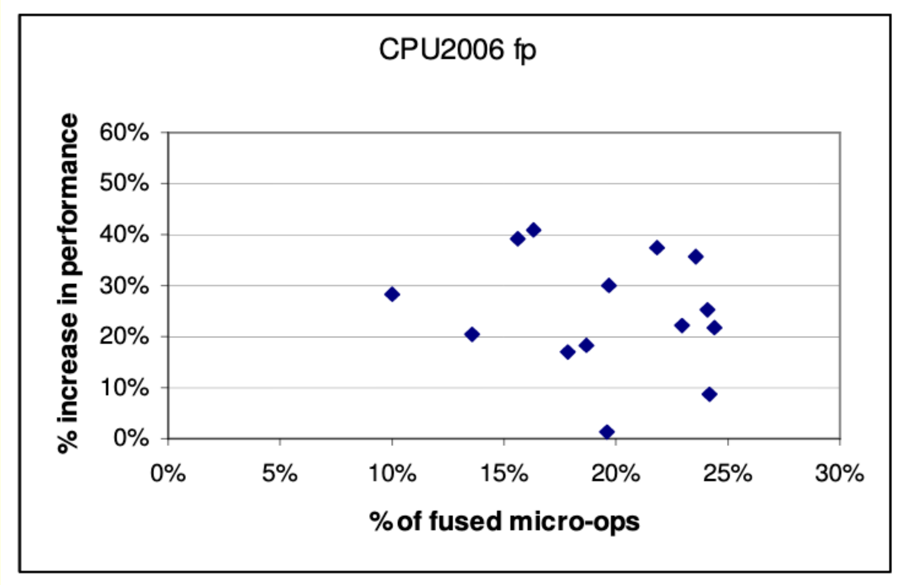
benchmark		015	034	039	047	048	052	056	077	078	089	093	094	av
abs	IN	1.8	1.7	1.5	4.4	2.2	1.2	1.9	1.6	2.4	3.1	2.4	2.8	2.2
	OC	1.3	1.5	1.5	3.2	2.0	1.1	1.8	1.4	2.3	2.1	1.9	2.1	1.9
	TOM	1.3	1.3	1.2	1.7	1.4	1.1	1.5	1.2	1.7	1.9	1.5	1.8	1.5
	SCB	2.8	3.0	2.3	3.8	3.3	2.7	3.4	2.6	3.3	3.6	2.8	3.7	3.1
rel	OC	0.72	0.88	1.00	0.73	0.91	0.92	0.95	0.87	0.96	0.68	0.79	0.75	0.82
	TOM	0.72	0.76	0.80	0.39	0.64	0.92	0.79	0.75	0.71	0.61	0.62	0.64	0.65
	SCB	1.56	1.76	1.53	0.86	1.50	2.25	1.79	1.62	1.38	1.16	1.17	1.32	1.38

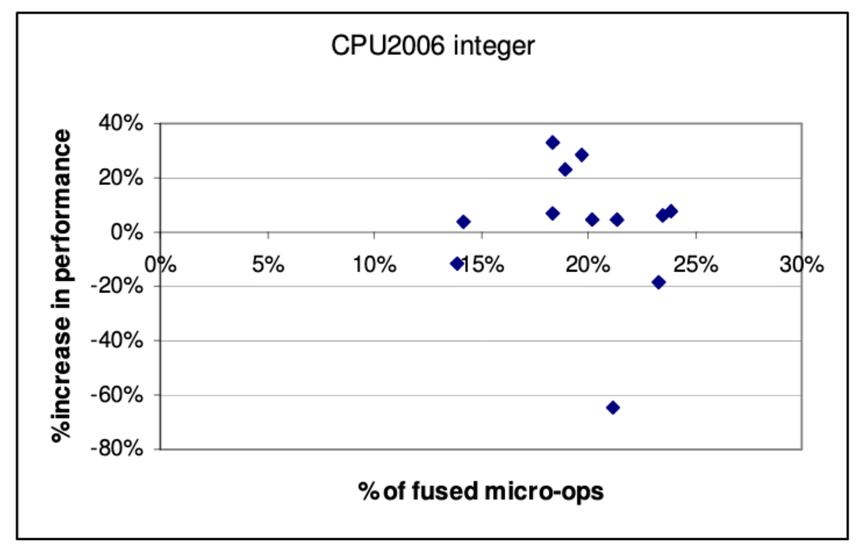
Floating-point Benchmarks

Integer Benchmarks

Benchmarking

- Study: “Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor”
  - ○ Documents performance of Intel’s micro fusion





Questions?

References

- Hennessey, *Computer Architecture*
- https://courses.cs.washington.edu/courses/csep548/06au/lectures/tomasulo.pdf
- https://www.anandtech.com/show/1998/3
- https://www.youtube.com/watch?v=jyjE6NHtkiA
- https://pdfs.semanticscholar.org/4371/3e64fe36ebe1cab7e5f94b5af10b050b2111.pdf
- https://link.springer.com/article/10.1186/s13673-014-0016-8
- http://web.eecs.umich.edu/~twenisch/470_F07/lectures/6.pdf
- https://en.wikichip.org/wiki/macro-operation_fusion
- https://www.cs.umd.edu/~meesh/cmsc411/website/projects/dynamic/tomasulo.html
- https://www.cc.gatech.edu/~milos/Teaching/CS6290F07/4_Tomasulo.pdf
- https://people.eecs.berkeley.edu/~pattrsn/252F96/Lecture04.pdf