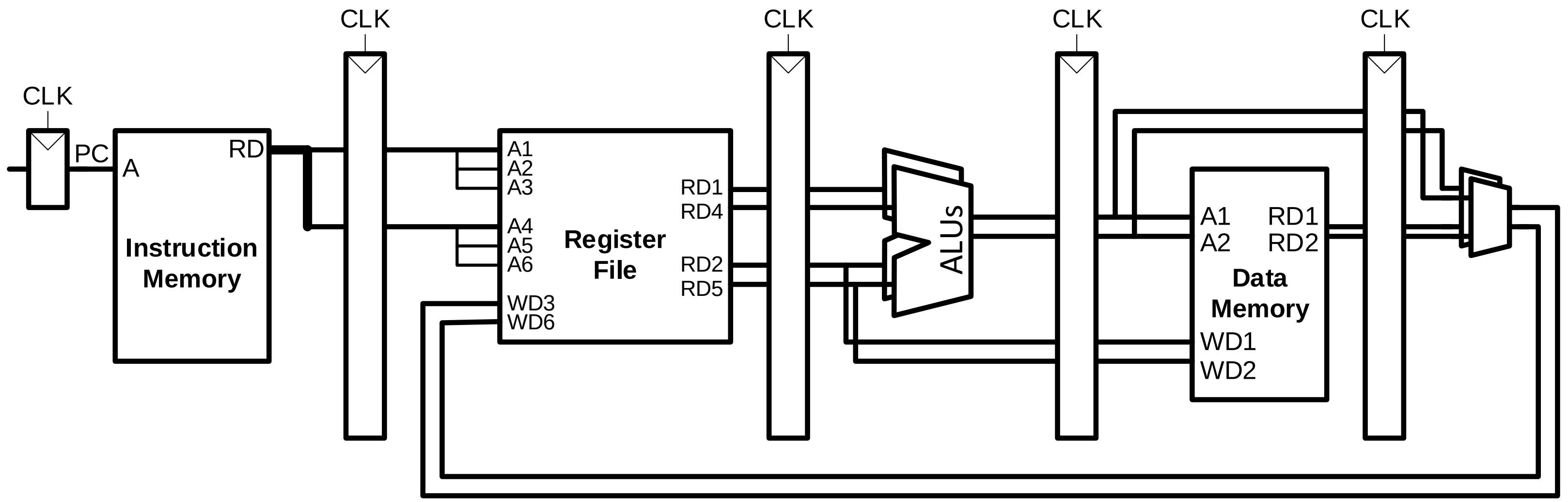
Chapter 7: Microarchitecture

Superscalar & Out of Order Processors

Superscalar Processors

- Multiple copies of datapath execute multiple instructions at once
- Dependencies make it tricky to issue multiple instructions at once

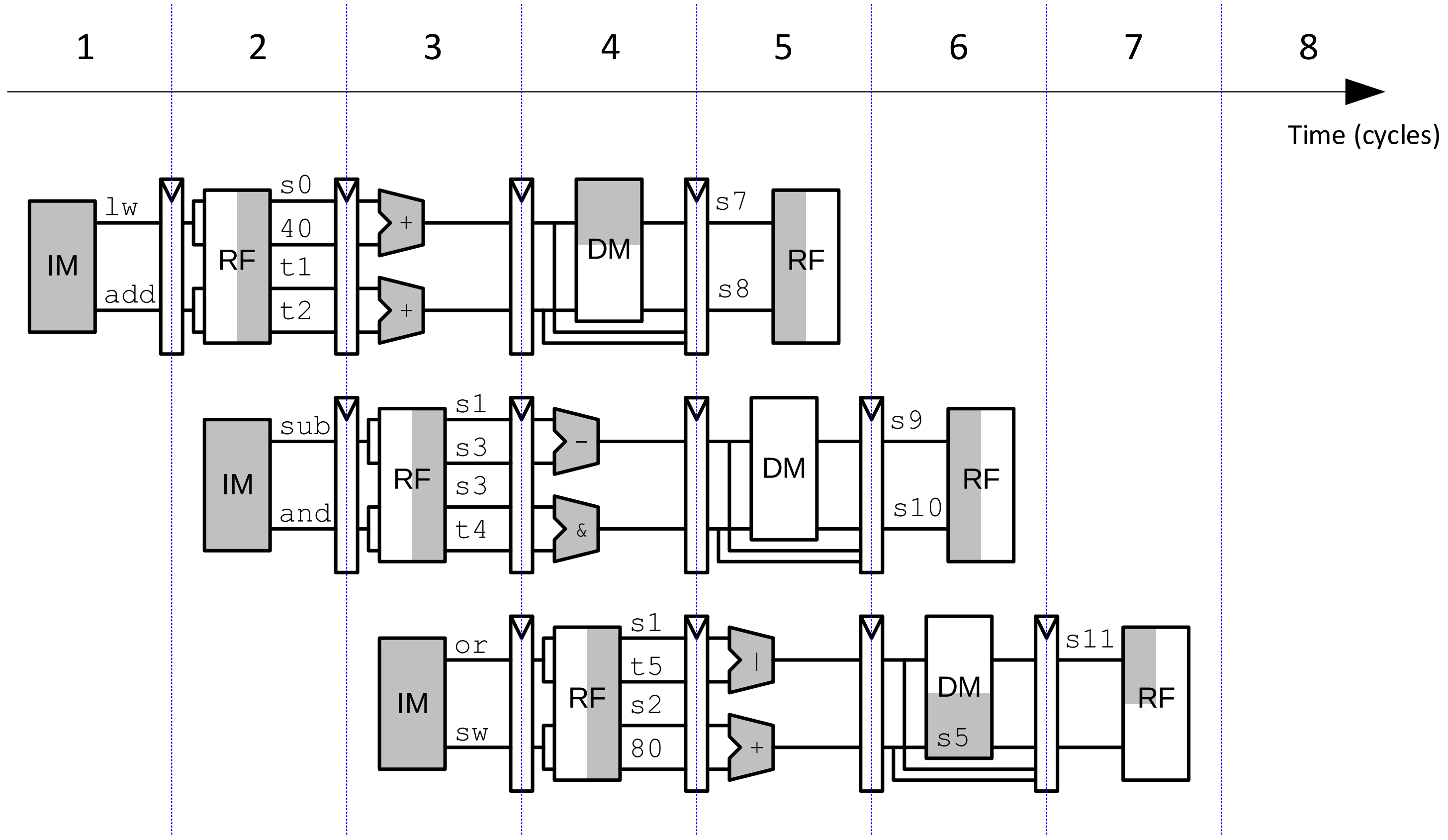


Superscalar Example

Ideal IPC: **2**

Actual IPC: **2**

```
lw   s7, 40(s0)
add  s8, t1, t2

sub  s9, s1, s3
and  s10, s3, t4

or   s11, s1, t5
sw   s5, 80(s2)
```

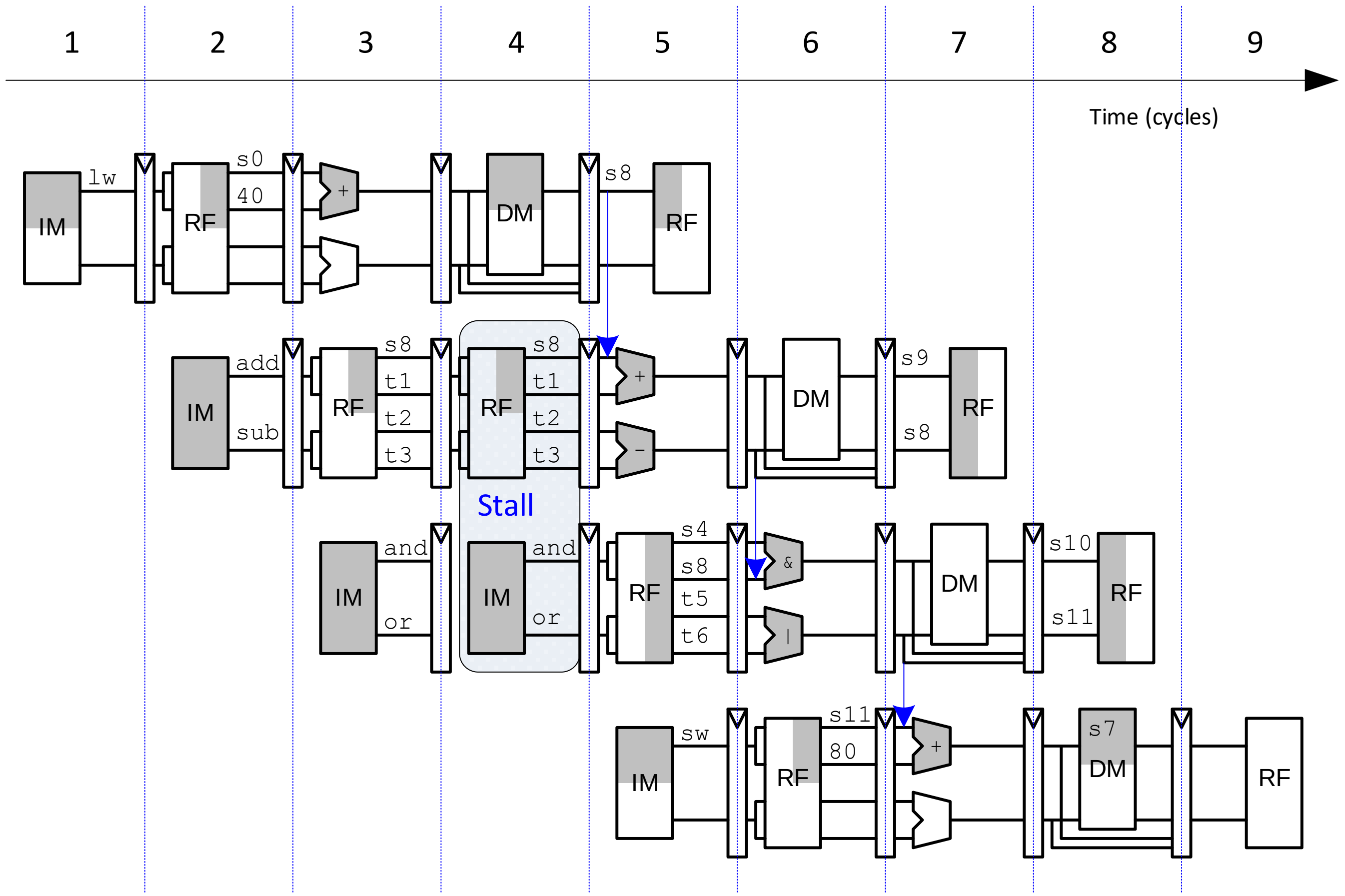


Superscalar with Dependencies

Ideal IPC: **2**

Actual IPC: **6/5 = 1.2**

```
lw   s8, 40(s0)

add  s9, s8, t1
sub  s8, t2, t3

and  s10, s4, s8
or   s11, t5, t6

sw   s7, 80(s11)
```



Out of Order (OOO) Processor

- Looks ahead across multiple instructions
- Issues as many instructions as possible at once
- Issues instructions out of order (as long as no dependencies)
- **Dependencies:**
  - **RAW** (read after write): one instruction writes, later instruction reads a register
  - **WAR** (write after read): one instruction reads, later instruction writes a register
  - **WAW** (write after write): one instruction writes, later instruction writes a register

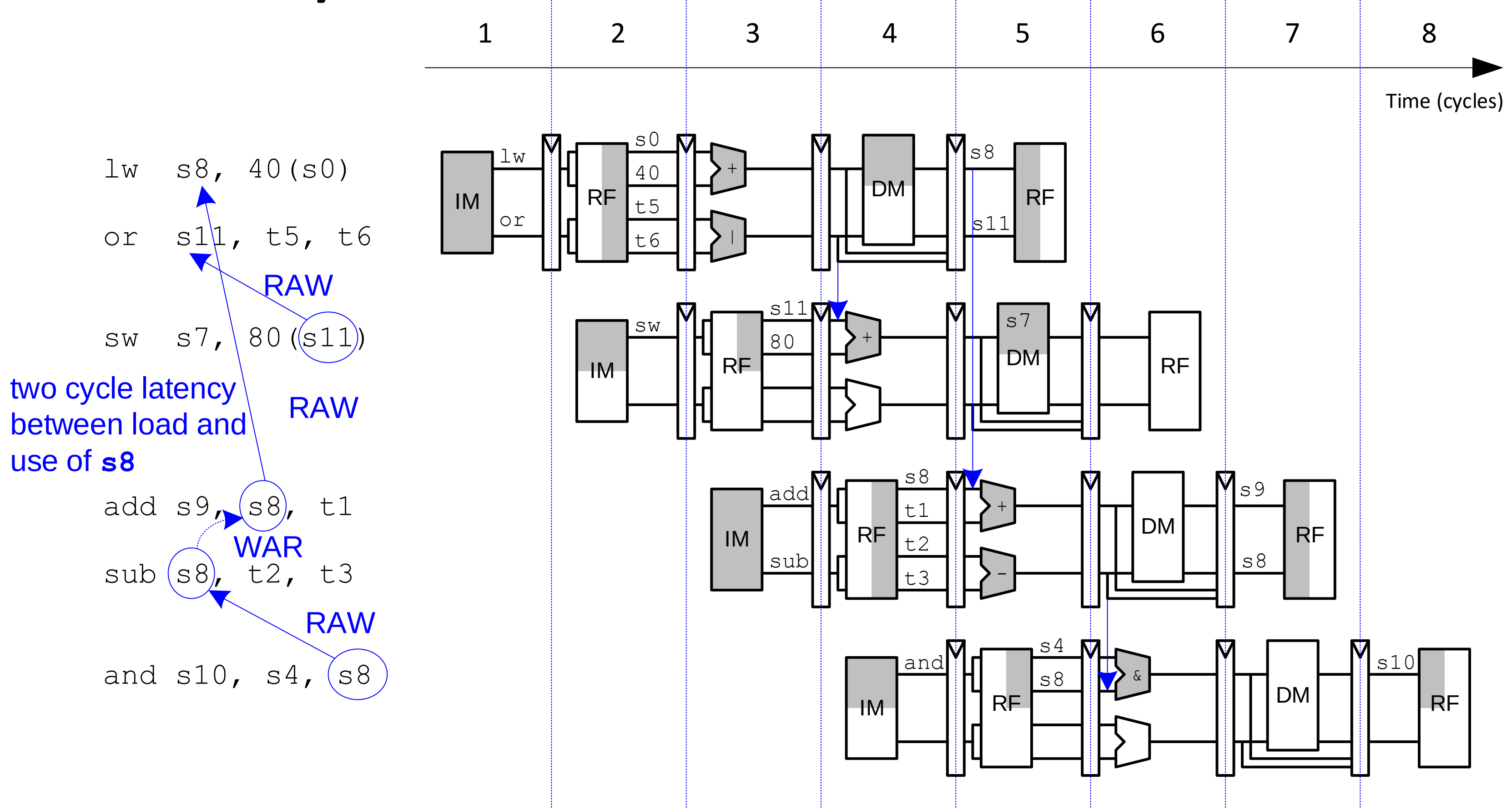
Out of Order (OOO) Processor

- **Instruction level parallelism (ILP):** number of instruction that can be issued simultaneously (average < 3)
- **Scoreboard:** table that keeps track of:
  - Instructions waiting to issue
  - Available functional units
  - Dependencies

Out of Order Processor Example

Ideal IPC: **2**

Actual IPC: **6/4 = 1.5**

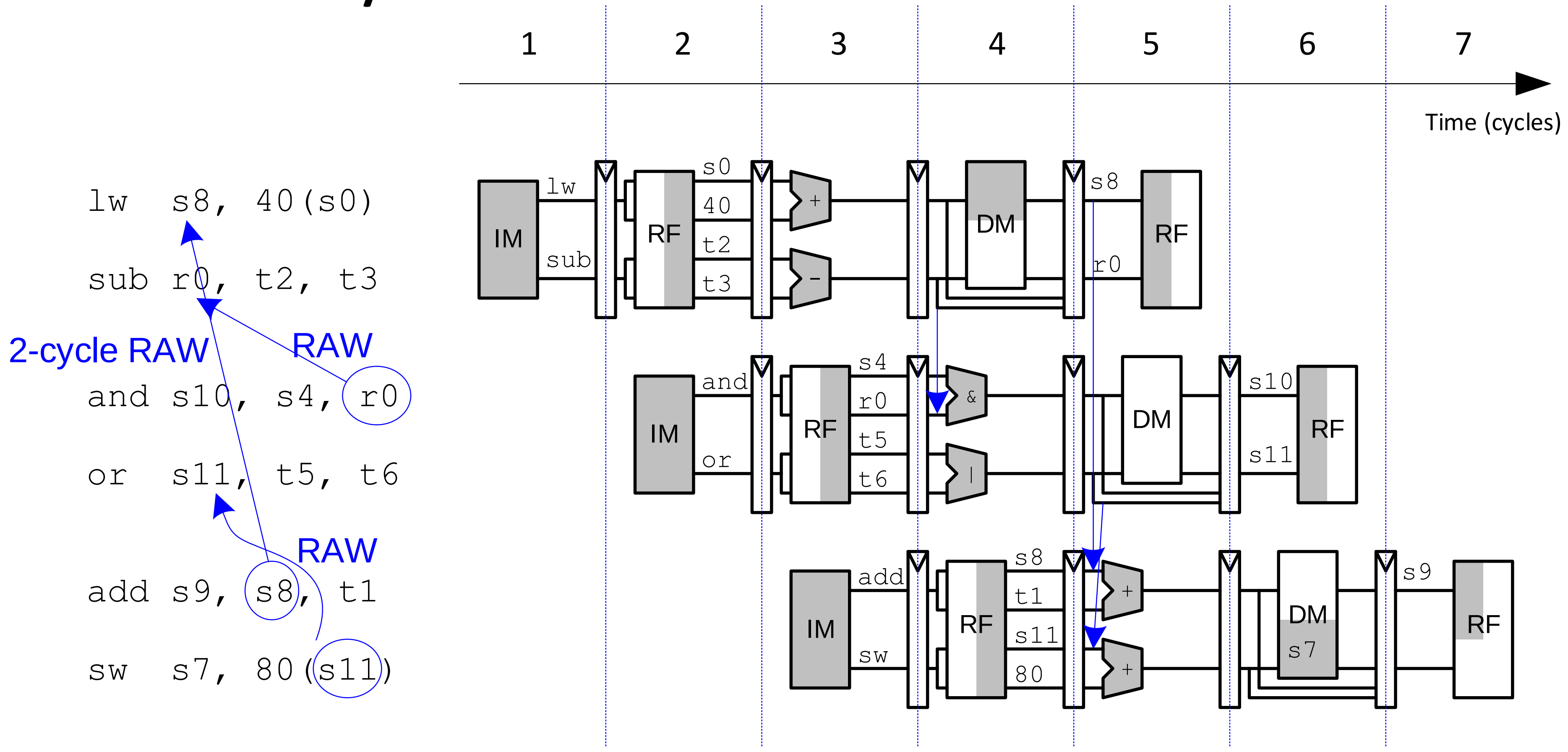


```
lw   s8, 40(s0)

or   s11, t5, t6

sw   s7, 80(s11)

add s9, s8, t1

sub s8, t2, t3

and s10, s4, s8
```

Register Renaming

Ideal IPC: **2**

Actual IPC: **6/3 = 2**

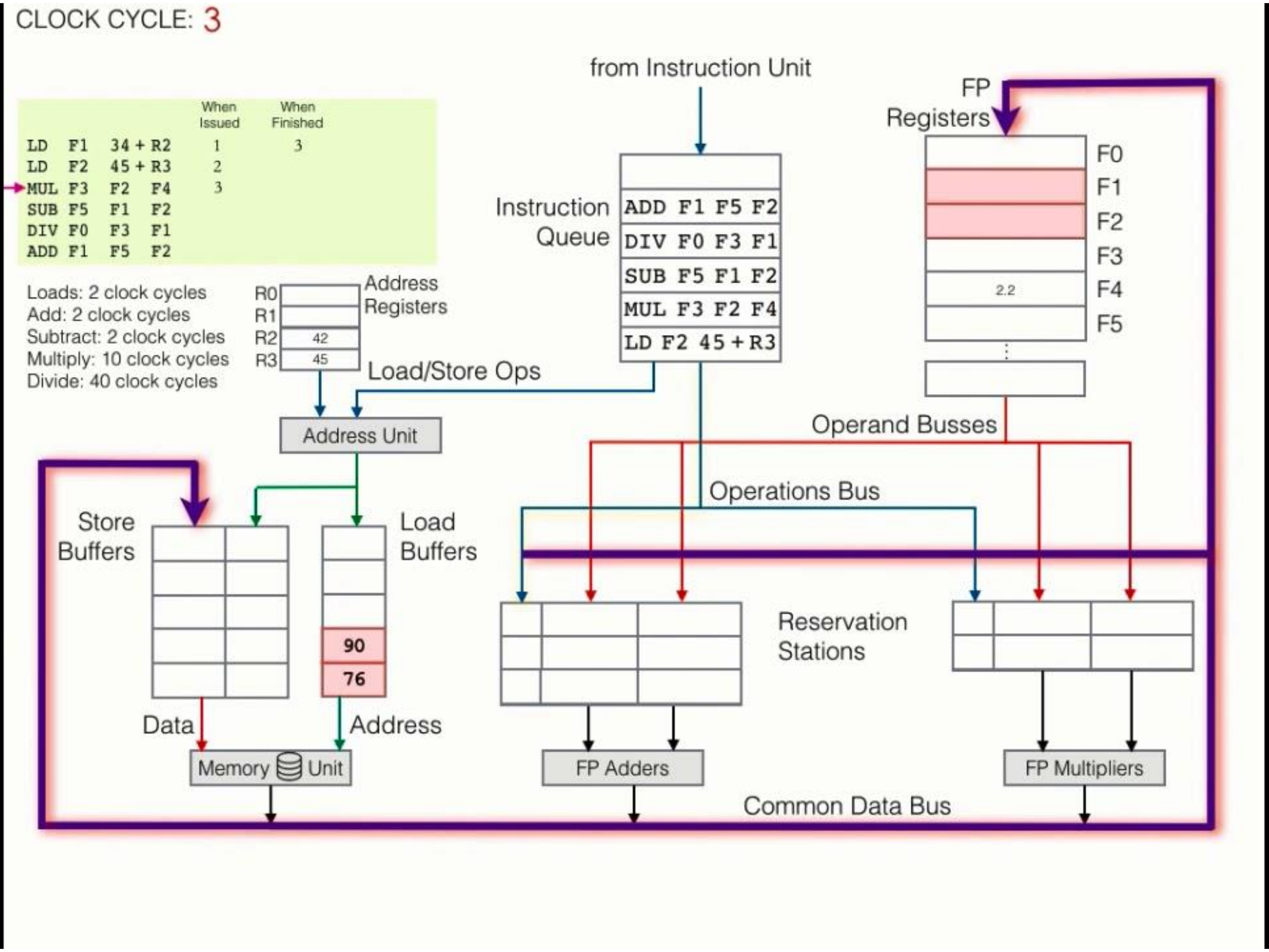


```
lw   s8, 40(s0)
sub  r0, t2, t3
and  s10, s4, r0
or   s11, t5, t6
add  s9, s8, t1
sw   s7, 80(s11)
```

Tomasulo’s Algorithm

- A great example of supporting:
  - Dynamic scheduling (out-of-order issuing).
  - Register renaming (reservation stations and extra register files such as address registers.
  - Floating point operations (different instructions can take different number of clock cycles to finish based on their complexity.
- An operation bus keeps track of dependencies between different instructions and update the reservation stations.
- An operands bus updates the reservation stations from the FP register file if the data in the file are up-to-date (no data dependency)
- A common data bus updates both the main FP register file and the reservation stations at the same time (in case of data dependencies).
- It’s biggest hurdle is control hazards because it limits the instructions that can be fetched at the same time.

Tomasulo’s Algorithm



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

Tomasulo’s Algorithm

- As seen in this example of Tomasulo’s algorithm that supports floating point arithmetic, Instructions are taking different number of clock cycles that varies form 2 for loads to 40 for FP division.
- There is an instruction queue waiting to be issued
- There is the following extra registers in addition to the regular FP register file:
  - Address registers used by both stores and loads to find the pointers to the memory.
  - Load and store buffers
  - Reservation stations for both FP addition and multiplication
- There is also multiple Functional Units (FU) which is a specialized ALU such as address unit, FP adder unit, and FP multiplier unit.
- Finally, there is common data bus, operation bus, and operands Bus and a data memory

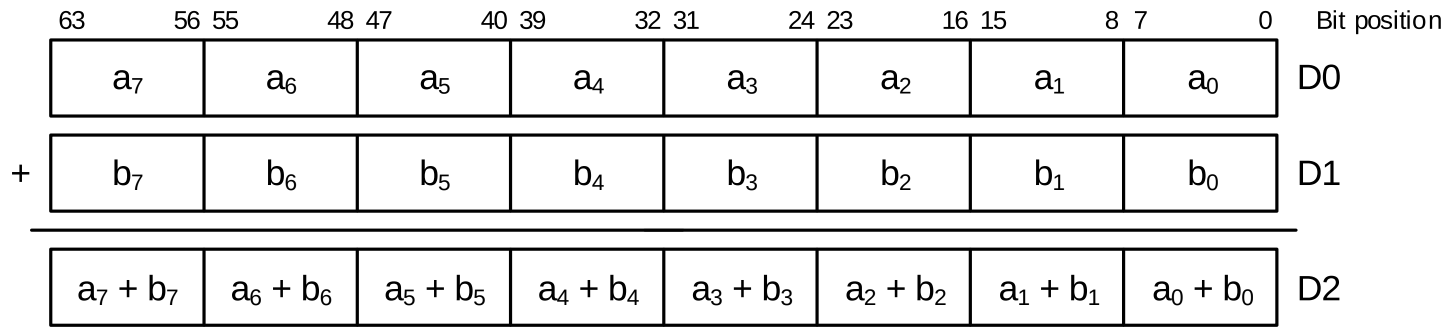
How Dynamic scheduling and register renaming work

1. Multiple instructions are fetched at once and sent to the instruction queue where they are decoded.
2. Instructions are distributed to different places such as address registers and reservation stations.
3. If the instruction is a load instruction it can proceed to the address unit then to the load buffer and wait to access the data memory based on the memory availability.
4. If the instruction is a store instruction it can proceed to the address unit but then wait in the store buffer for the data to be updated correctly based on the common data bus and the memory to be available.
5. If the instruction is an add or multiply it goes to the reservation station and wait for the functional unit to be ready and the operands to be updated form the common data bus.
6. Finally the common data bus will update the regular FP register file.

Note : the instruction is considered issued if it is being processed in its respective FU.

SIMD

- **Single Instruction Multiple Data** (SIMD)
  - Single instruction **acts on multiple pieces of data at once**
  - Common application: **graphics**
  - Can apply to short arithmetic operations (also called *packed arithmetic*)
- For example, add eight 8-bit elements



63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
Bit position
a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
D0
+
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
D1
$a_7 + b_7$ $a_6 + b_6$ $a_5 + b_5$ $a_4 + b_4$ $a_3 + b_3$ $a_2 + b_2$ $a_1 + b_1$ $a_0 + b_0$
D2