# 361.1.4201
# Computer Architecture
# Out-of-Order Execution
# (Dynamic Instruction Scheduling)
# Dr. Guy Tel-Zur

Based on slides by Prof. Onur Mutlu

Carnegie Mellon University

Spring 2015

With Dr. Guy Tel-Zur & Danny Sidner's modifications

Last update: 17/6/2021 23/6/2022, 7/6/2023

# Agenda for Today & Next Few Lectures

- Single-cycle Microarchitectures
- Multi-cycle and Microprogrammed Microarchitectures
- Pipelining
- Issues in Pipelining: Control & Data Dependence Handling, State Maintenance and Recovery, …
- **Out-of-Order Execution**
- Issues in OoO Execution: Load-Store Handling, …
- Alternative Approaches to Instruction Level Parallelism

# Readings Specifically for Today

- Smith and Sohi, "The Microarchitecture of Superscalar Processors," Proceedings of the IEEE, 1995
  - More advanced pipelining
  - Interrupt and exception handling
  - Out-of-order and superscalar execution concepts

- Kessler, "The Alpha 21264 Microprocessor," IEEE Micro 1999.

# Recap of Last Lecture

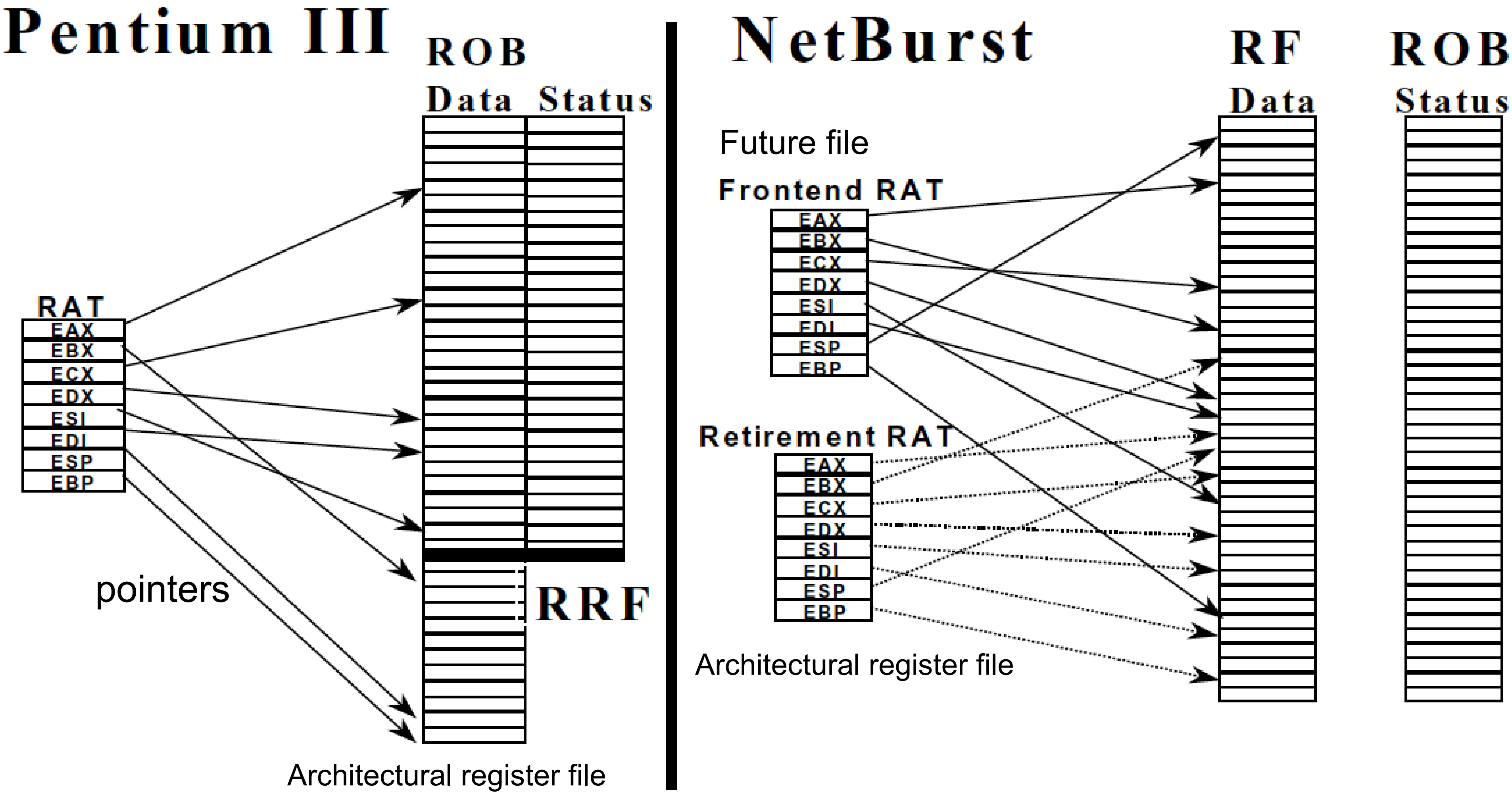
- Exceptions vs. Interrupts
- Precise Exceptions/Interrupts
- Why Do We Want Precise Exceptions?
- How Do We Ensure Precise Exceptions?
  - Reorder buffer
  - History buffer
  - Future register file (best of both worlds)
  - Checkpointing
- Register renaming with a reorder buffer
- How to Handle Exceptions
- How to Handle Branch Mispredictions
- Speed of State Recovery: Recovery and Interrupt Latency
  - Checkpointing
- Registers vs. Memory

# Important: Register Renaming with a Reorder Buffer

- Output and anti dependencies are not true dependencies
  - WHY? The same register refers to values that have nothing to do with each other
  - **They exist due to lack of register ID's (i.e. names) in the ISA**
- The register ID is **renamed** to the reorder buffer entry that will hold the register's value
  - Register ID → ROB entry ID
  - Architectural register ID → Physical register ID
  - After renaming, ROB entry ID used to refer to the register

- This eliminates anti- and output- dependencies
  - Gives the illusion that there are a large number of registers

# Review: Register Renaming Examples



Boggs et al., "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

# Review: Checkpointing Idea

- Goal: Restore the frontend state (future file) such that the correct next instruction after the branch can execute right away after the branch misprediction is resolved

שחזור ה-FF כך שההוראה, הבאה במקרה של חזוי שגוי של הסתעפות, תוכל להתבצע במיידי

- Idea: Checkpoint the frontend register state/map at the time a branch is decoded and keep the checkpointed state updated with results of instructions older than the branch
  - Upon branch misprediction, restore the checkpoint associated with the branch

השיטה: גיבוי (checkpoint) של ה- FF בזמן פענוח ההסתעפות כדי שיהיה ניתן לחזור לשם, כפי שנכתב למעלה, באמצעות שחזור הגיבוי במידה ומתברר שחיזוי ההסתעפות היה שגוי

- Hwu and Patt, "Checkpoint Repair for Out-of-order Execution Machines," ISCA 1987.

# Review: Checkpointing

- When a branch is decoded
  - Make a copy of the future file/map and associate it with the branch

- When an instruction produces a register value
  - All future file/map checkpoints that are younger than the instruction are updated with the value

- When a branch misprediction is detected
  - Restore the checkpointed future file/map for the mispredicted branch when the branch misprediction is resolved
  - Flush instructions in pipeline younger than the branch
  - Deallocate checkpoints younger than the branch

# Review: Registers versus Memory

- So far, we considered mainly registers as part of state

- What about memory?

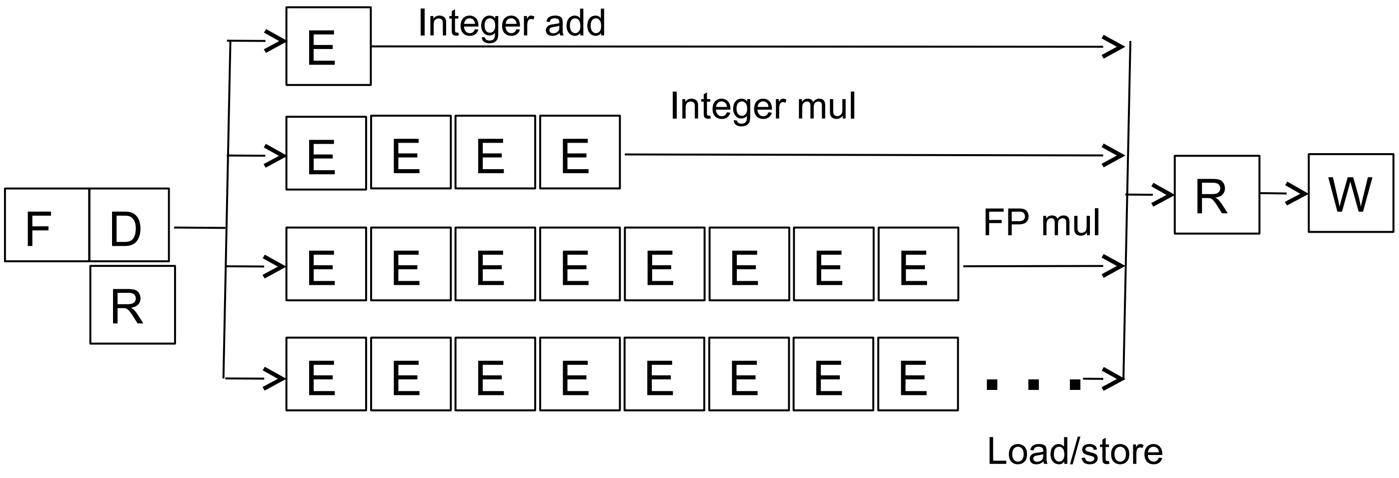
- What are the fundamental differences between registers and memory?
  - Register dependences known statically – memory dependences determined dynamically
  - Register state is small – memory state is large
  - Register state is not visible to other threads/processors – memory state is shared between threads/processors (in a shared memory multiprocessor)

# Maintaining Speculative Memory State: Stores

- Handling out-of-order completion of memory operations
  - UNDOing a memory write more difficult than UNDOing a register write. Why? קשה יותר לעשות UNDO לזיכרון מאשר לרגיסטר
  - One idea: Keep store address/data in reorder buffer
    - How does a load instruction find its data?
  - Store/write buffer: Similar to reorder buffer, but used only for store instructions
    - Program-order list of un-committed store operations
    - When store is decoded: Allocate a store buffer entry
    - When store address and data become available: Record in store buffer entry
    - When the store is the oldest instruction in the pipeline: Update the memory address (i.e. cache) with store data

# Review: In-Order Pipeline with Reorder Buffer

- Decode (D): Access regfile/ROB, allocate entry in ROB, check if instruction can execute, if so **dispatch** instruction (send to functional unit)
- Execute (E): Instructions can complete out-of-order
- Completion (R): Write result to reorder buffer
- Retirement/Commit (W): Check for exceptions; if none, write result to architectural register file or memory; else, flush pipeline and start from exception handler
- In-order dispatch/execution, out-of-order completion, in-order retirement



# Remember:
# Static vs. Dynamic Scheduling

# Remember: Questions to Ponder

- What is the role of the hardware vs. the software in the order in which instructions are executed in the pipeline?
  - Software based instruction scheduling → static scheduling
  - Hardware based instruction scheduling → dynamic scheduling

- What information does the compiler not know that makes static scheduling difficult?
  - Answer: Anything that is determined at run time
    - Variable-length operation latency, memory addr, branch direction

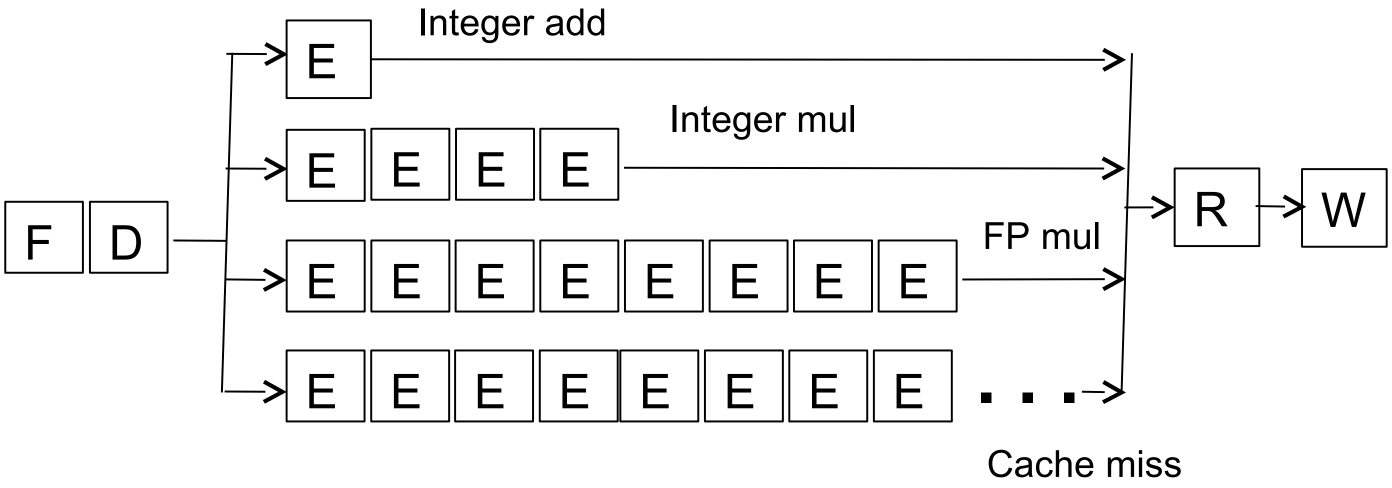
# Dynamic Instruction Scheduling

- Hardware has knowledge of dynamic events on a per-instruction basis (i.e., at a very fine granularity)
  - Cache misses
  - Branch mispredictions
  - Load/store addresses

- Wouldn't it be nice if hardware did the scheduling of instructions?

# Out-of-Order Execution (Dynamic Instruction Scheduling)

# An In-order Pipeline



- Problem: A true data dependency stalls dispatch of younger instructions into functional (execution) units
- Dispatch: Act of sending an instruction to a functional unit
- אנחנו לא רוצים שתהיה עצירה, stall, של ה-Dispatch

# Can We Do Better?

- What do the following two pieces of code have in common (with respect to execution in the previous design)?

```
IMUL  R3 ← R1, R2
ADD   R3 ← R3, R1
ADD   R1 ← R6, R7
IMUL  R5 ← R6, R8
ADD   R7 ← R9, R9
```

```
LD    R3 ← R1 (0)
ADD   R3 ← R3, R1
ADD   R1 ← R6, R7
IMUL  R5 ← R6, R8
ADD   R7 ← R9, R9
```

- Answer: First ADD stalls the whole pipeline!
  - ADD cannot dispatch because its source registers unavailable
  - Later **independent** instructions cannot get executed

- How are the above code portions different?
  - Answer: Load latency is variable (unknown until runtime)
  - What does this affect? Think compiler vs. microarchitecture

# Preventing Dispatch Stalls

- Multiple ways of doing it
- You have already seen at least THREE:
  - 1.
  - 2.
  - 3.
- What are the disadvantages of the above three?
- Any other way to prevent dispatch stalls?
  - Actually, you have briefly seen the basic idea before
    - Dataflow: fetch and "fire" an instruction when its inputs are ready
  - Problem: in-order dispatch (scheduling, or execution)
  - Solution: out-of-order dispatch (scheduling, or execution)

# Preventing Dispatch Stalls

- Multiple ways of doing it
- You have already seen at least THREE:
  - 1. Fine-grained multithreading
  - 2. Value prediction
  - 3. Compile-time instruction scheduling/reordering
- What are the disadvantages of the above three?

- Any other way to prevent dispatch stalls?
  - Actually, you have briefly seen the basic idea before
    - Dataflow: fetch and "fire" an instruction when its inputs are ready
  - Problem: in-order dispatch (scheduling, or execution)
  - Solution: out-of-order dispatch (scheduling, or execution)

# Out-of-order Execution (Dynamic Scheduling)

- Idea: Move the dependent instructions out of the way of independent ones (such that independent ones can execute)
  - Rest areas for dependent instructions: “Reservation stations"

- Monitor the source “values” of each instruction in the resting area
- When all source “values” of an instruction are available, “fire” (i.e. dispatch) the instruction
  - Instructions dispatched in dataflow (not control-flow) order

נדרשת תוספת לוגיקה

תזכורת: data flow model

- Benefit:
  - Latency tolerance: Allows independent instructions to execute and complete in the presence of a long latency operation

יתרון: מנצלים לטובתנו את השהות

# In-order vs. Out-of-order Dispatch

- In order dispatch + precise exceptions:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | E | E | E | E | R | W | | | | | | | |
| | F | D | STALL | | | E | R | W | | | | | | |
| | | F | STALL | | | D | E | R | W | | | | | |
| | | | | | | F | D | E | E | E | E | R | W | |
| | | | | | | | F | D | STALL | | | E | R | W |

IMUL R3 ← R1, R2
ADD R3 ← R3, R1
ADD R1 ← R6, R7
IMUL R5 ← R6, R8
ADD R7 ← R3, R5

- Out-of-order dispatch + precise exceptions:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | E | E | E | E | R | W | | | | |
| | F | D | WAIT | | | E | R | W | | | |
| | | F | D | E | R | | | | W | | |
| | | | F | D | E | E | E | E | R | W | |
| | | | | F | D | | WAIT | | E | R | W |

- 15 vs. 12 cycles

# Enabling OoO Execution

1. Need to link the consumer of a value to the producer הצורך
   - Register renaming: Associate a “tag” with each data value המענה
2. Need to buffer instructions until they are ready to execute הצורך
   - Insert instruction into reservation stations after renaming המענה
3. Instructions need to keep track of readiness of source values הצורך
   - Broadcast the “tag” when the value is produced המענה
   - Instructions compare their “source tags” to the broadcast tag → if match, source value becomes ready
4. When all source values of an instruction are ready, need to dispatch the instruction to its functional unit (FU) הצורך
   - Instruction wakes up if all sources are ready המענים
   - If multiple instructions are awake, need to select one per FU

# Tomasulo’s Algorithm

- OoO with register renaming invented by Robert Tomasulo
  - Used in IBM 360/91 Floating Point Units
  - **Read:** Tomasulo, “An Efficient Algorithm for Exploiting Multiple Arithmetic Units,” IBM Journal of R&D, Jan. 1967. - ראו השקף הבא

    המאמר רשות ואינו חובה
- What is the major difference today?
  - Precise exceptions: IBM 360/91 did NOT have this
  - Patt, Hwu, Shebanow, “HPS, a new microarchitecture: rationale and introduction,” MICRO 1985.
  - Patt et al., “Critical issues regarding HPS, a high performance microarchitecture,” MICRO 1985.
- Variants are used in most high-performance processors
  - Initially in Intel Pentium Pro, AMD K5
  - Alpha 21264, MIPS R10000, IBM POWER5, IBM z196, Oracle UltraSPARC T4, ARM Cortex A15

# Robert (Bob) Tomasulo

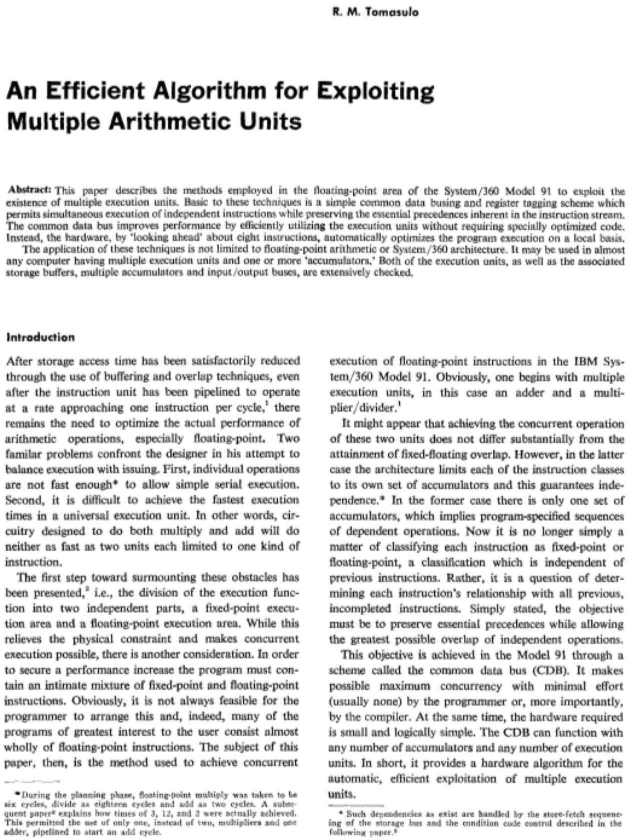
Robert (Bob) Tomasulo joined IBM research in 1956 after graduating from Manhattan College. After nearly a decade gaining broad experience in a variety of technical and leadership roles, he transitioned to mainframe development, including the IBM System/360 Model 91 and its successors. Following his 25 year career with IBM, Tomasulo worked on an incubator project at Storage Technology Corporation to develop the first CMOS-based mainframe system; co-founded, a startup to develop one of the earliest microprocessor-based server systems; and worked as a consultant on processor architecture and for Amdahl Consulting.

On 30 January 2008, Tomasulo spoke at the University of Michigan College of Engineering about his career and the history and development of out-of-order execution. View the seminar. believed to be his last public appearance. (source: https://www.computer.org/profiles/robert-tomasulo)

His last public lecture from 2008:

http://leccap.engin.umich.edu/leccap/viewer/r/pvSbKs

https://www.cs.virginia.edu/~evans/greatworks/tomasulo.pdf



R. M. Tomasulo

## An Efficient Algorithm for Exploiting Multiple Arithmetic Units

**Abstract:** This paper describes the methods employed in the floating-point area of the System/360 Model 91 to exploit the existence of multiple execution units. Basic to these techniques is a simple common data busing and register tagging scheme which permits simultaneous execution of independent instructions while preserving the essential precedences inherent in the instruction stream. The common data bus improves performance by efficiently utilizing the execution units without requiring specially optimized code. Instead, the hardware, by 'looking ahead' about eight instructions, automatically optimizes the program execution on a local basis.

The application of these techniques is not limited to floating-point arithmetic or System/360 architecture. It may be used in almost any computer having multiple execution units and one or more 'accumulators.' Both of the execution units, as well as the associated storage buffers, multiple accumulators and input/output buses, are extensively checked.

### Introduction

After storage access time has been satisfactorily reduced through the use of buffering and overlap techniques, even after the instruction unit has been pipelined to operate at a rate approaching one instruction per cycle,[1] there remains the need to optimize the actual performance of arithmetic operations, especially floating-point. Two familar problems confront the designer in his attempt to balance execution with issuing. First, individual operations are not fast enough* to allow simple serial execution. Second, it is difficult to achieve the fastest execution times in a universal execution unit. In other words, circuitry designed to do both multiply and add will do neither as fast as two units each limited to one kind of instruction.

The first step toward surmounting these obstacles has been presented,[2] i.e., the division of the execution function into two independent parts, a fixed-point execution area and a floating-point execution area. While this relieves the physical constraint and makes concurrent execution possible, there is another consideration. In order to secure a performance increase the program must contain an intimate mixture of fixed-point and floating-point instructions. Obviously, it is not always feasible for the programmer to arrange this and, indeed, many of the programs of greatest interest to the user consist almost wholly of floating-point instructions. The subject of this paper, then, is the method used to achieve concurrent execution of floating-point instructions in the IBM System/360 Model 91. Obviously, one begins with multiple execution units, in this case an adder and a multiplier/divider.[1]

It might appear that achieving the concurrent operation of these two units does not differ substantially from the attainment of fixed-floating overlap. However, in the latter case the architecture limits each of the instruction classes to its own set of accumulators and this guarantees independence.* In the former case there is only one set of accumulators, which implies program-specified sequences of dependent operations. Now it is no longer simply a matter of classifying each instruction as fixed-point or floating-point, a classification which is independent of previous instructions. Rather, it is a question of determining each instruction's relationship with all previous, incompleted instructions. Simply stated, the objective must be to preserve essential precedences while allowing the greatest possible overlap of independent operations.

This objective is achieved in the Model 91 through a scheme called the common data bus (CDB). It makes possible maximum concurrency with minimal effort (usually none) by the programmer or, more importantly, by the compiler. At the same time, the hardware required is small and logically simple. The CDB can function with any number of accumulators and any number of execution units. In short, it provides a hardware algorithm for the automatic, efficient exploitation of multiple execution units.

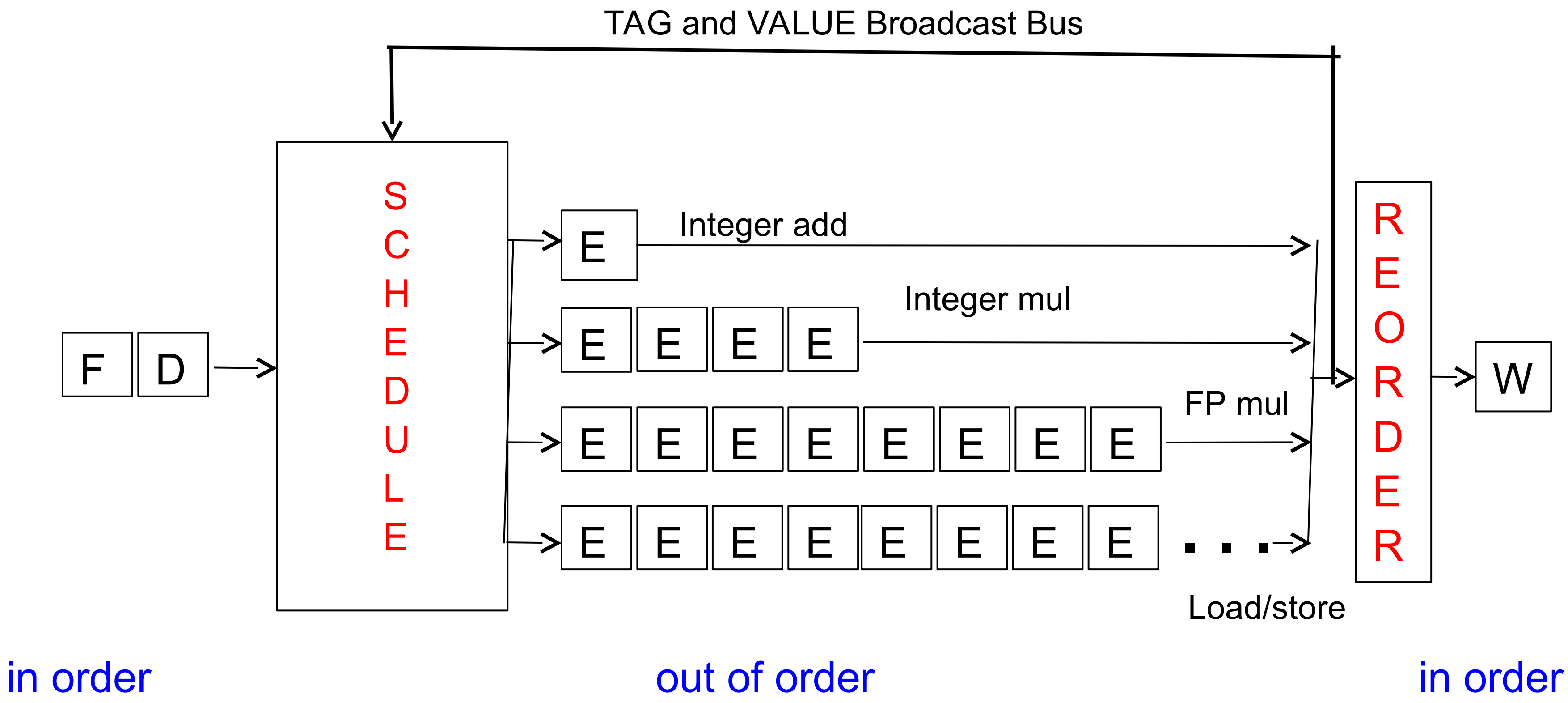
*During the planning phase, floating-point multiply was taken to be six cycles, divide as eighteen cycles and add as two cycles. A subsequent paper[3] explains how times of 3, 12, and 2 were actually achieved. This permitted the use of only one, instead of two, multipliers and one adder, pipelined to start an add cycle.

* Such dependencies as exist are handled by the store-fetch sequencing of the storage bus and the condition code control described in the following paper.[3]

# קריאה נוספת אודות Tomasulo’s Algorithm

H&P CAQA 3.4-3.6

# Two Humps in a  Modern Pipeline



- Hump 1: Reservation stations (scheduling window) כל ההוראות נכנסות
לכאן, בשונה ממה שנאמר קודם שרק ההוראות התלויות
- Hump 2: Reordering (reorder buffer, aka instruction window or active window)

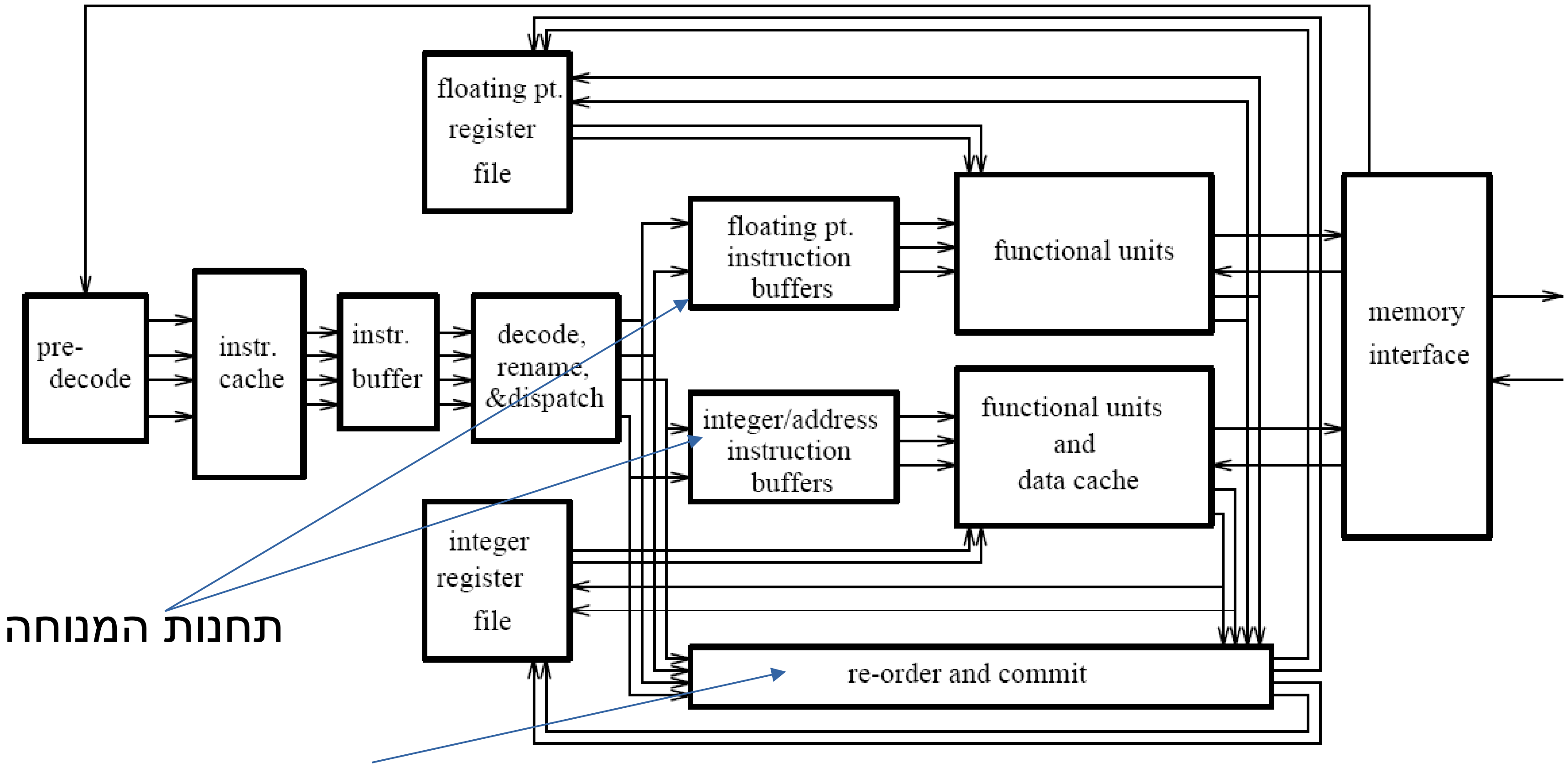
# Two Humps in a Modern Pipeline

active window)

שתי הדבשות

Photo credit: http://true-wildlife.blogspot.ch/2010/10/bactrian-camel.html

# General Organization of an OOO Processor

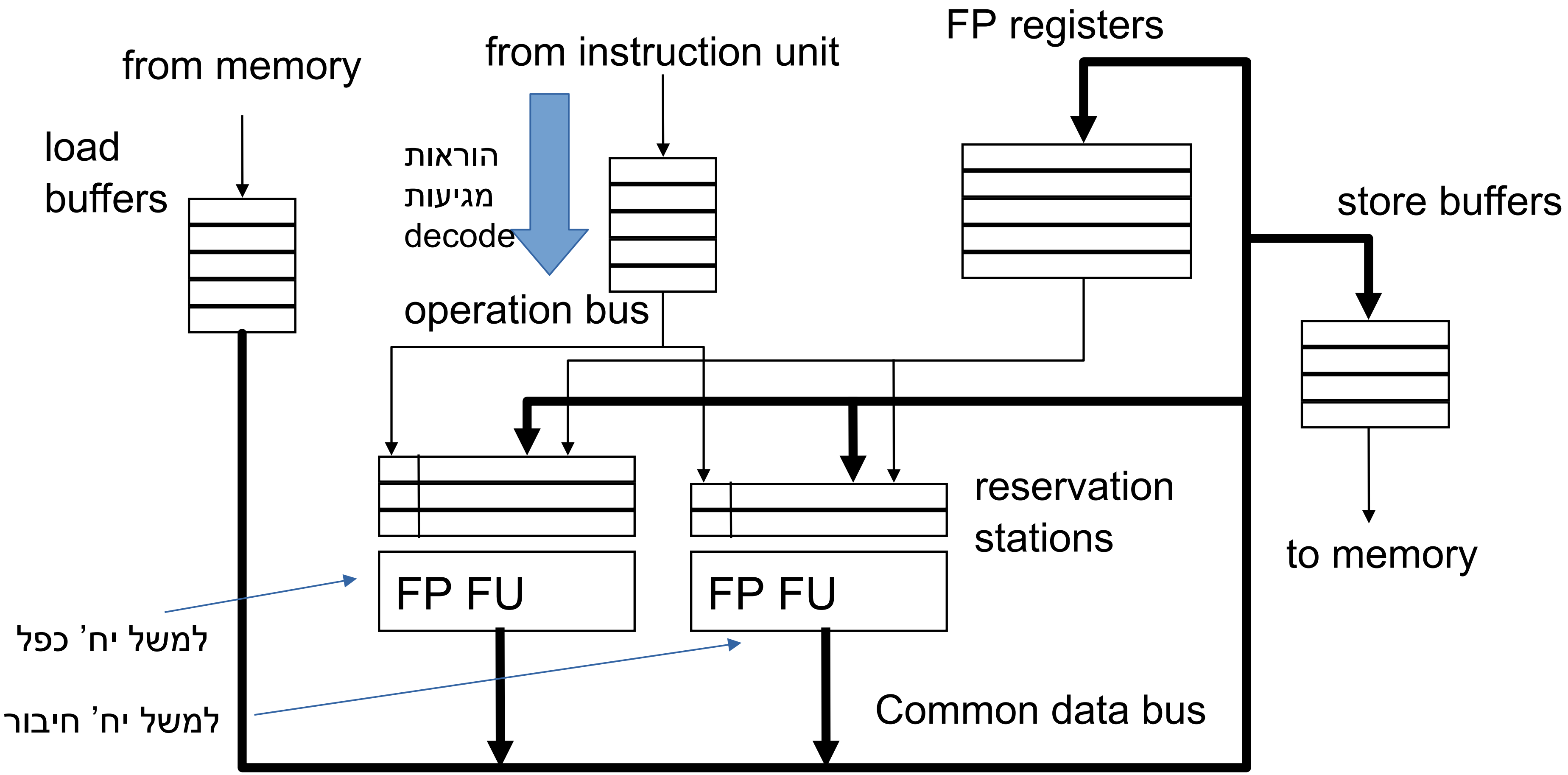

floating pt. register file
pre- decode
instr. cache
instr. buffer
decode, rename, &dispatch
floating pt. instruction buffers
functional units
memory interface
integer/address instruction buffers
functional units and data cache
integer register file
re-order and commit

תחנות המנוחה

תחנת ה- reorder

- Smith and Sohi, “The Microarchitecture of Superscalar Processors,” Proc. IEEE, Dec. 1995.

# Tomasulo’s Machine: IBM 360/91



FP=Floating Point, FU=Functional Unit, CDB = Common Data Bus :ראשי תיבות

# Tomasulo’s Algorithm

- If reservation station available before renaming אם התחנה זמינה – מלא אותה
  - Instruction + renamed operands (source value/tag) inserted into the reservation station
  - Only rename if reservation station is available
- Else stall אם לא, עצור
- While in reservation station, each instruction: כל עוד שוהה הוראה בתחנה
  - Watches common data bus (CDB) for tag of its sources
  - When tag seen, grab value for the source and keep it in the reservation station
  - When both operands available, instruction ready to be dispatched
- Dispatch instruction to the Functional Unit when instruction is ready
- After instruction finishes in the Functional Unit
  - Arbitrate (לברור) for CDB
  - Put tagged value onto CDB (tag broadcast)
  - Register file is connected to the CDB
    - Register contains a tag indicating the latest writer to the register
    - If the tag in the register file matches the broadcast tag, write broadcast value into register (and set valid bit)
  - Reclaim rename tag
    - no valid copy of tag in system!

# Register Renaming

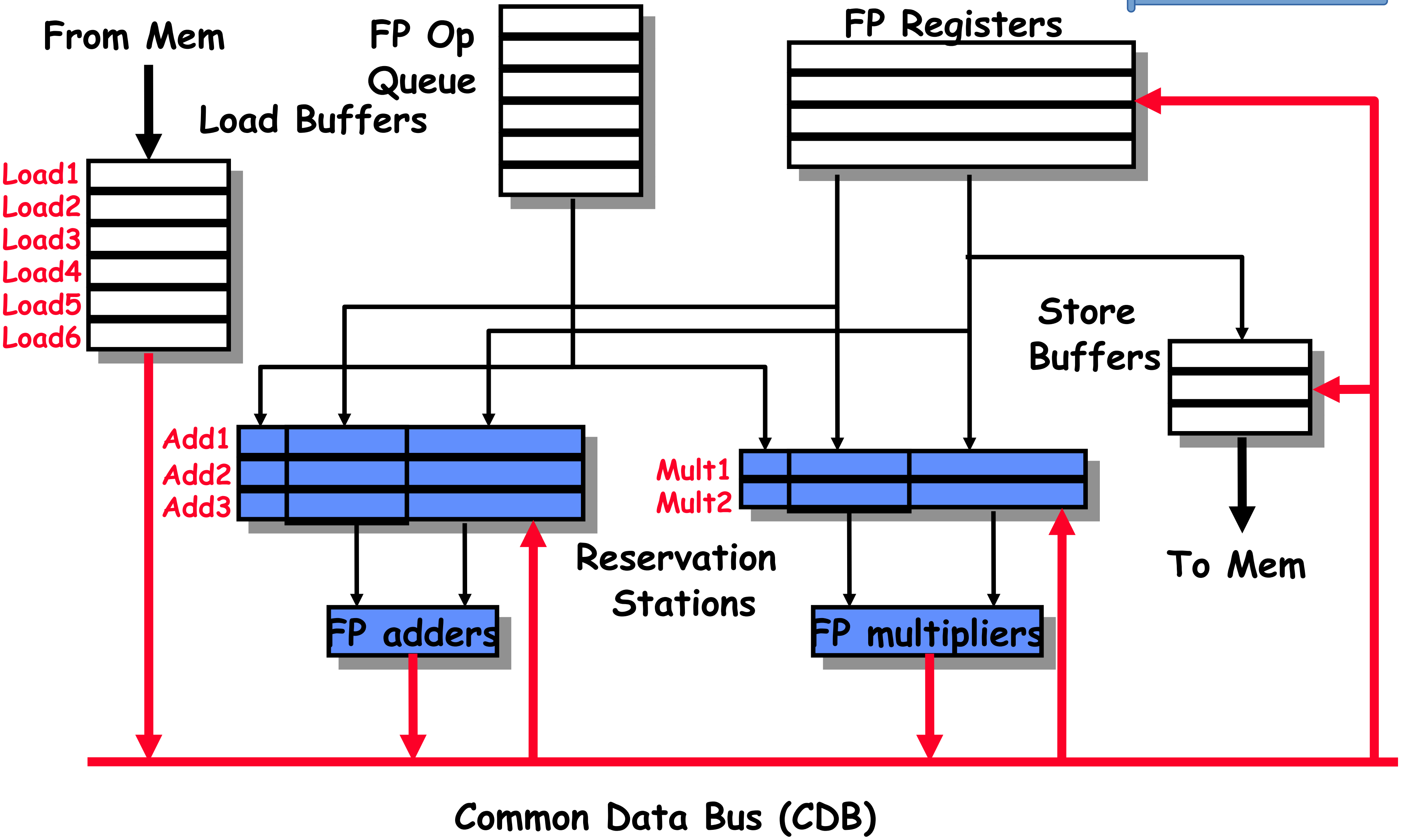
- Output and anti dependencies are not true dependencies
  - WHY? The same register refers to values that have nothing to do with each other
  - **They exist because not enough register ID’s (i.e. names) in the ISA**
- The register ID is renamed to the reservation station entry that will hold the register’s value
  - Register ID → RS entry ID
  - Architectural register ID → Physical register ID
  - After renaming, RS entry ID used to refer to the register
- This eliminates anti- and output- dependencies
  - Approximates the performance effect of a large number of registers even though ISA has a small number

דוגמה ראשונית לחימום...

# Tomasulo Organization

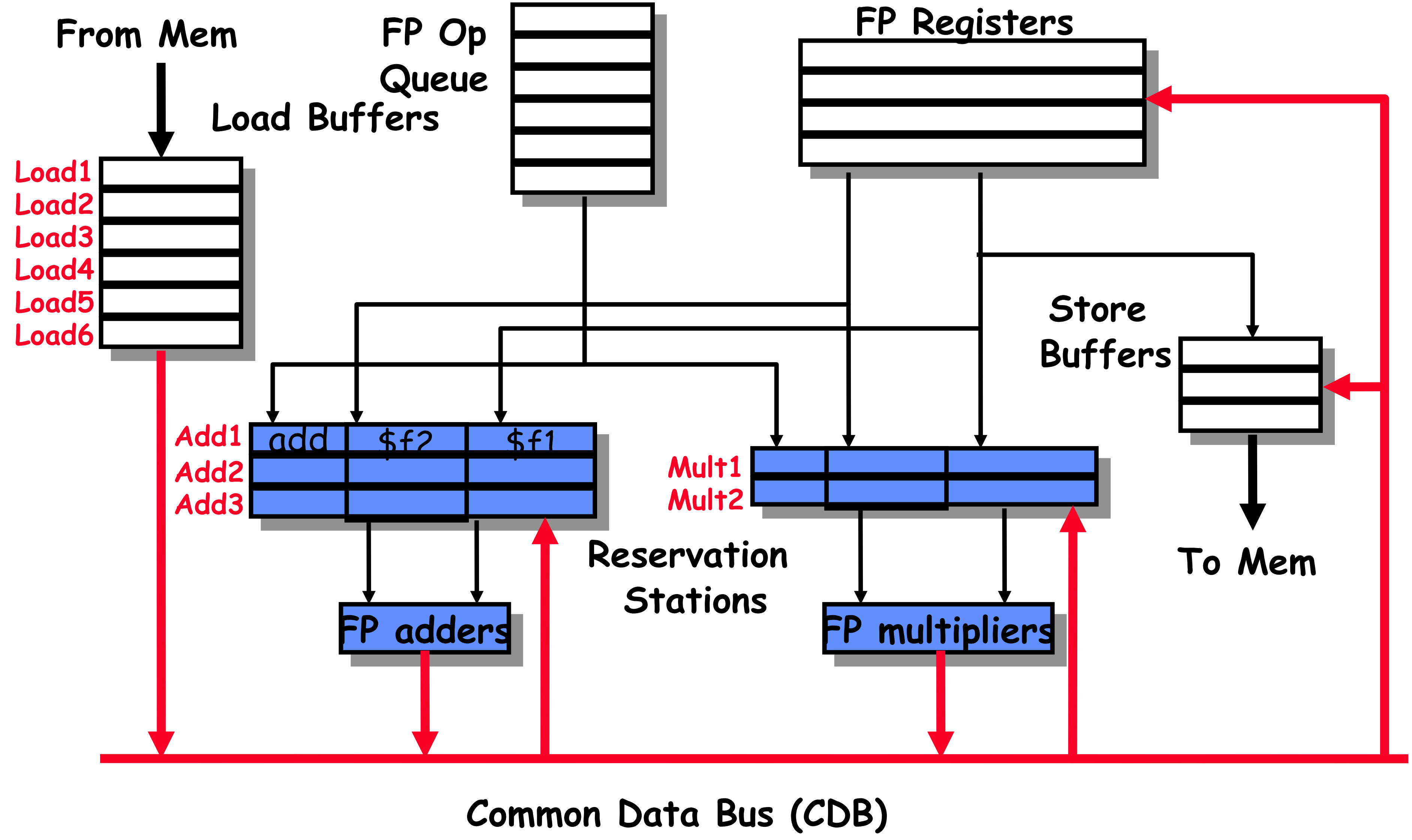
add $f7,$f2,$f1
mult $f4,$f7,$f3
sub $f5,$f0,$f4

אנימציה



→ add $f7,$f2,$f1
mult $f4,$f7,$f3
sub $f5,$f0,$f4

# Tomasulo Organization

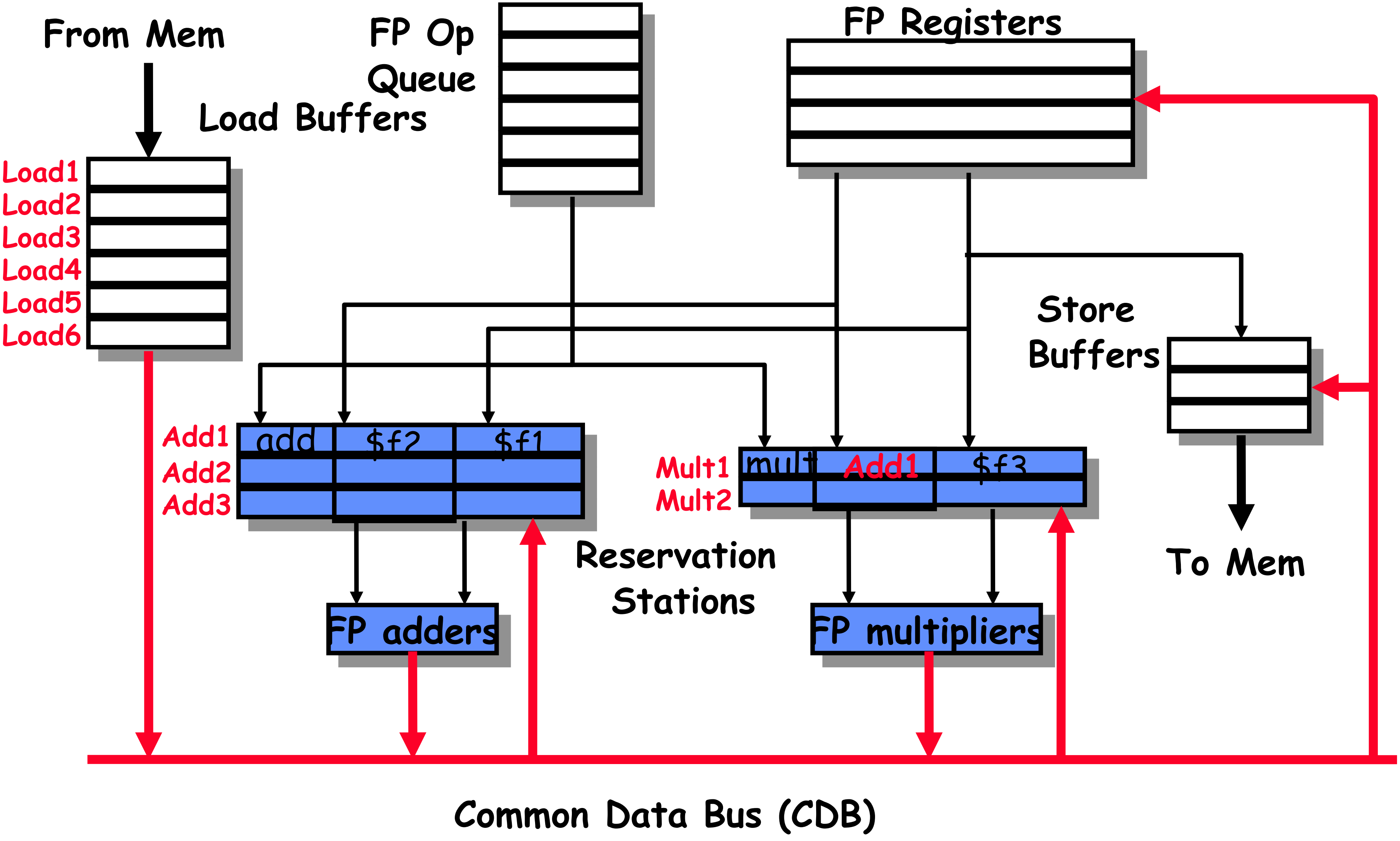


# Tomasulo Organization

add $f7,$f2,$f1

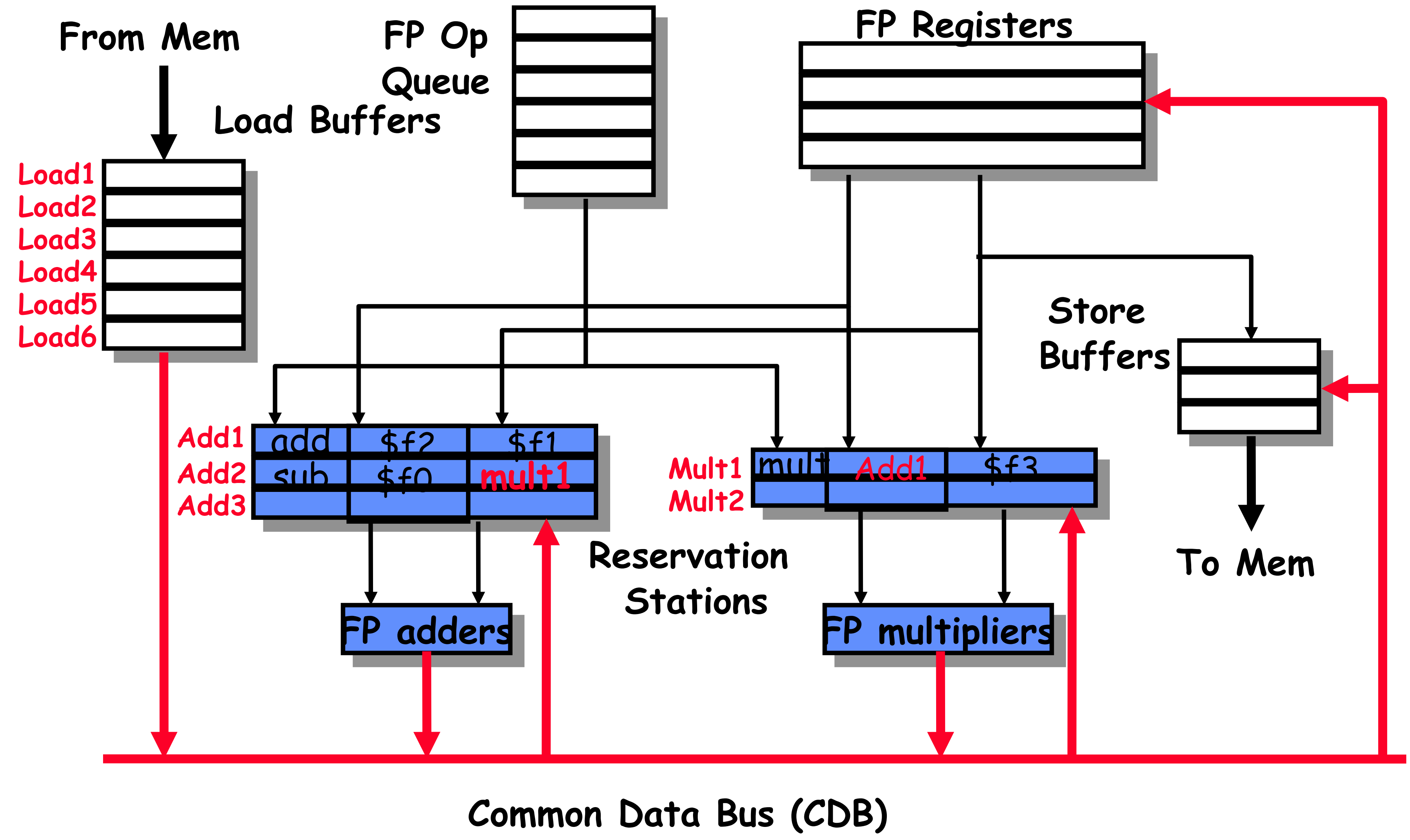
→ mult $f4,$f7,$f3

sub $f5,$f0,$f4



# Tomasulo Organization

```
  add $f7,$f2,$f1
  mult $f4,$f7,$f3
→ sub $f5,$f0,$f4
```



From Mem

FP Op Queue

Load Buffers

FP Registers

Load1
Load2
Load3
Load4
Load5
Load6

Store Buffers

Add1 add $f2 $f1
Add2 sub $f0 mult1
Add3

Mult1 mult Add1 $f3
Mult2

To Mem

Reservation Stations

FP adders

FP multipliers

Common Data Bus (CDB)

אנימציה

# Reservation Station Components

Op: Operation to perform in the unit (e.g., add or sub)

Vj, Vk: Value of Source operands

- Store buffers has one V field, =result to be stored

Qj, Qk: Specifies the Reservation Stations that supposed to produce the source registers (= the values to be written into Vj,Vk)

- Note: Qj,Qk=0 means ready (Values in Vj,Vk are valid)
- Store buffers only have Qi, Vi for RS producing result

Busy: Indicates reservation station or FU is busy

Register Status - Qi: Indicates which Reservation station is supposed to write to the register.

Regs[i]: the Value of the register. The value is correct when Qi=0 means that there are no pending instructions that will write to that register.

אנימציה

# Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station free (no structural hazard),
   control issues instruction & sends operands (=renames the registers).

2. Execute—operate on operands (EX)

   When both operands ready then execute;
   if not ready, watch Common Data Bus for result

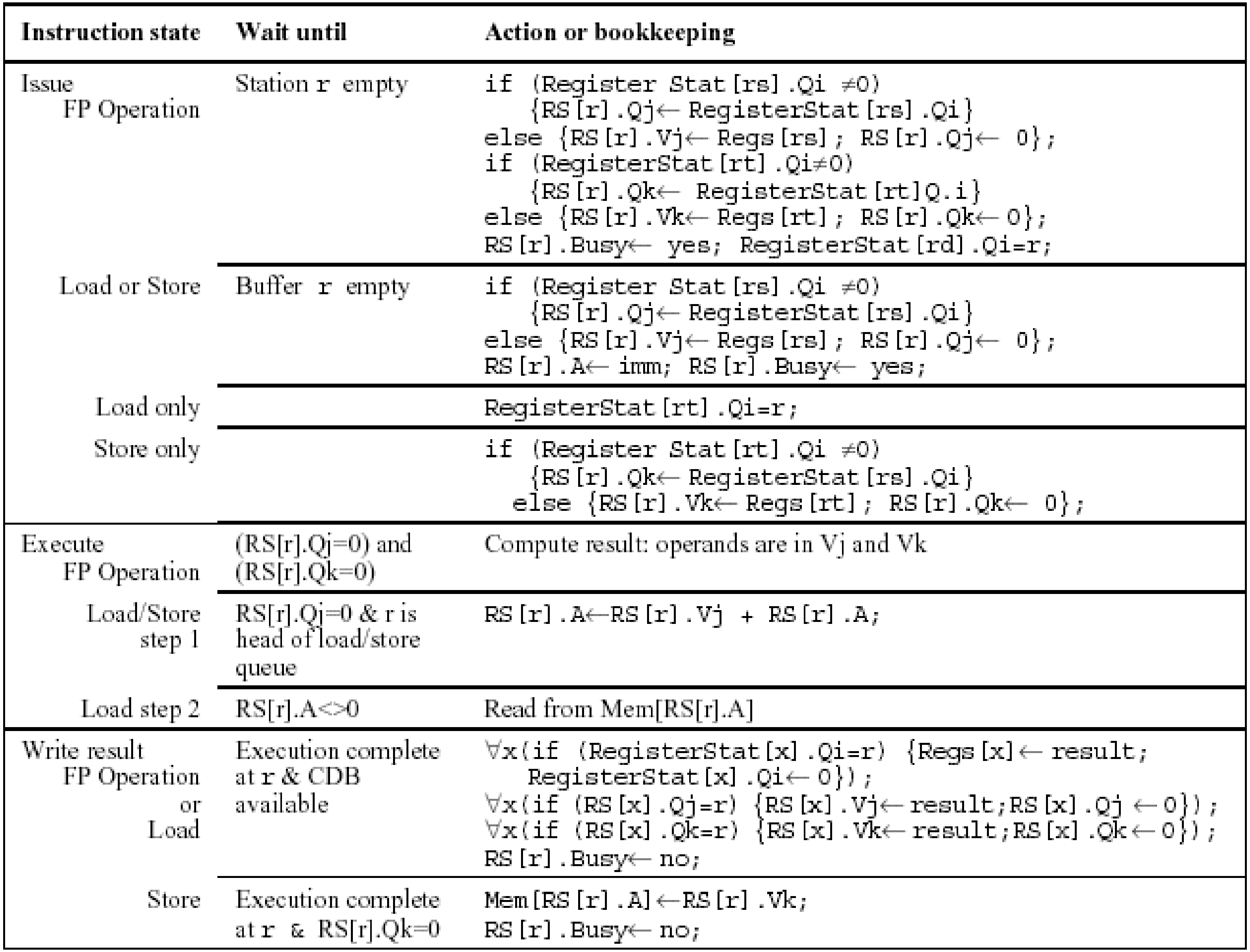
3. Write result—finish execution (WB)

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

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits of data + 4 bits of Functional Unit source address
  - Write if matches expected Functional Unit (produces result)
  - Does the broadcast

- The units speed in the following example is:
  3 clocks for FP add/sub; 10 for mult; 40 clks for divide

# Details of Tomasulo Algorithm



| Instruction state | Wait until | Action or bookkeeping |
|---|---|---|
| Issue<br>FP Operation | Station r empty | if (Register Stat[rs].Qi ≠0)<br>{RS[r].Qj← RegisterStat[rs].Qi}<br>else {RS[r].Vj← Regs[rs]; RS[r].Qj← 0};<br>if (RegisterStat[rt].Qi≠0)<br>{RS[r].Qk← RegisterStat[rt]Q.i}<br>else {RS[r].Vk← Regs[rt]; RS[r].Qk← 0};<br>RS[r].Busy← yes; RegisterStat[rd].Qi=r; |
| Load or Store | Buffer r empty | if (Register Stat[rs].Qi ≠0)<br>{RS[r].Qj← RegisterStat[rs].Qi}<br>else {RS[r].Vj← Regs[rs]; RS[r].Qj← 0};<br>RS[r].A← imm; RS[r].Busy← yes; |
| Load only | | RegisterStat[rt].Qi=r; |
| Store only | | if (Register Stat[rt].Qi ≠0)<br>{RS[r].Qk← RegisterStat[rs].Qi}<br>else {RS[r].Vk← Regs[rt]; RS[r].Qk← 0}; |
| Execute<br>FP Operation | (RS[r].Qj=0) and<br>(RS[r].Qk=0) | Compute result: operands are in Vj and Vk |
| Load/Store<br>step 1 | RS[r].Qj=0 & r is head of load/store queue | RS[r].A←RS[r].Vj + RS[r].A; |
| Load step 2 | RS[r].A<>0 | Read from Mem[RS[r].A] |
| Write result<br>FP Operation<br>or<br>Load | Execution complete at r & CDB available | ∀x(if (RegisterStat[x].Qi=r) {Regs[x]← result;<br>RegisterStat[x].Qi← 0});<br>∀x(if (RS[x].Qj=r) {RS[x].Vj← result;RS[x].Qj ← 0});<br>∀x(if (RS[x].Qk=r) {RS[x].Vk← result;RS[x].Qk ← 0});<br>RS[r].Busy← no; |
| Store | Execution complete at r & RS[r].Qk=0 | Mem[RS[r].A]←RS[r].Vk;<br>RS[r].Busy← no; |

# Details of Tomasulo Algorithm

**FIGURE 3.5 Steps in the algorithm and what is required for each step.** For the issuing instruction, **rd** is the destination,**rs** and **rt** are the source register numbers, **imm** is the sign-extended immediate field, and **r** is the reservation station or buffer that the instruction is assigned to. **RS** is the reservation-station data structure. The value returned by a FP unit or by the load unit is called **result**. **RegisterStat** is the register status data structure (not the register file, which is **Regs[ ]** ). When an instruction is issued, the destination register has its **Qi** field set to the number of the buffer or reservation station to which the instruction is issued. If the operands are available in the registers, they are stored in the **V** fields. Otherwise,the **Q** fields are set to indicate the reservation station that will produce the values needed as source operands. The instruction waits at the reservation station until both its operands are available, indicated by zero in the **Q** fields. The **Q** fields are set to zero either when this instruction is issued, or when an instruction on which this instruction depends completes and does its write back. When an instruction has finished execution and the **CDB** is available, it can do its write back. All the buffers, registers, and reservation stations whose value of **Qj** or **Qk** is the same as the completing reservation station update their values from the **CDB** and mark the **Q** fields to indicate that values have been received. Thus, the **CDB** can broadcast its result to many destinations in a single clock cycle, and if the waiting instructions have their operands, they can all begin execution on the next clock cycle. Loads go through two steps in Execute, and stores perform slightly differently during Write Result, where they may have to wait for the value to store. Remember that to preserve exception behavior, instructions should not be allowed to execute if a branch that is earlier in program order has not yet completed. Because any concept of program order is not maintained after the Issue stage, this restriction is usually implemented by preventing any instruction from leaving the Issue step, if there is a pending branch already in the pipeline. Later, we will see how speculation support removes this .restriction

נביא עתה דוגמה ראשונה לאופן פעולת האלגוריתם של טומסולו:

# Tomasulo’s algorithm
# Example #1

אנימציה

# Tomasulo Example

Instruction stream
Shown for ease –
(not a part of the system)

The "system":
Everything below
the black line

*Instruction status:*

| Instruction | | j | k | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | | | |
| LD | F2 | 45+ | R3 | | | |
| MULTD | F0 | F2 | F4 | | | |
| SUBD | F8 | F6 | F2 | | | |
| DIVD | F10 | F0 | F6 | | | |
| ADDD | F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

3 Load/Buffers

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

FU count
down

3 FP Adder R.S.
2 FP Mult R.S.

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | *FU* | | | | | | | | | |

Clock cycle
counter

# Tomasulo Example Cycle 1

## issue LD1

## *Instruction status:*

|  |  |  |  |  | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction |  | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD | F6 | 34+ | R2 | 1 |  |  |
| LD | F2 | 45+ | R3 |  |  |  |
| MULTD | F0 | F2 | F4 |  |  |  |
| SUBD | F8 | F6 | F2 |  |  |  |
| DIVD | F10 | F0 | F6 |  |  |  |
| ADDD | F6 | F8 | F2 |  |  |  |

|  | Busy | Address |
|---|---|---|
| Load1 | Yes | 34,R2 |
| Load2 | No |  |
| Load3 | No |  |

## *Reservation Stations:*

|  |  |  |  | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
|  | Add1 | No |  |  |  |  |  |
|  | Add2 | No |  |  |  |  |  |
|  | Add3 | No |  |  |  |  |  |
|  | Mult1 | No |  |  |  |  |  |
|  | Mult2 | No |  |  |  |  |  |

## *Register result status:*

| Clock |  | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | *FU* |  |  |  | Load1 |  |  |  |  |  |

# Tomasulo Example Cycle 2

אנימציה

## issue LD2 (+calc adr of LD1)

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec*<br>*Comp* | *Write*<br>*Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | | |
| LD | F2 | 45+ | R3 | 2 | | |
| MULTD | F0 | F2 | F4 | | | |
| SUBD | F8 | F6 | F2 | | | |
| DIVD | F10 | F0 | F6 | | | |
| ADDD | F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | Yes | 34+R2 |
| Load2 | Yes | 45+R3 |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **2** | *FU* | | Load2 | | Load1 | | | | | |

**Note: Can have multiple loads outstanding**

# Tomasulo Example Cycle 2

## issue LD2 (+calc adr of LD1)

*Instruction status:*

| | | | | *Exec* | *Write* |
|---|---|---|---|---|---|
| Instruction | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD F6 | 34+ | R2 | 1 | | |
| LD F2 | 45+ | R3 | 2 | | |
| MULTD F0 | F2 | F4 | | | |
| SUBD F8 | F6 | F2 | | | |
| DIVD F10 | F0 | F6 | | | |
| ADDD F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | Yes | 34+R2 |
| Load2 | Yes | 45+R3 |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

We see that for the next instr., F4 is ready (Q=0), but F2 is waiting for the Load unit (Q=Load2)

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **2** | *FU* | 0 | Load2 | 0 | Load1 | 0 | 0 | 0 | | 0 |

# Tomasulo Example Cycle 3

אנימציה

**issue Mult, complete LD1 (Memory rd)**

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | |
| LD | F2 | 45+ | R3 | 2 | | |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | | | |
| DIVD | F10 | F0 | F6 | | | |
| ADDD | F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | Yes | 34+R2 |
| Load2 | Yes | 45+R3 |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | Yes | MULTD | ? | R(F4) | Load2 | 0 |
| | Mult2 | No | | | | | |

If now someone writes into F4, we still have its value in the Mult1 Vk register => no WAR hazard can occur !

(not in final Reg F4 or in Vk of Mult1)

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | *FU* | Mult1 | Load2 | 0 | Load1 | 0 | 0 | 0 | | 0 |

- **Note: registers names are removed ("renamed") in Reservation Stations; MULT issued**
- **Load1 completing; what waits for Load1?**

F6+Sub

# Tomasulo Example Cycle 4

אנימציה

## Issu Sub, Complete Ld2, WB LD1

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | | |
| DIVD | F10 | F0 | F6 | | | |
| ADDD | F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | Yes | 45+R3 |
| Load3 | No | |

*Reservation Stations:*

| *Time* | *Name* | *Busy* | *Op* | *S1 Vj* | *S2 Vk* | *RS Qj* | *RS Qk* |
|---|---|---|---|---|---|---|---|
| | Add1 | Yes | SUBD | M(A1) | ? | 0 | Load2 |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | Yes | MULTD | ? | R(F4) | Load2 | 0 |
| | Mult2 | No | | | | | |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | *FU* | Mult1 | Load2 | 0 | M(A1) | Add1 | 0 | 0 | | 0 |

Writing from the CDB to F6 and Add1 means Write Back to F6 and Forwarding to the FP Add unit

- **Load2 completing; what is waiting for Load2?**

F2 + Mult1+Add1

# Tomasulo Example Cycle 5

אנימציה

## Issu Div, start Execute Sub & Mult, WB LD2

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | | |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| *Time* | *Name* | *Busy* | *Op* | *S1 Vj* | *S2 Vk* | *RS Qj* | *RS Qk* |
|---|---|---|---|---|---|---|---|
| 2 | Add1 | Yes | SUBD | M(A1) | M(A2) | 0 | 0 |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 10 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | *FU* | Mult1 | M(A2) | 0 | M(A1) | Add1 | Mult2 | 0 | | 0 |
| | | | 0 | | 0 | | | | | |

- **Timer starts down for Add1, Mult1**

# Tomasulo Example Cycle 6

אנימציה

**Issu Add, cont. Execute Sub & Mult**

*Instruction status:*

| Instruction | | j | k | *Issue* | *Exec*<br>*Comp* | *Write*<br>*Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | | |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| *Time* | *Name* | *Busy* | *Op* | *S1*<br>*Vj* | *S2*<br>*Vk* | *RS*<br>*Qj* | *RS*<br>*Qk* |
|---|---|---|---|---|---|---|---|
| 1 | Add1 | Yes | SUBD | M(A1) | M(A2) | 0 | 0 |
| | Add2 | Yes | ADDD | | M(A2) | Add1 | 0 |
| | Add3 | No | | | | | |
| 9 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **6** | *FU* | Mult1 | M(A2) | 0 | Add2 | Add1 | Mult2 | 0 | | 0 |

0

- **Issue ADDD here despite name dependency on F6?**

**No problem! Since old value of F6 as already in Vk of Mult2**

# Tomasulo Example Cycle 7

אנימציה

**Complete Sub, cont. Execute the Mult**

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| 0 | Add1 | Yes | SUBD | M(A1) | M(A2) | 0 | 0 |
| | Add2 | Yes | ADDD | | M(A2) | Add1 | 0 |
| | Add3 | No | | | | | |
| 8 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 7 | *FU* | Mult1 | M(A2) | 0 | Add2 | Add1 | Mult2 | 0 | | 0 |

0

- **Add1 (SUBD) completing; what is waiting for it?**

# Tomasulo Example Cycle 8

## WB the Sub, cont. the Mult, start the Add

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| 2 | Add2 | Yes | ADDD | (M-M) | M(A2) | 0 | 0 |
| | Add3 | No | | | | | |
| 7 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **8** | *FU* | Mult1 | M(A2) | 0 | Add2 | (M-M) | Mult2 | 0 | | 0 |
| | | | 0 | | | 0 | | | | |

# Tomasulo Example Cycle 9

## cont. the Mult & the Add

*Instruction status:*

| | | | | | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction | | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| 1 | Add2 | Yes | ADDD | (M-M) | M(A2) | 0 | 0 |
| | Add3 | No | | | | | |
| 6 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 9 | *FU* | Mult1 | M(A2) | 0 | Add2 | (M-M) | Mult2 | 0 | | 0 |
| | | | ↖ 0 | | | ↖ 0 | | | | |

# Tomasulo Example Cycle 10

אנימציה

**cont. the Mult, complete the Add**

*Instruction status:*

| | | | | | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction | | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| 0 | Add2 | Yes | ADDD | (M-M) | M(A2) | 0 | 0 |
| | Add3 | No | | | | | |
| 5 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | *FU* | Mult1 | M(A2) | 0 | Add2 | (M-M) | Mult2 | 0 | | 0 |

0 0

- **Add2 (ADDD) completing; what is waiting for it?**

# Tomasulo Example Cycle 11

אנימציה

cont. the Mult,  WB the Add

*Instruction status:*

| | | | | | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction | | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 4 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | *FU* | Mult1 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |
| | | | 0 | | 0 | 0 | | | | |

- **Write result of ADDD here?**
- **All quick instructions complete in this cycle!** (=OOO completion)

# Tomasulo Example Cycle 12

## cont. the Mult (Div still waiting)

*Instruction status:*

| | | | | | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction | | *j* | *k* | *Issue* | *Comp* | *Result* |
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 3 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **12** | *FU* | Mult1 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |
| | | | ↖ 0 | | ↖ 0 | ↖ 0 | | | | |

# Tomasulo Example Cycle 13

## cont. the Mult

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec*<br>*Comp* | *Write*<br>*Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 2 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **13** | *FU* | Mult1 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |
| | | | ↖<br>0 | | ↖<br>0 | ↖<br>0 | | | | |

# Tomasulo Example Cycle 14

## cont. the Mult

*Instruction status:*

| Instruction | | *j* | *k* | *Issue* | *Exec Comp* | *Write Result* |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| *Time* | *Name* | *Busy* | *Op* | *S1 Vj* | *S2 Vk* | *RS Qj* | *RS Qk* |
|---|---|---|---|---|---|---|---|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 1 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **14** | *FU* | Mult1 | M(A2) | 0 | (M-M+N | (M-M) | Mult2 | 0 | | 0 |
| | | | 0 | | 0 | 0 | | | | |

# Tomasulo Example Cycle 15

אנימציה

complete the Mult

*Instruction status:*

| Instruction | | j | k | Issue | Exec Comp | Write Result |
|---|---|---|---|---|---|---|
| LD | F6 | 34+ | R2 | 1 | 3 | 4 |
| LD | F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 | F4 | 3 | 15 | |
| SUBD | F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 | F6 | 5 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| Time | Name | Busy | Op | S1 Vj | S2 Vk | RS Qj | RS Qk |
|---|---|---|---|---|---|---|---|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 0 | Mult1 | Yes | MULTD | M(A2) | R(F4) | 0 | 0 |
| | Mult2 | Yes | DIVD | | M(A1) | Mult1 | 0 |

*Register result status:*

| Clock | | F0 | F2 | F4 | F6 | F8 | F10 | F12 | ... | F30 |
|---|---|---|---|---|---|---|---|---|---|---|
| 15 | FU | Mult1 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |

0

0

0

- **Mult1 (MULTD) completing; what is waiting for it?**

F0 + Mult1

# Tomasulo Example Cycle 16

אנימציה

## WB the Mult & start the Div

*Instruction status:*

| | | | | | *Exec* | *Write* |
|---|---|---|---|---|---|---|
| Instruction | | *j* | *k* | *Issue* | *Comp* | *Result* |
| 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 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| 40 | Mult2 | Yes | DIVD | M*F4 | M(A1) | 0 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| 16 | *FU* | M*F4 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |
| | | 0 | 0 | | 0 | 0 | | | | |

- **Just waiting for Mult2 (DIVD) to complete**

# Tomasulo Example Cycle 55

## cont. the Div

*Instruction status:*

| Instruction | | j | k | Issue | Exec<br>Comp | Write<br>Result |
|---|---|---|---|---|---|---|
| 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 | | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | *S1* | *S2* | *RS* | *RS* |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| 1 | Mult2 | Yes | DIVD | M*F4 | M(A1) | 0 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **55** | *FU* | M*F4 | M(A2) | 0 | (M-M+M | (M-M) | Mult2 | 0 | | 0 |
| | | ↖ 0 | ↖ 0 | | ↖ 0 | ↖ 0 | | | | |

# Tomasulo Example Cycle 56

אנימציה

**complete the Div**

*Instruction status:*

| Instruction | | j | k | Issue | Exec<br>Comp | Write<br>Result |
|---|---|---|---|---|---|---|
| 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 | 56 | |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | S1 | S2 | RS | RS |
|---|---|---|---|---|---|---|---|
| Time | Name | Busy | Op | Vj | Vk | Qj | Qk |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| 0 | Mult2 | Yes | DIVD | M*F4 | M(A1) | 0 | 0 |

*Register result status:*

| Clock | | F0 | F2 | F4 | F6 | F8 | F10 | F12 | ... | F30 |
|---|---|---|---|---|---|---|---|---|---|---|
| 56 | FU | M*F4 | M(A2) | 0 | (M-M+N | (M-M) | Mult2 | 0 | | 0 |
| | | 0 | 0 | | 0 | 0 | | | | |

- **Mult2 (DIVD) is completing; what is waiting for it?**

# Tomasulo Example Cycle 57

אנימציה

## WB the Div

*Instruction status:*

| Instruction | | j | k | Issue | Exec Comp | Write Result |
|---|---|---|---|---|---|---|
| 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 | 56 | 57 |
| ADDD | F6 | F8 | F2 | 6 | 10 | 11 |

| | Busy | Address |
|---|---|---|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

*Reservation Stations:*

| | | | | S1 | S2 | RS | RS |
|---|---|---|---|---|---|---|---|
| *Time* | *Name* | *Busy* | *Op* | *Vj* | *Vk* | *Qj* | *Qk* |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | Yes | DIVD | M*F4 | M(A1) | 0 | 0 |

*Register result status:*

| Clock | | *F0* | *F2* | *F4* | *F6* | *F8* | *F10* | *F12* | ... | *F30* |
|---|---|---|---|---|---|---|---|---|---|---|
| **56** | *FU* | M*F4 | M(A2) | 0 | (M-M+N | (M-M) | Result | 0 | | 0 |
| | | ↖ 0 | ↖ 0 | | ↖ 0 | ↖ 0 | | | | |

- **Once again: In-order issue, out-of-order execution and out-of-order completion.**

# RAW hazard prevention

- RAW is prevented by setting the Qj & Qk of each Reservation Station (FU) to the value of the unit supposed to write into these registers when issuing instruction Ij to Reservation Station r.
- That way, the r Reservation Station, will always wait for the results of the 2 units calculating Vj and Vk
- I.e., we always start calculation when Vj & Vk are ready => No RAW hazards

# WAR hazard prevention

- Since when a result of any unit is ready, we copy it into the appropriate Reservation Station registers, (and to the Register file) we actually "renamed" the registers. (use other registers instead of the orig ones)
- The data in these "renamed" registers stays there, even if someone writes to the Register File. => No WAR hazards
- No one writes into the Reservation Station registers Vj and Vk while it is still busy
- So, once a calculated result is ready, it is immediately read and renamed and any subsequent write to the register file, has no influence on the execution of that instruction
- The renaming during the issue stage ensures that an issued instruction is always performed OK

# WAW hazard prevention

- WAW happens only when we do not use the result of the first write. If we use it, there is a WAR
- In the Issue stage we set RegisterStat[Rd].Qi=r
- Issue is "in order". This means that the last instruction that is supposed to write to that register, Ij, determines its Qi field
- Therefore, when that instruction finishes, it will update the register. (and will set Qi to 0)
- If that Qi field had a different r value, coming from an earlier instruction, Ik, it is overwritten when Ij is issued, so even if that early instruction completes operation later than the Ij instruction, Qi is already 0, and no WAW occurs, since the Register File is not updated

# WAW hazard prevention (cont.)

- Furthermore, the Register File value is available for future (later) instructions

- WAW and WAR: Say that an instruction, Ip, that is between Ik and Ij needs the result of Ik. When that instruction is issued, the Qi field still holds the FU specified by Ik, so that instruction waits for "the result of Ik", and the later issued instruction Ij has no influence. (This is actually a WAR hazard prevention for the instruction Ip)
- As mentioned before,the renaming during the issue stage makes sure that an instruction is always performed correctly – eliminating the WAR

# Tomasulo's scheme

- Performs an instruction as early as possible using multiple FUs (having many input regs pairs)
- Prevents RAW hazards by waiting for the input registers to be ready before starting execution in a FU
- Prevents WAR hazards by copying registers (or FU's results) into the FUs input registers = Renaming
- During issue the appropriate reg in the Register file is flagged to wait for the appropriate FU result. Since Issue of instructions is in order, the last instruction determines which FU writes to the Reg File – Thus, no WAW can ever occur

האלגוריתם של טומסולו מבצע הוראות הכי מוקדם שניתן על-גבי היחידות הפונקציונליות.

- האלגוריתם מונע סכנת RAW על-ידי המתנה לקלטים עד שיהיו מוכנים ואז מתחיל בביצוע ב-FU.

- האלגורתם מונע גם תלות WAR על-ידי יצירת תוכן של הרגיסטרים כקלט ל- FU (שינוי שם).

-הערך האחרון של הרגיסטר הוא הקובע ולכן אין גם תלות מסוג WAW

# Tomasulo Drawbacks

- Complexity
  - delays of 360/91, MIPS 10000, Alpha 21264
- Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Each CDB must go to multiple functional units ⇒high capacitance, high wiring density
  - Number of functional units that can complete per cycle limited to one!
    - » Multiple CDBs ⇒ more FU logic for parallel assoc stores
- Non-precise interrupts!
  - We will address this later

# About Loads and Stores

- Say we Load from a memory location and then Store to the same memory location.The order must be kept to prevent WAR hazard
- Say we Store to a memory location and then Load from the same memory location. The order must be kept in order to prevent RAW hazard
- In a similar manner, WAW hazards should be prevented.
- In order to do this, we must compare the effective memory address (i.e., after the address calculation)
- A simple way to ensure correctness is to calculate all the effective addresses "in-order" (order of Loads before Store doesn't matter)

# End of Example #1
# on Tomasulo's algorithm

# Tomasulo’s Algorithm: Renaming

- Register rename table (register alias table)

|  | tag | value | valid? |
|---|---|---|---|
| R0 |  |  | 1 |
| R1 |  |  | 1 |
| R2 |  |  | 1 |
| R3 |  |  | 1 |
| R4 |  |  | 1 |
| R5 |  |  | 1 |
| R6 |  |  | 1 |
| R7 |  |  | 1 |
| R8 |  |  | 1 |
| R9 |  |  | 1 |

נחזור על הדברים הפעם
עם קונבנציה מעט שונה
וגם נראה דוגמה נוספת.

# Tomasulo’s Algorithm

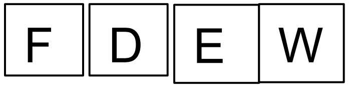
- If reservation station available before renaming אם התחנה זמינה – מלא אותה
  - Instruction + renamed operands (source value/tag) inserted into the reservation station
  - Only rename if reservation station is available

**CDB=Common Data Bus**

- Else stall אם לא, עצור
- While in reservation station, each instruction:
  - Watches common data bus (CDB) for tag of its sources
  - When tag seen, grab value for the source and keep it in the reservation station
  - When both operands available, instruction ready to be dispatched
- Dispatch instruction to the Functional Unit when instruction is ready
- After instruction finishes in the Functional Unit
  - Arbitrate (לבורר) for CDB
  - Put tagged value onto CDB (tag broadcast)
  - Register file is connected to the CDB
    - Register contains a tag indicating the latest writer to the register
    - If the tag in the register file matches the broadcast tag, write broadcast value into register (and set valid bit)
  - Reclaim rename tag
    - no valid copy of tag in system!

# An Exercise

```
MUL   R3 ← R1, R2
ADD   R5 ← R3, R4
ADD   R7 ← R2, R6
ADD   R10 ← R8, R9
MUL   R11 ← R7, R10
ADD   R5 ← R5, R11
```



F D E W

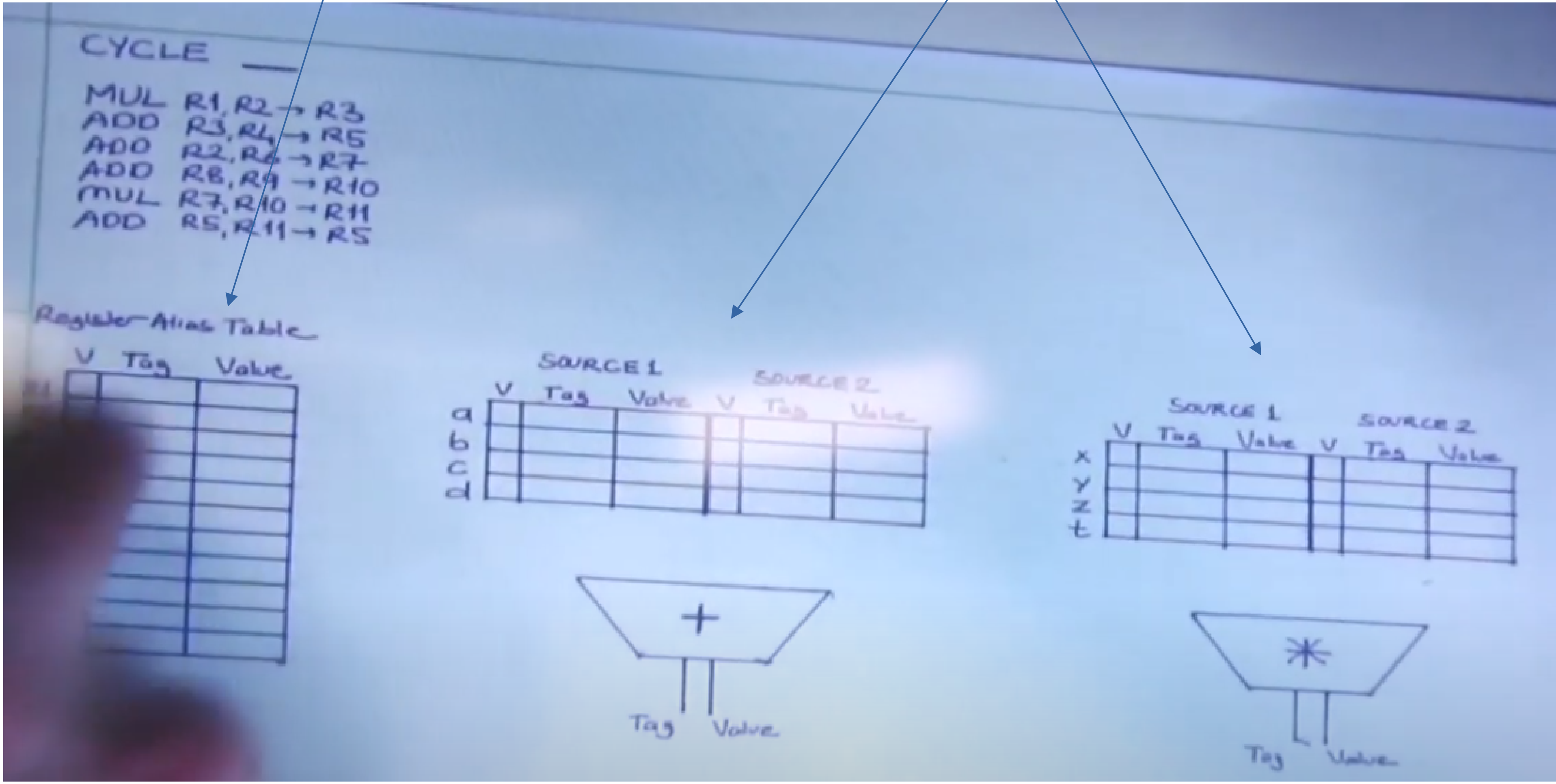
- Assume ADD (4 cycle execute), MUL (6 cycle execute)
- Assume one adder and one multiplier
- How many cycles
  - in a non-pipelined machine
  - in an in-order-dispatch pipelined machine with imprecise exceptions (and full forwarding)
  - in an out-of-order dispatch pipelined machine imprecise exceptions (full forwarding)

Register alias table (v=valid bit, tag, value)

V=0 מישהו עומד לכתוב לרגיסטר הזה, הוא בפייפליין.

אם לרגיסטר סימן V valid אז יש לו ערך אחרת יש לו tag

Reservation station

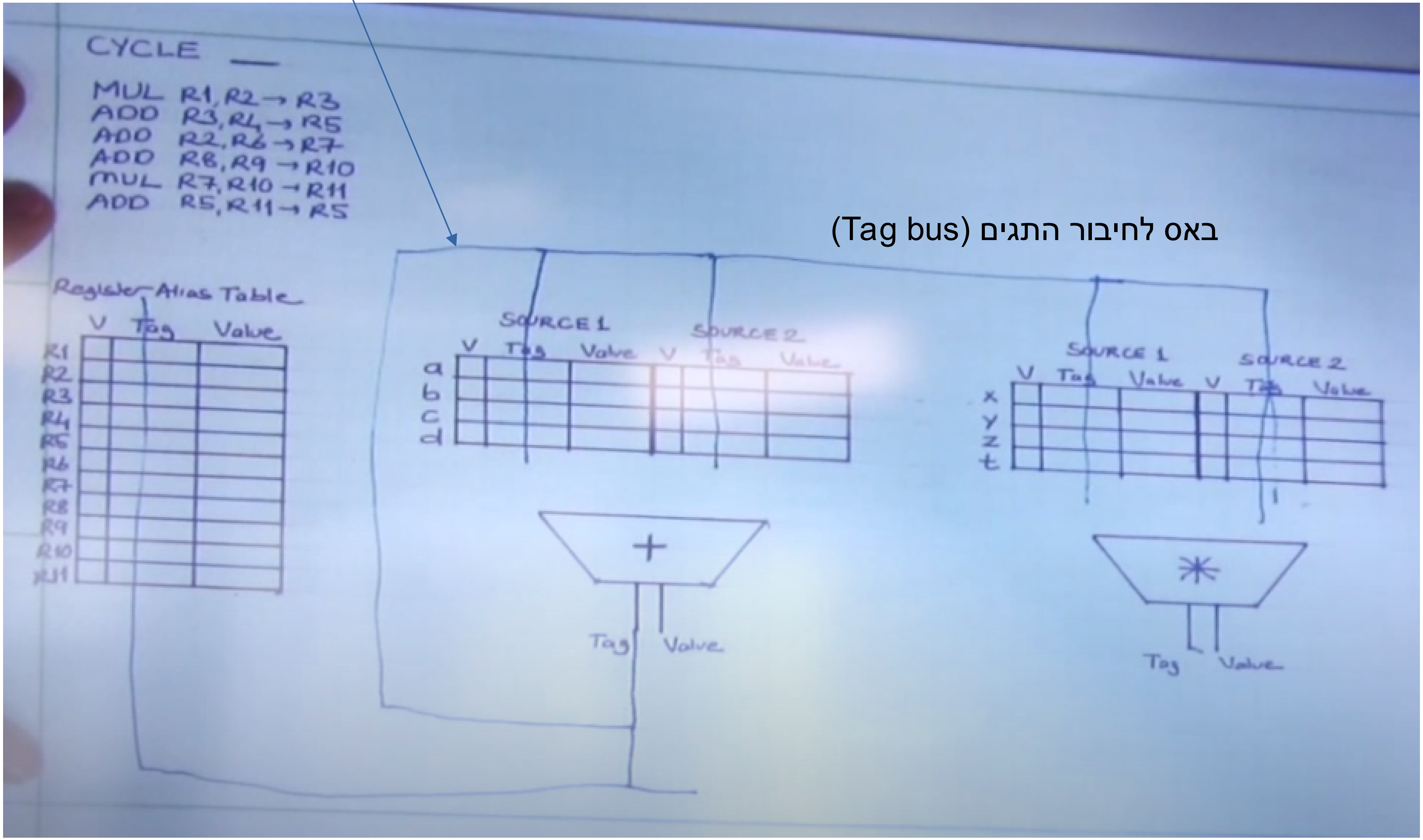


CYCLE __

MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5

אם רגיסטר תקין יש לו ערך אחרת יש לו תג

CDB=Common Data Bus



CYCLE ___

```
MUL  R1,R2 → R3
ADD  R3,R4 → R5
ADD  R2,R6 → R7
ADD  R8,R9 → R10
MUL  R7,R10 → R11
ADD  R5,R11 → R5
```

Register Alias Table

|  | V | Tag | Value |
|---|---|---|---|
| R1 |  |  |  |
| R2 |  |  |  |
| R3 |  |  |  |
| R4 |  |  |  |
| R5 |  |  |  |
| R6 |  |  |  |
| R7 |  |  |  |
| R8 |  |  |  |
| R9 |  |  |  |
| R10 |  |  |  |
| R11 |  |  |  |

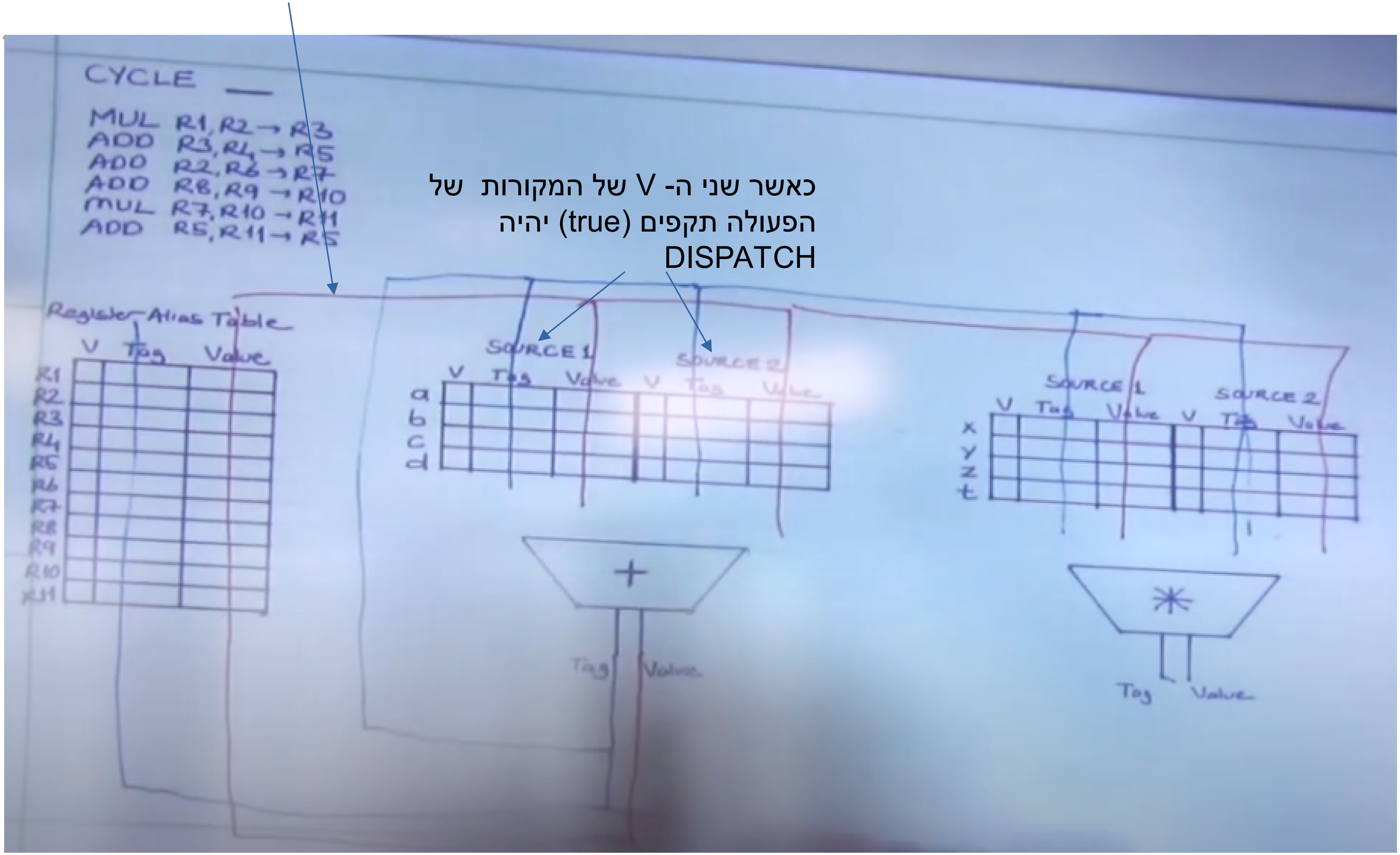
|  | SOURCE 1 |  |  | SOURCE 2 |  |  |
|---|---|---|---|---|---|---|
|  | V | Tag | Value | V | Tag | Value |
| a |  |  |  |  |  |  |
| b |  |  |  |  |  |  |
| c |  |  |  |  |  |  |
| d |  |  |  |  |  |  |

|  | SOURCE 1 |  |  | SOURCE 2 |  |  |
|---|---|---|---|---|---|---|
|  | V | Tag | Value | V | Tag | Value |
| x |  |  |  |  |  |  |
| y |  |  |  |  |  |  |
| z |  |  |  |  |  |  |
| t |  |  |  |  |  |  |

(Tag bus) באס לחיבור התגים

וישנם עוד buses למשל גם לכפול

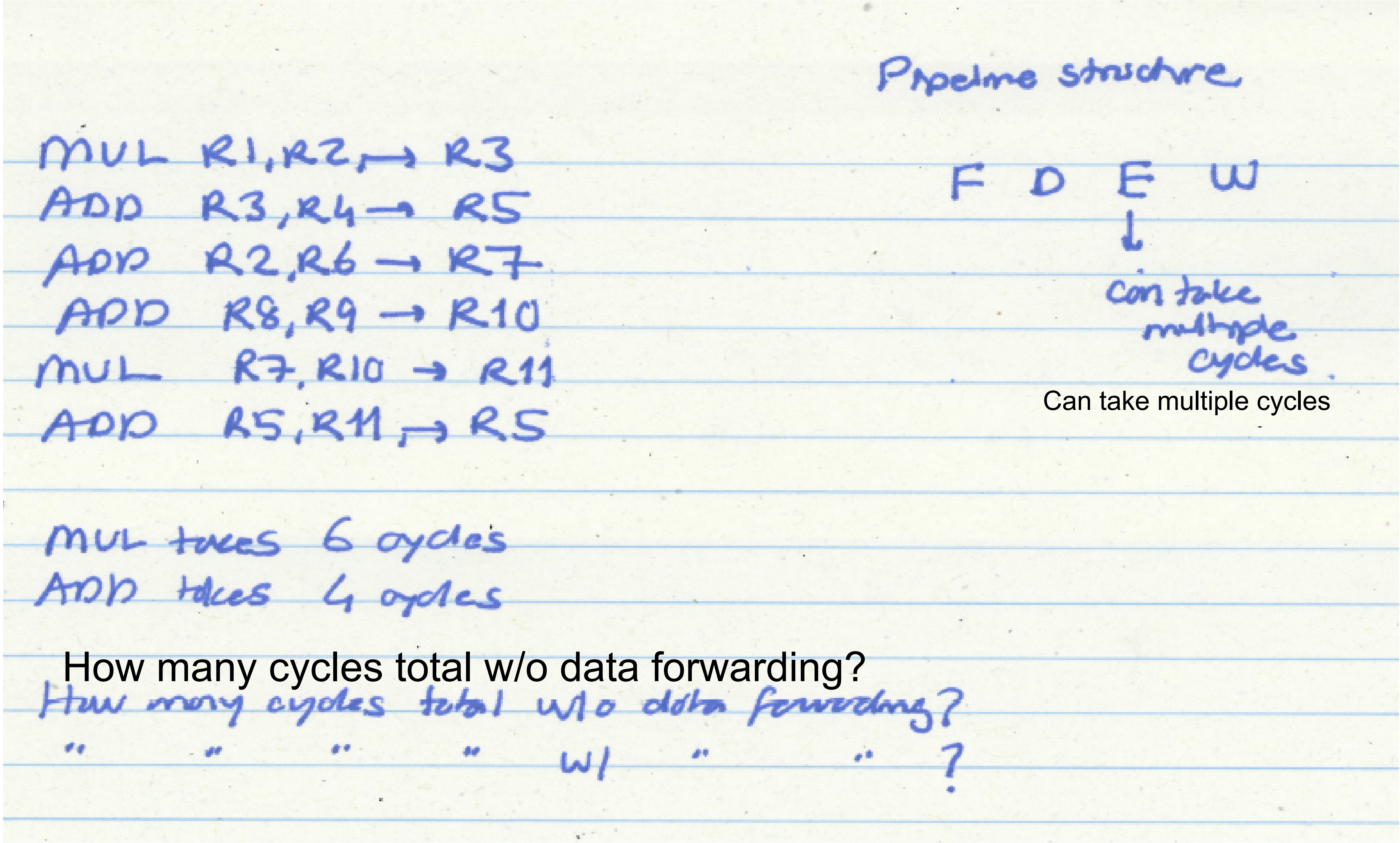
Value bus



כאשר שני ה- V של המקורות של הפעולה תקפים (true) יהיה DISPATCH

התג הוא ה- ID של ה-
reservation station

# Exercise Continued



Pipeline structure

F D E W

↓

can take multiple cycles

Can take multiple cycles

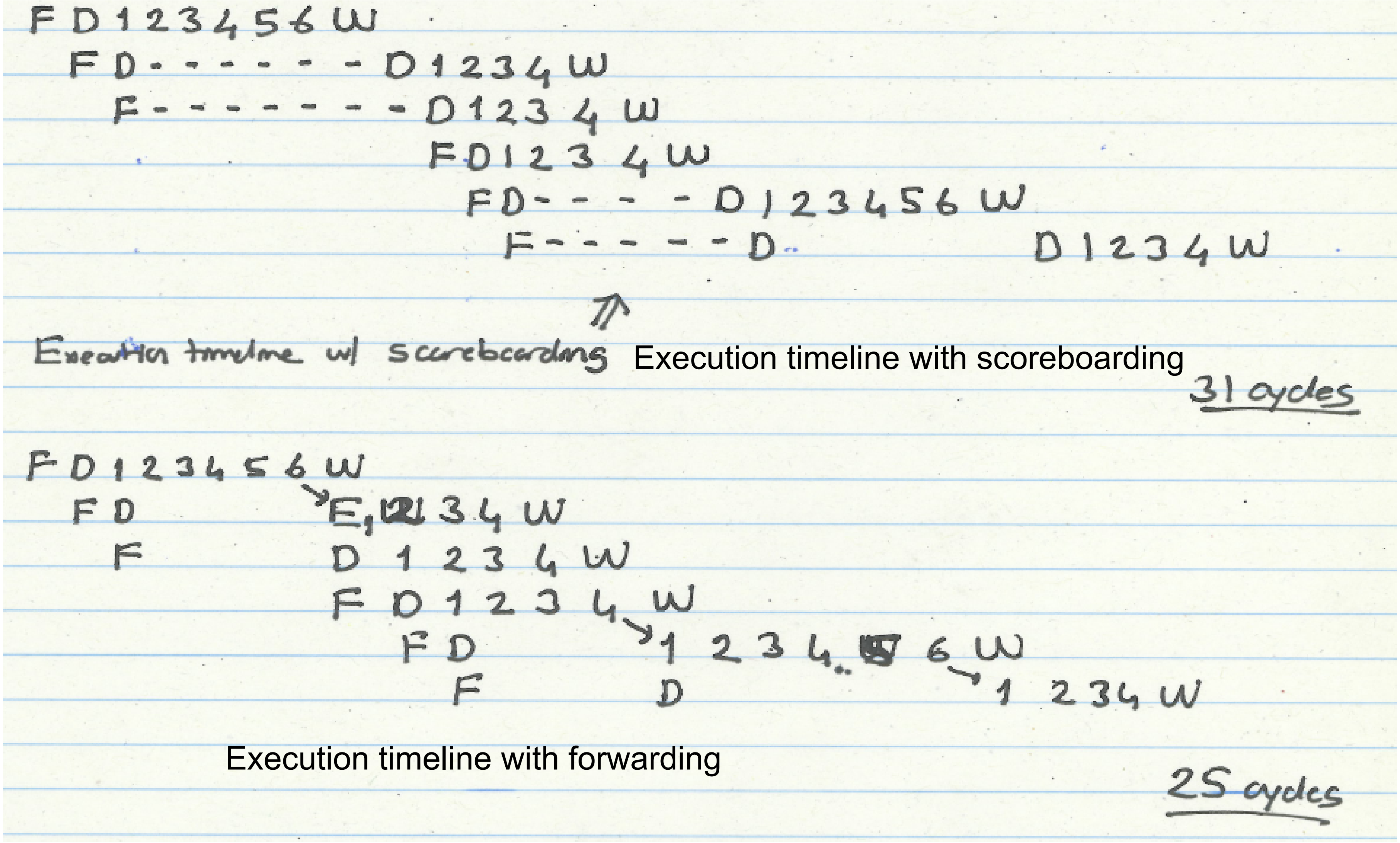
MUL R1,R2 → R3
ADD R3,R4 → R5
ADD R2,R6 → R7
ADD R8,R9 → R10
MUL R7,R10 → R11
ADD R5,R11 → R5

MUL takes 6 cycles
ADD takes 4 cycles

How many cycles total w/o data forwarding?

How many cycles total w/o data forwarding?
" " " " w/ " " ?

# Exercise Continued



```
F D 1 2 3 4 5 6 W
  F D - - - - - - D 1 2 3 4 W
    F - - - - - - - D 1 2 3 4 W
                  F D 1 2 3 4 W
                    F D - - - - D 1 2 3 4 5 6 W
                      F - - - - - D           D 1 2 3 4 W

Execution timeline w/ scoreboarding
                                                        31 cycles
```

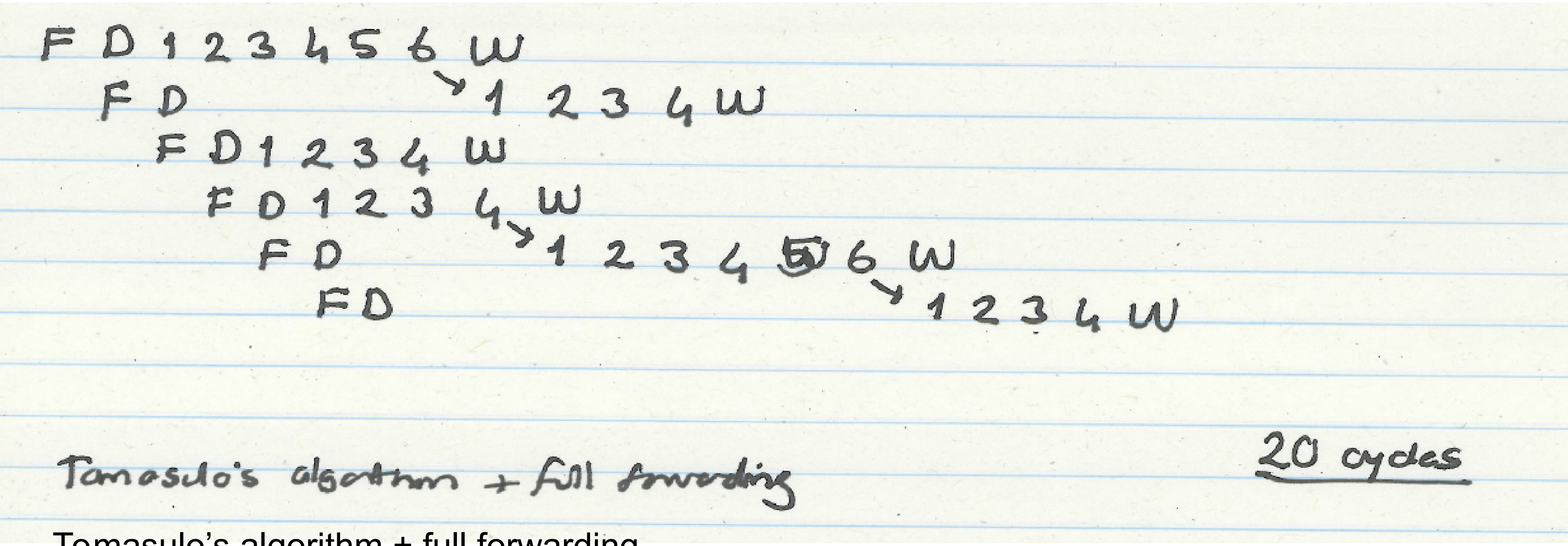
Execution timeline with scoreboarding

```
F D 1 2 3 4 5 6 W
  F D         E1 2 3 4 W
    F         D 1 2 3 4 W
              F D 1 2 3 4 W
                F D       1 2 3 4 5 6 W
                  F       D         1 2 3 4 W
                                                        25 cycles
```

Execution timeline with forwarding

# Exercise Continued

```
MUL   R3 ← R1, R2
ADD   R5 ← R3, R4
ADD   R7 ← R2, R6
ADD   R10 ← R8, R9
MUL   R11 ← R7, R10
ADD   R5 ← R5, R11
```



| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | 1 | 2 | 3 | 4 | 5 | 6 | W | | | | | | | | | | | |
| | F | D | | | | | | 1 | 2 | 3 | 4 | W | | | | | | | |
| | | F | D | 1 | 2 | 3 | 4 | W | | | | | | | | | | | |
| | | | F | D | 1 | 2 | 3 | 4 | W | | | | | | | | | | |
| | | | | F | D | | | | 1 | 2 | 3 | 4 | 5 | 6 | W | | | | |
| | | | | | F | D | | | | | | | | | 1 | 2 | 3 | 4 | W |

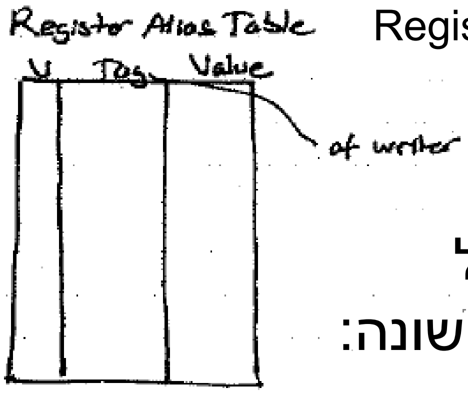
Tomasulo's algorithm + full forwarding

20 cycles

Tomasulo's algorithm + full forwarding

31 cycles → 25 cycles → 20 cycles. 11/31=0.35, 35% improvement!

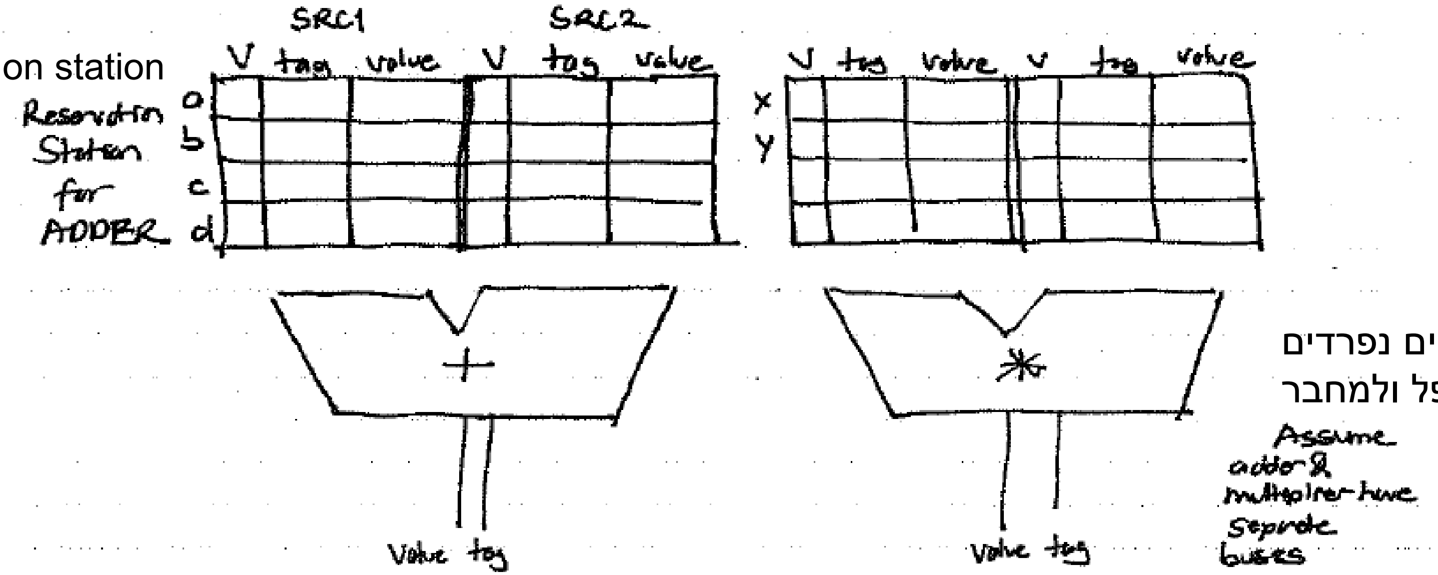
# How It Works



Register alias table

נביא עתה דוגמה שניה לביצוע האלגוריתם של
טומסולו בשני הבדלים בהשוואה לדוגמה הראשונה:
א. בדוגמה זו יש RS נפרד לכל FU.
ב. הקונבנציה של שמות העמודות ב-RS מעט שונה.

Reservation station for adder



נניח שיש בסים נפרדים
לכופל ולמחבר

אנימציה

# Our First OoO Machine Simulation

## Program We Will Simulate

```
MUL   R1, R2    →   R3
ADD   R3, R4    →   R5
ADD   R2, R6    →   R7
ADD   R8, R9    →   R10
MUL   R7, R10   →   R11
ADD   R5, R11   →   R5
```

Initially:

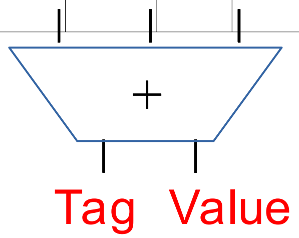
1. RS's are all Invalid (Empty)
2. All Registers are Valid

## Register Alias Table

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 3 |
| R4 | 1 | | 4 |
| R5 | 1 | | 5 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |

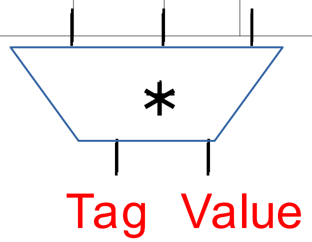
## RS for ADD Unit

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |



## RS for MUL Unit

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | | | | | | |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |



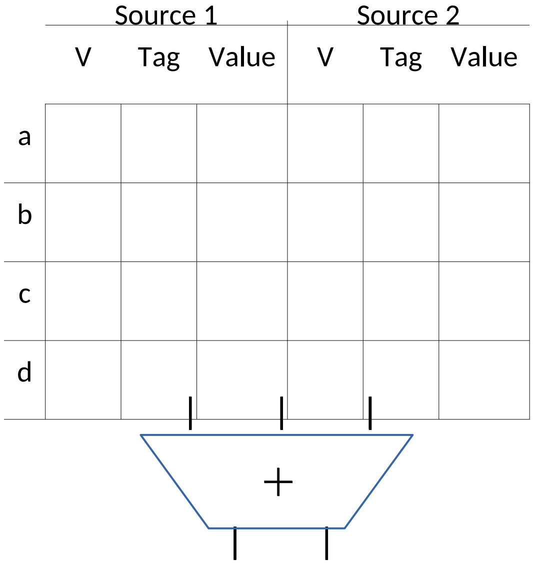
ADD and MUL Execution Units have separate buses

# Cycle 0

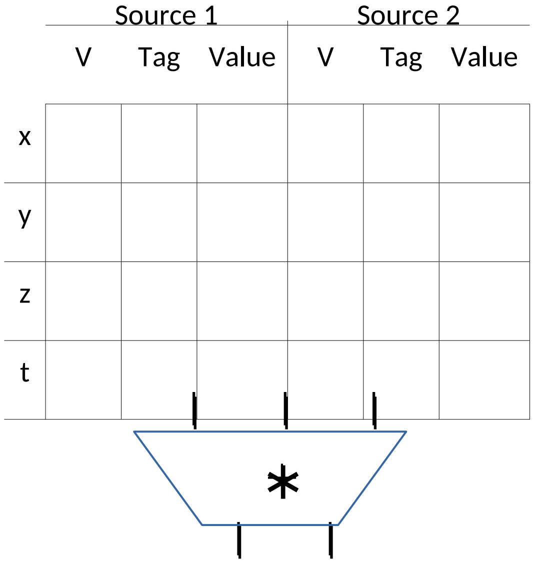
Cycle

```
MUL   R1, R2    →   R3
ADD   R3, R4    →   R5
ADD   R2, R6    →   R7
ADD   R8, R9    →   R10
MUL   R7, R10   →   R11
ADD   R5, R11   →   R5
```

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 3 |
| R4 | 1 | | 4 |
| R5 | 1 | | 5 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |

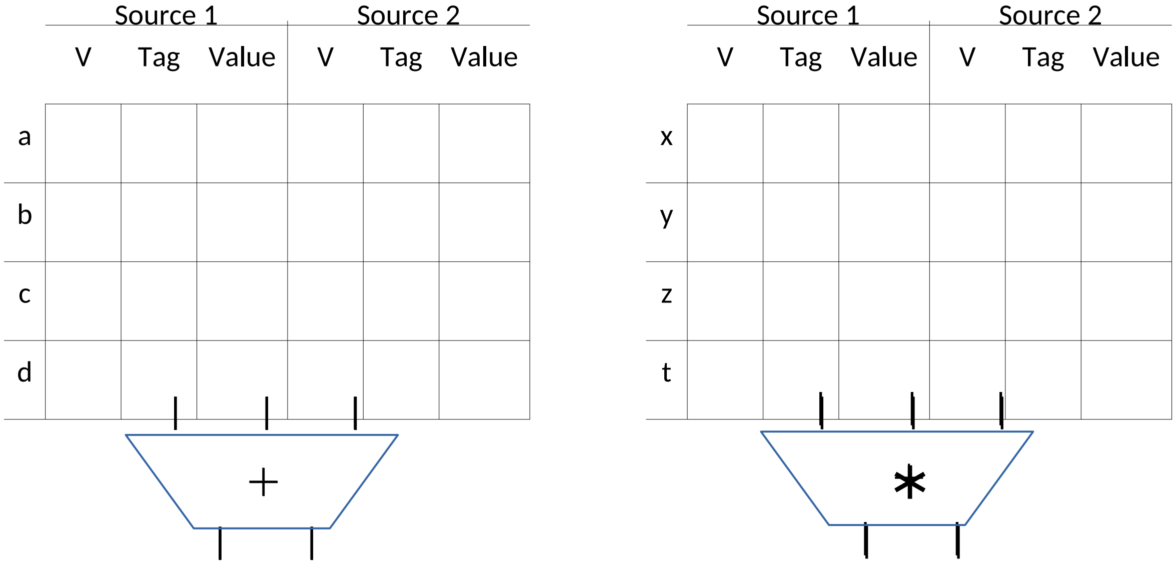


| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | | | | | | |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |

# Cycle 1

| | Cycle | 1 |
|---|---|---|
| MUL | R1, R2 → R3 | F |
| ADD | R3, R4 → R5 | |
| ADD | R2, R6 → R7 | |
| ADD | R8, R9 → R10 | |
| MUL | R7, R10 → R11 | |
| ADD | R5, R11 → R5 | |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 3 |
| R4 | 1 | | 4 |
| R5 | 1 | | 5 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | | | | | | |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |

# Cycle 2

אנימציה

MUL gets decoded and allocated into RS x

Step 1: Check if reservation station available. Yes: x

Step 2: Access *the Register Alias Table*

Step 3: Put source registers into reservation station x.

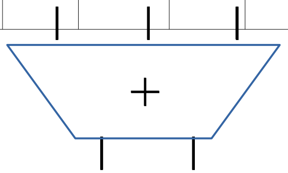
Step 4: Rename destination register R3 → x

R3 is now renamed to x.
Its new value will produced by the *reservation station* that is identified by tag x.

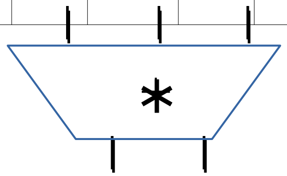
| | | | | Cycle | 1 | 2 |
|---|---|---|---|---|---|---|
| MUL | R1, | R2 | → | R3 | F | D |
| ADD | R3, | R4 | → | R5 | | F |
| ADD | R2, | R6 | → | R7 | | |
| ADD | R8, | R9 | → | R10 | | |
| MUL | R7, | R10 | → | R11 | | |
| ADD | R5, | R11 | → | R5 | | |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1<br>1 | | 1<br>1 |
| R2 | 1<br>1 | | 2<br>2 |
| R3 | 0 | x | |
| R4 | 1 | | 4 |
| R5 | 1 | | 5 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | | ~ | | | ~ | |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |



MUL in RS x is ready to execute in the next cycle!

# Cycle 3

אנימציה

1. MUL in RS x starts executing
2. ADD gets decoded and allocated into RS a

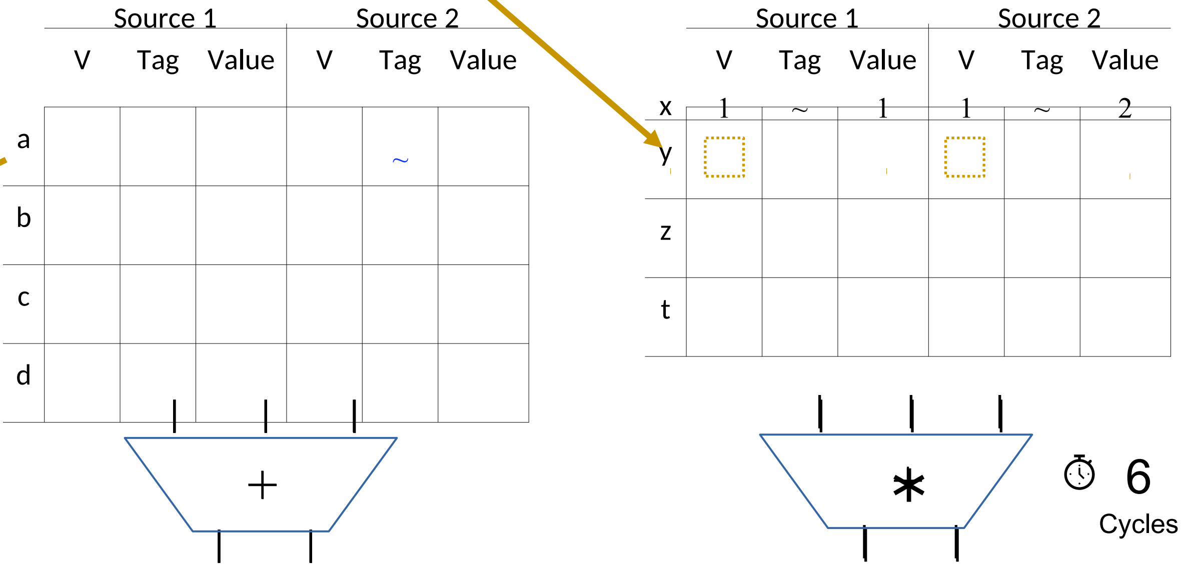
| | Cycle | 1 | 2 | 3 |
|---|---|---|---|---|
| MUL | R1, R2 → R3 | F | D | $E_1$ |
| ADD | R3, R4 → R5 | | F | D |
| ADD | R2, R6 → R7 | | | F |
| ADD | R8, R9 → R10 | | | |
| MUL | R7, R10 → R11 | | | |
| ADD | R5, R11 → R5 | | | |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 0<br>0 | x<br>x | |
| R4 | 1<br>0 | a | 4 |
| R5 | 1 | | 5 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |

Check readiness (Both sources ready?) → Wakeup

Ready → Dispatch the instruction to the MUL unit

Same Steps 1-4 for ADD… Rename R5 → a



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | ~ | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |

ADD in *RS a* cannot execute in the next cycle: one source is not valid

# Cycle 4

אנימציה

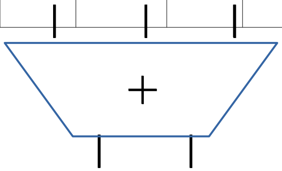
| | Cycle | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| MUL | R1, R2 → R3 | F | D | $E_1$ | $E_2$ |
| ADD | R3, R4 → R5 | | F | D | - |
| ADD | R2, R6 → R7 | | | F | D |
| ADD | R8, R9 → R10 | | | | F |
| MUL | R7, R10 → R11 | | | | |
| ADD | R5, R11 → R5 | | | | |

ADD in RS a waits because one source is not valid.

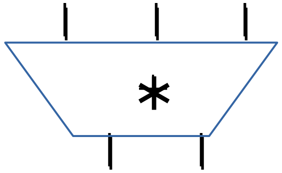
Rename R7 → b

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1<br>1 | | 2<br>2 |
| R3 | 0 | x | |
| R4 | 1 | | 4 |
| R5 | 0<br>1<br>0 | a<br>b | 6 |
| R6 | 1 | | 6 |
| R7 | 1 | | 7 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 0 | x | | 1 | ~ | 4 |
| b | | ~ | | | ~ | |
| c | | | | | | |
| d | | | | | | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |



ADD in RS b is ready to execute in the next cycle!

It will be executed out of order in the next cycle.

# Cycle 5

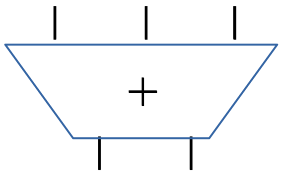
אנימציה

| Cycle | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| MUL R1, R2 → R3 | F | D | $E_1$ | $E_2$ | $E_3$ |
| ADD R3, R4 → R5 | | F | D | - | - |
| ADD R2, R6 → R7 | | | F | D | $E_1$ |
| ADD R8, R9 → R10 | | | | F | D |
| MUL R7, R10 → R11 | | | | | F |
| ADD R5, R11 → R5 | | | | | |

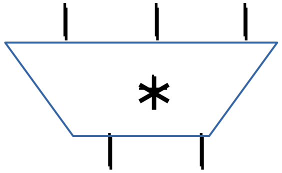
| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 0 | x | |
| R4 | 1 | | 4 |
| R5 | 0 | a | |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 0<br>1 | c | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 10 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 0 | x | | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | | | | | | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |



⏱ 4
Cycles



ADD in RS c is ready to execute in the next cycle!

# Cycle 6

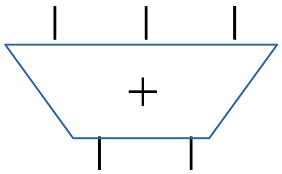
אנימציה

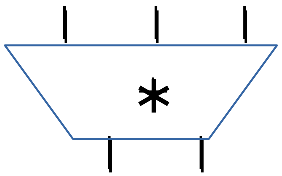
| | | | Cycle | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| MUL | R1, R2 | → | R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| ADD | R3, R4 | → | R5 | | F | D | - | - | - |
| ADD | R2, R6 | → | R7 | | | F | D | $E_1$ | $E_2$ |
| ADD | R8, R9 | → | R10 | | | | F | D | $E_1$ |
| MUL | R7, R10 | → | R11 | | | | | F | D |
| ADD | R5, R11 | → | R5 | | | | | | F |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 0 | x | |
| R4 | 1 | | 4 |
| R5 | 0 | a | |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 1<br>0 | y | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 0 | x | | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | | | | | | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 0 | b | | 0 | c | |
| z | | | | | | |
| t | | | | | | |





# Cycle 7

אנימציה

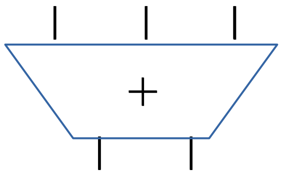
All 6 instructions are now decoded and renamed

Note what happened to R5!

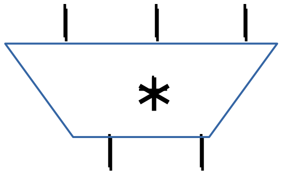
| | | | | | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL | R1, R2 | → | R3 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ |
| ADD | R3, R4 | → | R5 | | | | F | D | - | - | - | - |
| ADD | R2, R6 | → | R7 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ |
| ADD | R8, R9 | → | R10 | | | | | | F | D | $E_1$ | $E_2$ |
| MUL | R7, R10 | → | R11 | | | | | | | F | D | - |
| ADD | R5, R11 | → | R5 | | | | | | | | F | D |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 0 | x | |
| R4 | 0 | d | |
| R5 | 0 | a | |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 0 | x | | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 0 | b | | 0 | c | |
| z | | | | | | |
| t | | | | | | |



אנימציה

# Cycle 8 (First Slide)

| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | |
| MUL R7, R10 → R11 | | | | | | F | D | - | |
| ADD R5, R11 → R5 | | | | | | | F | D | |

MUL in RS x is done

Broadcast MUL's tag (x)

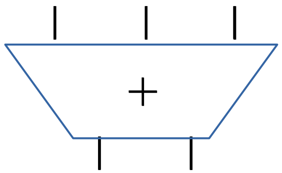
- ✓ Check tag
- ✓ Check for invalidity

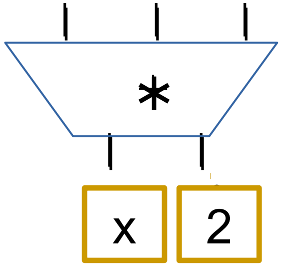
Broadcast MUL's result (2)

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 0 | b | | 0 | c | |
| z | | | | | | |
| t | | | | | | |





ADD in RS a is ready to execute in the next cycle!

# Cycle 8 (Second Slide)

אנימציה

| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | |
| MUL R7, R10 → R11 | | | | | | F | D | - | |
| ADD R5, R11 → R5 | | | | | | | F | D | |

ADD in RS b is also done

Broadcast ADD's tag (b)

✓ Check tag

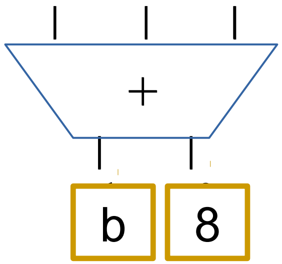
✓ Check for invalidity

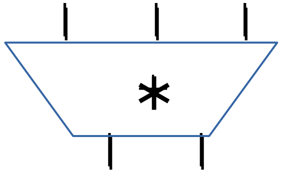
Broadcast ADD's result (8)

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0<br>1 | d | 8 |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 0 | b | | 0 | c | |
| z | 1 | | 8 | | | |
| t | | | | | | |





MUL in RS y is still NOT ready to execute in the next cycle!

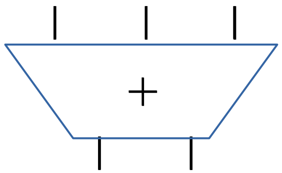
# Cycle 8 (Third Slide)

אנימציה

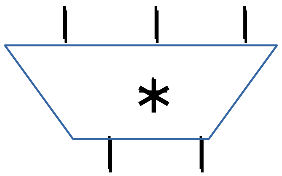
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ |
| MUL R7, R10 → R11 | | | | | | F | D | - | - |
| ADD R5, R11 → R5 | | | | | | | F | D | - |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | b | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |
| R11 | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 0 | c | |
| z | | | | | | |
| t | | | | | | |



# Cycle 9

אנימציה

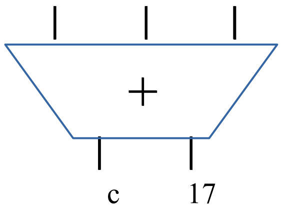
| Cycle | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 | → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W |
| ADD R3, R4 | → R5 | | F | D | - | - | - | - | - | $E_1$ |
| ADD R2, R6 | → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W |
| ADD R8, R9 | → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| MUL R7, R10 | → R11 | | | | | F | D | - | - | - |
| ADD R5, R11 | → R5 | | | | | | F | D | - | - |

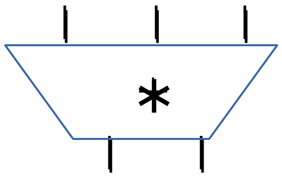
Broadcast and Update

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1<br>1 | | 17<br>8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |
| | | | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 0<br>1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





MUL in RS y is ready to execute in the next cycle!

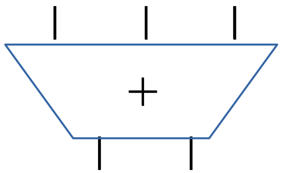
# Cycle 10

אנימציה

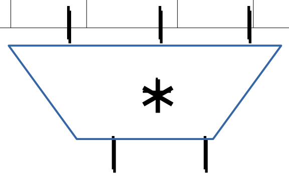
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 | → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | |
| ADD R3, R4 | → R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ |
| ADD R2, R6 | → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | |
| ADD R8, R9 | → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W |
| MUL R7, R10 | → R11 | | | | | F | D | - | - | - | $E_1$ |
| ADD R5, R11 | → R5 | | | | | | F | D | - | - | - |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |



# Cycle 11

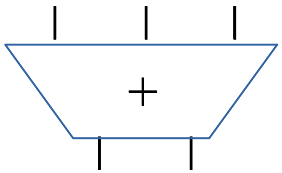
אנימציה

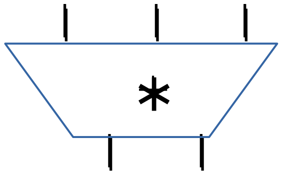
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 | → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | |
| ADD R3, R4 | → R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ |
| ADD R2, R6 | → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | |
| ADD R8, R9 | → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | |
| MUL R7, R10 | → R11 | | | | | F | D | - | - | - | $E_1$ | $E_2$ |
| ADD R5, R11 | → R5 | | | | | | F | D | - | - | - | - |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 12

אנימציה

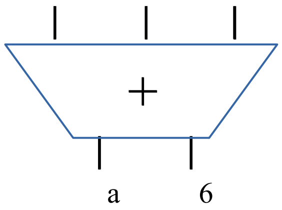
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | |
| MUL R7, R10 → R11 | | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ |
| ADD R5, R11 → R5 | | | | | | | F | D | - | - | - | - | - |

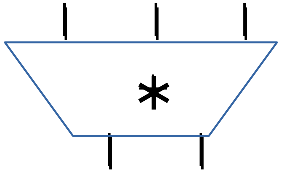
Broadcast and Update

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | ^<br>1 | ~ | 6 | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 13

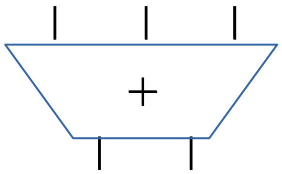
אנימציה

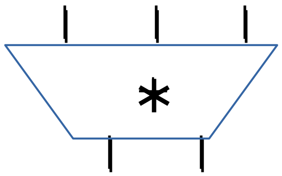
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 | → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | |
| ADD R3, R4 | → R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W |
| ADD R2, R6 | → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | |
| ADD R8, R9 | → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | |
| MUL R7, R10 | → R11 | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| ADD R5, R11 | → R5 | | | | | | F | D | - | - | - | - | - | - |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 14

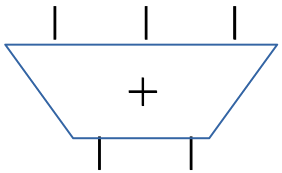
אנימציה

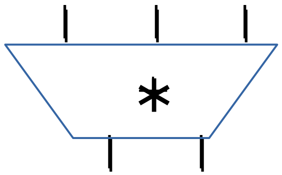
| Instruction | | | Cycle 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 | → | R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | |
| ADD R3, R4 | → | R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | |
| ADD R2, R6 | → | R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | |
| ADD R8, R9 | → | R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | |
| MUL R7, R10 | → | R11 | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ |
| ADD R5, R11 | → | R5 | | | | | | F | D | - | - | - | - | - | - | - |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 15

אנימציה

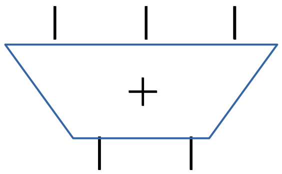
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | |
| MUL R7, R10 → R11 | | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ |
| ADD R5, R11 → R5 | | | | | | | F | D | - | - | - | - | - | - | - | - |

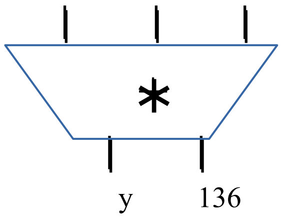
Broadcast and Update

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 136 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





ADD in RS d is ready to execute in the next cycle!

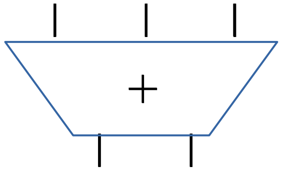
# Cycle 16

אנימציה

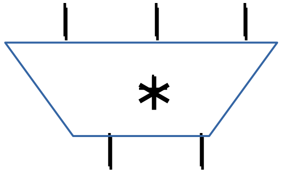
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | | |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | |
| MUL R7, R10 → R11 | | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W |
| ADD R5, R11 → R5 | | | | | | | F | D | - | - | - | - | - | - | - | - | $E_1$ |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |



# Cycle 17

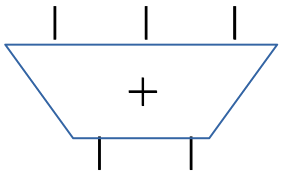
אנימציה

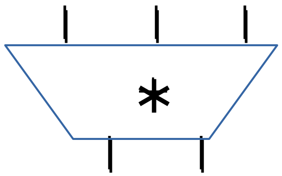
| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | | | |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | |
| MUL R7, R10 → R11 | | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | |
| ADD R5, R11 → R5 | | | | | | | F | D | - | - | - | - | - | - | - | - | $E_1$ | $E_2$ |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 18

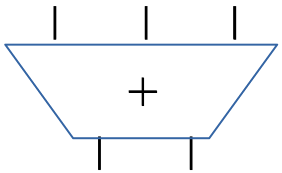
אנימציה

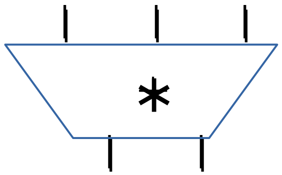
| Cycle | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL | R1, R2 | → | R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | | | | |
| ADD | R3, R4 | → | R5 | | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | |
| ADD | R2, R6 | → | R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | | |
| ADD | R8, R9 | → | R10 | | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | |
| MUL | R7, R10 | → | R11 | | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | |
| ADD | R5, R11 | → | R5 | | | | | | | F | D | - | - | - | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |





# Cycle 19

אנימציה

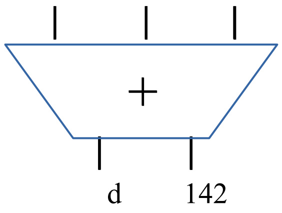
| Instruction | Cycle 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | | | | | |
| ADD R3, R4 → R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | |
| ADD R2, R6 → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | | | |
| ADD R8, R9 → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | | |
| MUL R7, R10 → R11 | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | |
| ADD R5, R11 → R5 | | | | | | F | D | - | - | - | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ |

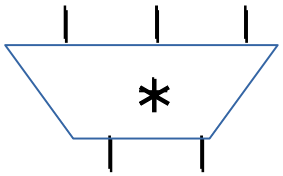
Broadcast and Update

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 142 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |



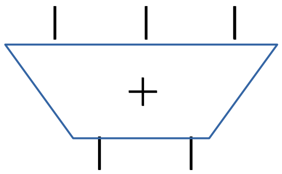


# Cycle 20

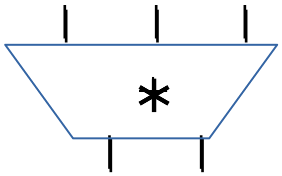
| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | | | | | | | | |
| ADD R3, R4 → R5 | | F | D | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | |
| ADD R2, R6 → R7 | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | | | | |
| ADD R8, R9 → R10 | | | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | W | | | | | | | | | | |
| MUL R7, R10 → R11 | | | | | F | D | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | W | | | | W |
| ADD R5, R11 → R5 | | | | | | F | D | - | - | - | - | - | - | - | - | $E_1$ | $E_2$ | $E_3$ | $E_4$ | |

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 1 | | 2 |
| R4 | 1 | | 4 |
| R5 | 1 | | 142 |
| R6 | 1 | | 6 |
| R7 | 1 | | 8 |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 1 | | 17 |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 1 | ~ | 2 | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 1 | ~ | 6 | 1 | ~ | 136 |

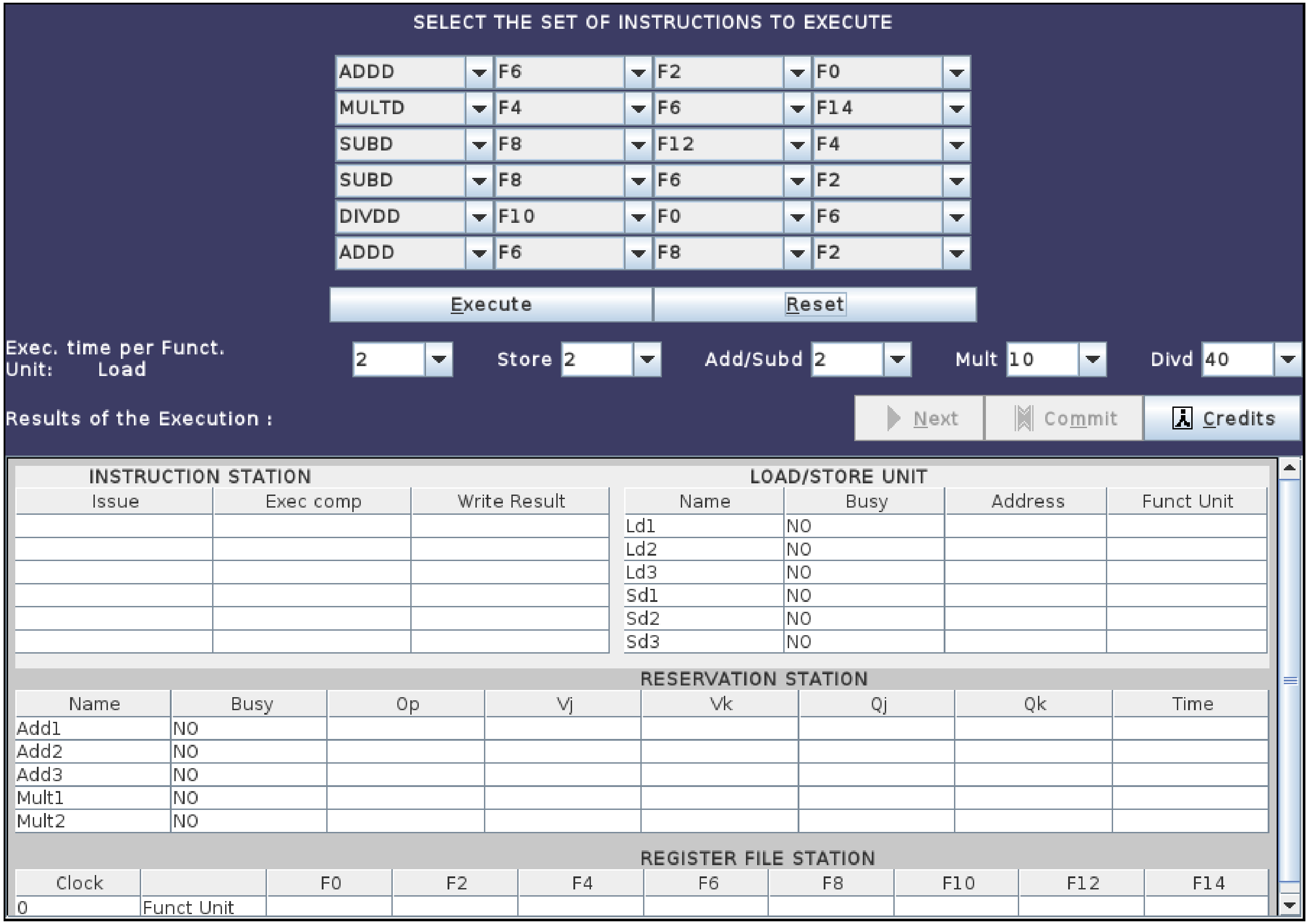


| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 1 | ~ | 8 | 1 | ~ | 17 |
| z | | | | | | |
| t | | | | | | |



# דוגמא נוספת:

**Dynamic Scheduling Using Tomasulo's Algorithm**



SELECT THE SET OF INSTRUCTIONS TO EXECUTE

| | | | |
|---|---|---|---|
| ADDD | F6 | F2 | F0 |
| MULTD | F4 | F6 | F14 |
| SUBD | F8 | F12 | F4 |
| SUBD | F8 | F6 | F2 |
| DIVDD | F10 | F0 | F6 |
| ADDD | F6 | F8 | F2 |

Exec. time per Funct. Unit: Load 2 Store 2 Add/Subd 2 Mult 10 Divd 40

Results of the Execution :

INSTRUCTION STATION

| Issue | Exec comp | Write Result |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

LOAD/STORE UNIT

| Name | Busy | Address | Funct Unit |
|---|---|---|---|
| Ld1 | NO | | |
| Ld2 | NO | | |
| Ld3 | NO | | |
| Sd1 | NO | | |
| Sd2 | NO | | |
| Sd3 | NO | | |

RESERVATION STATION

| Name | Busy | Op | Vj | Vk | Qj | Qk | Time |
|---|---|---|---|---|---|---|---|
| Add1 | NO | | | | | | |
| Add2 | NO | | | | | | |
| Add3 | NO | | | | | | |
| Mult1 | NO | | | | | | |
| Mult2 | NO | | | | | | |

REGISTER FILE STATION

| Clock | | F0 | F2 | F4 | F6 | F8 | F10 | F12 | F14 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Funct Unit | | | | | | | | |

https://www.ecs.umass.edu/ece/koren/architecture/Tomasulo/AppletTomasulo.html

# Some Questions

- What is needed in hardware to perform tag broadcast and value capture?

  → make a value valid

  → wake up an instruction

- Does the tag have to be the ID of the Reservation Station Entry?

- What can potentially become the critical path?
  - Tag broadcast → value capture → instruction wake up

- How can you reduce the potential critical paths?

# Dataflow Graph for Our Example

```
MUL   R3 ← R1, R2
ADD   R5 ← R3, R4
ADD   R7 ← R2, R6
ADD   R10 ← R8, R9
MUL   R11 ← R7, R10
ADD   R5 ← R5, R11
```

נחזור וניזכר במודל ה-Dataflow ונקשור אותו עם האלגוריתם של טומסולו.
נדגים זאת באמצעות התבוננות מחודשת על מחזור מסוים (למשל מחזור שעון מס' 7) – השקף הבא

# Cycle 7

אנימציה

All 6 instructions are now decoded and renamed

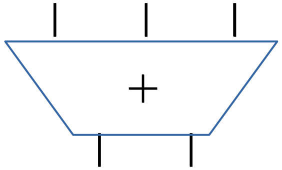
Note what happened to R5!

| | Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| MUL R1, R2 → R3 | | F | D | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ |
| ADD R3, R4 → R5 | | | F | D | - | - | - | - |
| ADD R2, R6 → R7 | | | | F | D | $E_1$ | $E_2$ | $E_3$ |
| ADD R8, R9 → R10 | | | | | F | D | $E_1$ | $E_2$ |
| MUL R7, R10 → R11 | | | | | | F | D | - |
| ADD R5, R11 → R5 | | | | | | | F | D |

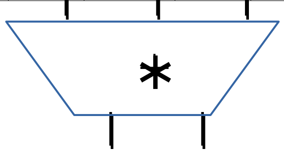
אנו רואים כי יש בשלבים שונים של
ביצוע 3 הוראות בהמתנה בתחנת
החיבור ו-2 הוראות בהמתנה בתחנת
הכפל

| Register | Valid | Tag | Value |
|---|---|---|---|
| R1 | 1 | | 1 |
| R2 | 1 | | 2 |
| R3 | 0 | x | |
| R4 | 1 | | 4 |
| R5 | 0 | d | |
| R6 | 1 | | 6 |
| R7 | 0 | b | |
| R8 | 1 | | 8 |
| R9 | 1 | | 9 |
| R10 | 0 | c | |
| R11 | 0 | y | |

| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | 0 | x | | 1 | ~ | 4 |
| b | 1 | ~ | 2 | 1 | ~ | 6 |
| c | 1 | ~ | 8 | 1 | ~ | 9 |
| d | 0 | a | | 0 | y | |



| | Source 1 | | | Source 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | 1 | ~ | 1 | 1 | ~ | 2 |
| y | 0 | b | | 0 | c | |
| z | | | | | | |
| t | | | | | | |



R5 was a and in cycle 7 it became d

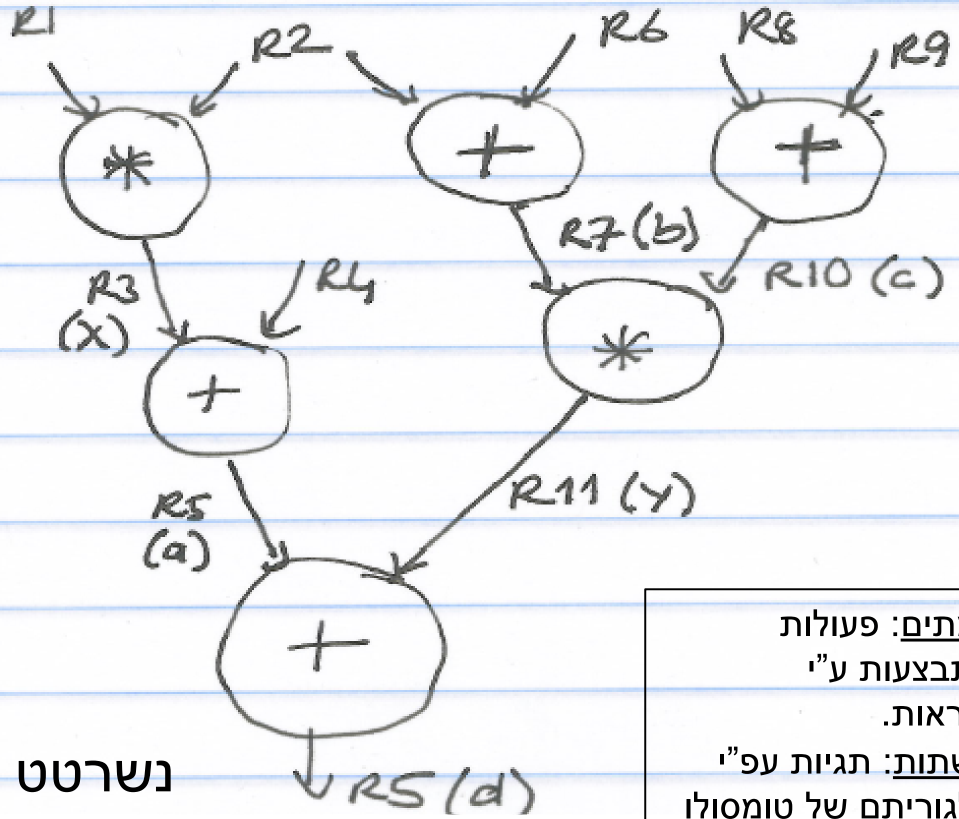
# Corresponding Dataflow Graph (Reverse Engineered)

```
MUL  R1,R2→R3 (x)
ADD  R3,R4→R5 (a)
ADD  R2,R6→R7 (b)
ADD  R8,R9→R10 (c)
MUL  R7,R10→R11 (y)
ADD  R5,R11→R5 (d)
```

Dataflow graph

Nodes: operations performed by the instruction

Arcs: tags in Tomasulo's algorithms



<u>הצמתים</u>: פעולות המתבצעות ע"י ההוראות.

<u>הקשתות</u>: תגיות עפ"י האלגוריתם של טומסולו

נשרטט את התרשים מהסוף להתחלה ולכן נתחיל מהתלות של R5 בהוראה האחרונה ונעלה במעלה הקוד

# Some More Questions (Design Choices)

- When is a reservation station entry deallocated?
- Should the reservation stations be dedicated to each functional unit or global across functional units?
  - Centralized vs. Distributed: What are the tradeoffs?
- Should reservation stations and ROB store data values or should there be a centralized physical register file where all data values are stored?
  - What are the tradeoffs?
- Timing: Exactly when does an instruction broadcast its tag?
- Many other design choices for OoO engines

# Tomasulo’s algorithm with reorder buffer

(11 slides )

# What about Precise Interrupts?

- Tomasulo had:

  In-order issue, out-of-order execution, and out-of-order completion

- Need to "fix" the out-of-order completion aspect so that we can find precise breakpoint in instruction stream.

  (We prefer to address this later – in speculative H/W)

- A similar problem in branch. Instruction following the branch may update the GPR before branch is resolved. This was solved by *not* issuing an instruction until all branch instructions preceding it are resolved!

# H/W speculation:

H/w speculation means:

We assume that we predict the branch correctly and therefore continue execution.

If it turns out that our *guess* was wrong, we should be able to reverse the calculation.

To do that, we won't update the GPR or the Memory until all previous branches are resolved

אנו מניחים חיזוי נכון של ההסתעפות. אם מתברר שהניחוש היה שגוי עלינו לבטל
את החישוב. כדי לעשות זאת לא נעדכן את ה- GPR או את הזיכרון עד אשר
ההסתעפויות שקדמו להוראה נפתרו.

# Speculative Tomasulo Algorithm

In the original scheme we issued instructions although earlier instructions were stalled. That was the main idea.The instructions were issued in-order, executed out-of-order and completed (& wrote to the GPR) also in out-of-order fashion

To prevent control hazards (due to branch instructions), we stalled the issue process when we had an unresolved branch prior to the instruction trying to be issued

This means that we did not issue instructions from different Basic Blocks and had stalls in loops

יש ניגוד עניינים בין חישוב OOO, כפי שהכרנו עד-כה, לבין חיזוי הסתעפות.. לא ניתן
לקיים את שניהם ביחד...

In the speculative scheme we avoid this by avoiding out of order completion. We want in-order completion!

# Speculative Tomasulo Algorithm

- So, in the speculative scheme we in-order issue, out-of-order execution and in-order completion (writing to the GPR or Memory)
- To do that, we need to have another buffer, replacing the GPR for keeping the calculated results (that are already calculated but still waiting to their turn to be written back to the GPR). We also need a mechanism that keeps track of the order of completion (Here completion is called Commit )
- A simple solution is a cyclic buffer (FIFO like) to which we write down the instruction when it is issued. That buffer, called the Reorder Buffer (ROB) has therefore, the correct order of the instructions. We use this buffer also for storing the result that is supposed to be written into the GPR when the instruction commits
- The head of the ROB has the instruction that commits (completes). If this is a regular instruction, we update the GPR or the Memory with the result kept in the ROB, and delete the instruction from the ROB. The next instruction is now committing. If this a correctly predicted branch, we just delete it. If this is a mispredicted branch, we clear the ROB and start executing the correct instructions

אנימציה

# Four Steps of Speculative Tomasulo Algorithm

1. **Issue**—get instruction from FP Op Queue

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

2. **Execution**—operate on operands (EX)

   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")

3. **Write result**—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. **Commit**—update register with reorder result

   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

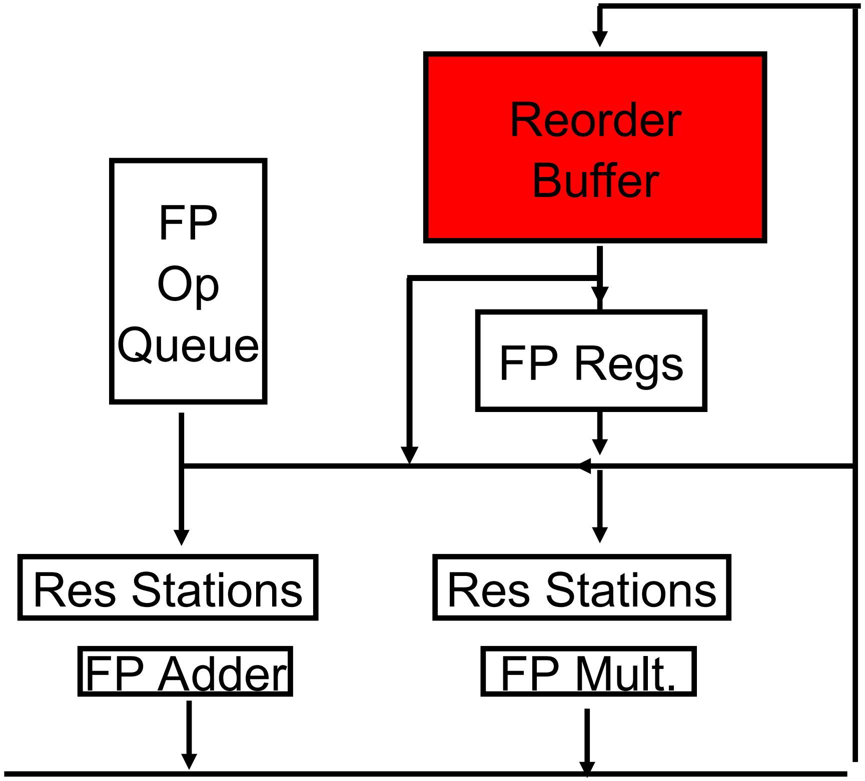
# Differences between the Speculative Tomasulo Algorithm & the previous one

1. The main difference: Results are not written to the Register File, but to the Reorder Buffer (ROB).
2. In the original algorithm, the FUs results are written back (via the CDB) to all Reservation Station waiting for that data and to the *appropriate register in the Register File* – In the speculative algorithm, the FUs results are written back (also via the CDB) to all Reservation Station waiting for that data and to the *appropriate entry in the ROB*. So the Register File is updated only during the Commit phase
3. In the speculative scheme, the ROB data is written to the appropriate register in the GPR only when the instruction commits
4. Instead of marking Qj & Qk with the FU calculating them, they are marked with the ROB entry number of the instruction that produces them. So the FU (Reservation Station) needs to have a tag with this number and transmit the tag on the CDB when result is ready (and the CDB available)

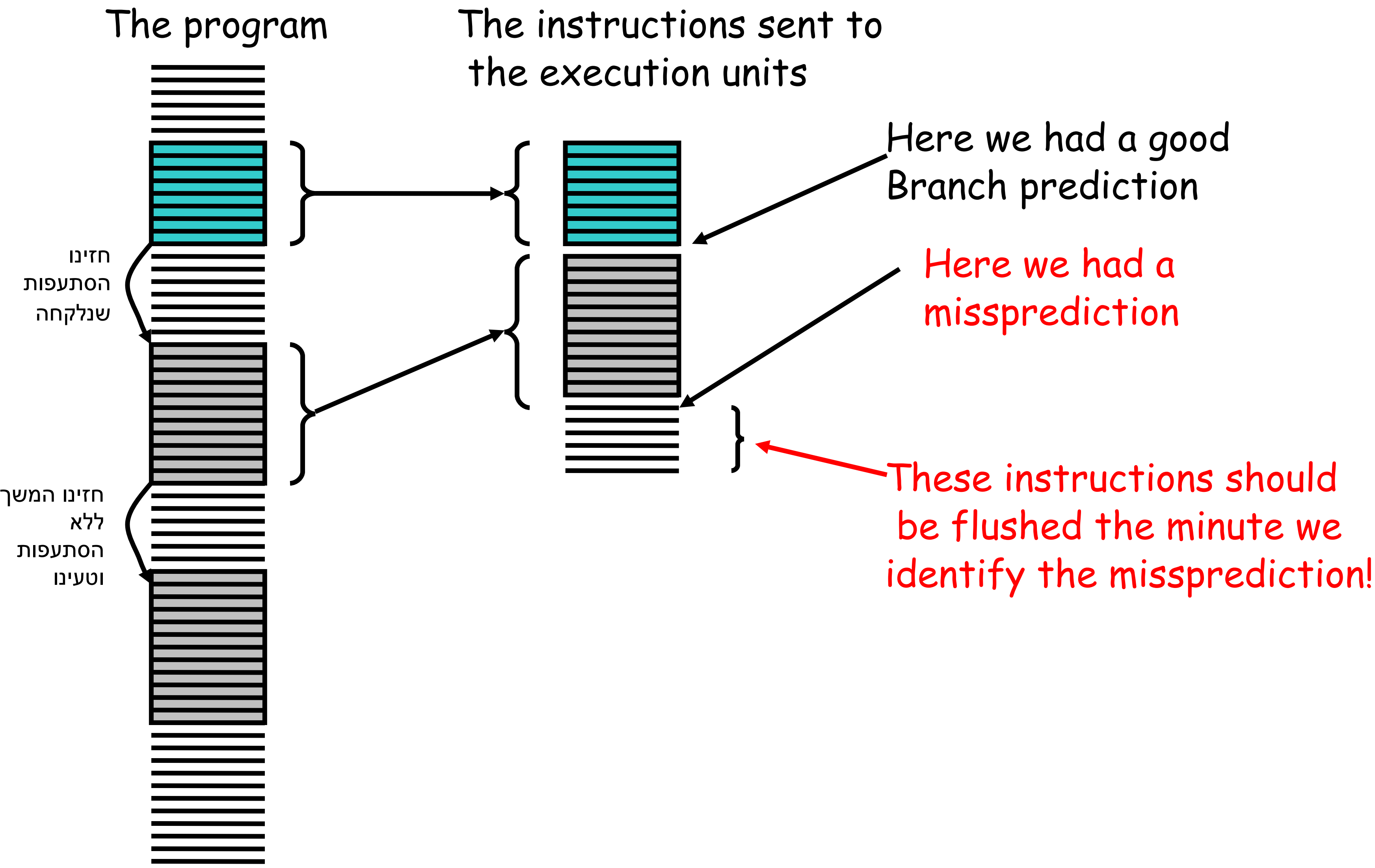
אנימציה

# HW support for precise interrupts

- Need HW buffer for results of uncommitted instructions: *reorder buffer*
  - 3 fields: instr, destination, value
  - Use reorder buffer number instead of reservation station when execution completes
  - Supplies the operands between execution complete & commit
  - (Reorder buffer can be operand source => more registers like RS)
  - Instructions commit
  - Once instruction commits, result is put into register
  - As a result, easy to undo speculated instructions on mispredicted branches or exceptions



# Graphic representation



The program

The instructions sent to the execution units

Here we had a good Branch prediction

Here we had a missprediction

These instructions should be flushed the minute we identify the missprediction!

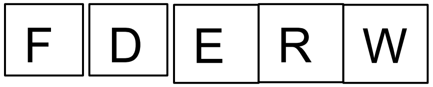
# Relationship between precise interrupts and specultation:

- Speculation is a form of guessing.
- Important for branch prediction:
  - Need to "take our best shot" at predicting branch direction.
- If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly:
  - This is exactly same as precise exceptions!
- Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*

End of 11 slides
on Tomasulo's algorithm
with reorder buffer

# An Exercise, with Precise Exceptions

```
MUL   R3 ← R1, R2
ADD   R5 ← R3, R4
ADD   R7 ← R2, R6
ADD   R10 ← R8, R9
MUL   R11 ← R7, R10
ADD   R5 ← R5, R11
```



- Assume ADD (4 cycle execute), MUL (6 cycle execute)
- Assume one adder and one multiplier
- How many cycles
  - in a non-pipelined machine
  - in an in-order-dispatch pipelined machine with reorder buffer (and full forwarding)
  - in an out-of-order dispatch pipelined machine with reorder buffer (full forwarding)

# Out-of-Order Execution with Precise Exceptions

- Idea: Use a reorder buffer to reorder instructions before committing them to architectural state

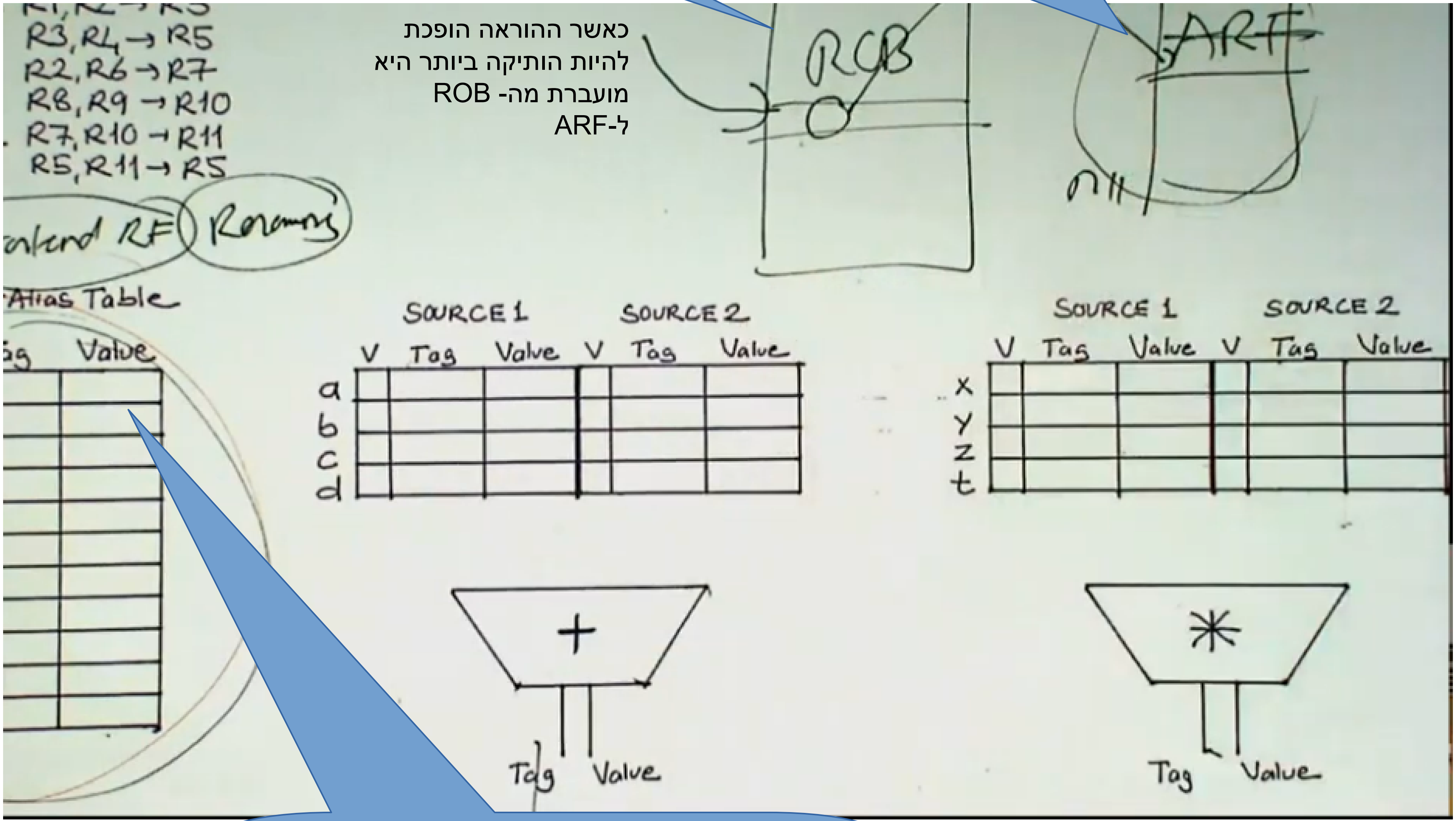
- An instruction updates the RAT when it completes execution
  - Also called frontend register file

- An instruction updates a **separate** architectural register file when it retires
  - i.e., when it is the oldest in the machine and has completed execution
  - In other words, the architectural register file is always updated in program order

- On an exception: flush pipeline, copy architectural register file into frontend register file

ROB

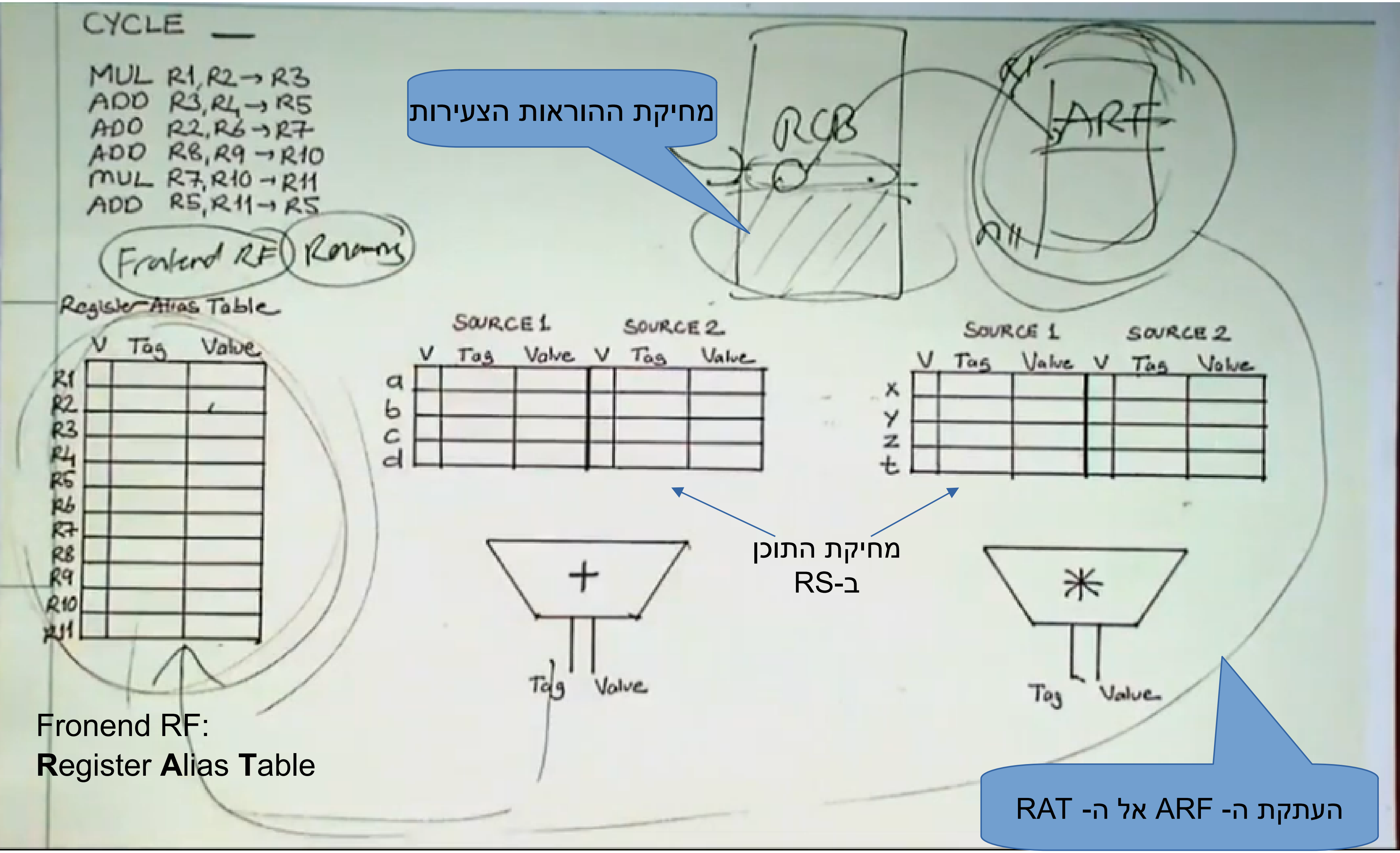
Architectural Register File
תמיד מתעדכן לפי סדר התכנית



כאשר ההוראה הופכת
להיות הותיקה ביותר היא
מועברת מה- ROB
ל-ARF

Frontend RF, used for renaming
זה לחלוטין מיקרוארכיטקטורה,
כלומר לא נראה ע"י המתכנת

# כאשר יש חריגה



```
MUL R1,R2 → R3
ADD R3,R4 → R5
ADD R2,R6 → R7
ADD R8,R9 → R10
MUL R7,R10 → R11
ADD R5,R11 → R5
```

Register Alias Table

| | V | Tag | Value |
|---|---|---|---|
| R1 | | | |
| R2 | | | |
| R3 | | | |
| R4 | | | |
| R5 | | | |
| R6 | | | |
| R7 | | | |
| R8 | | | |
| R9 | | | |
| R10 | | | |
| R11 | | | |

| | SOURCE 1 | | | SOURCE 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| a | | | | | | |
| b | | | | | | |
| c | | | | | | |
| d | | | | | | |

| | SOURCE 1 | | | SOURCE 2 | | |
|---|---|---|---|---|---|---|
| | V | Tag | Value | V | Tag | Value |
| x | | | | | | |
| y | | | | | | |
| z | | | | | | |
| t | | | | | | |

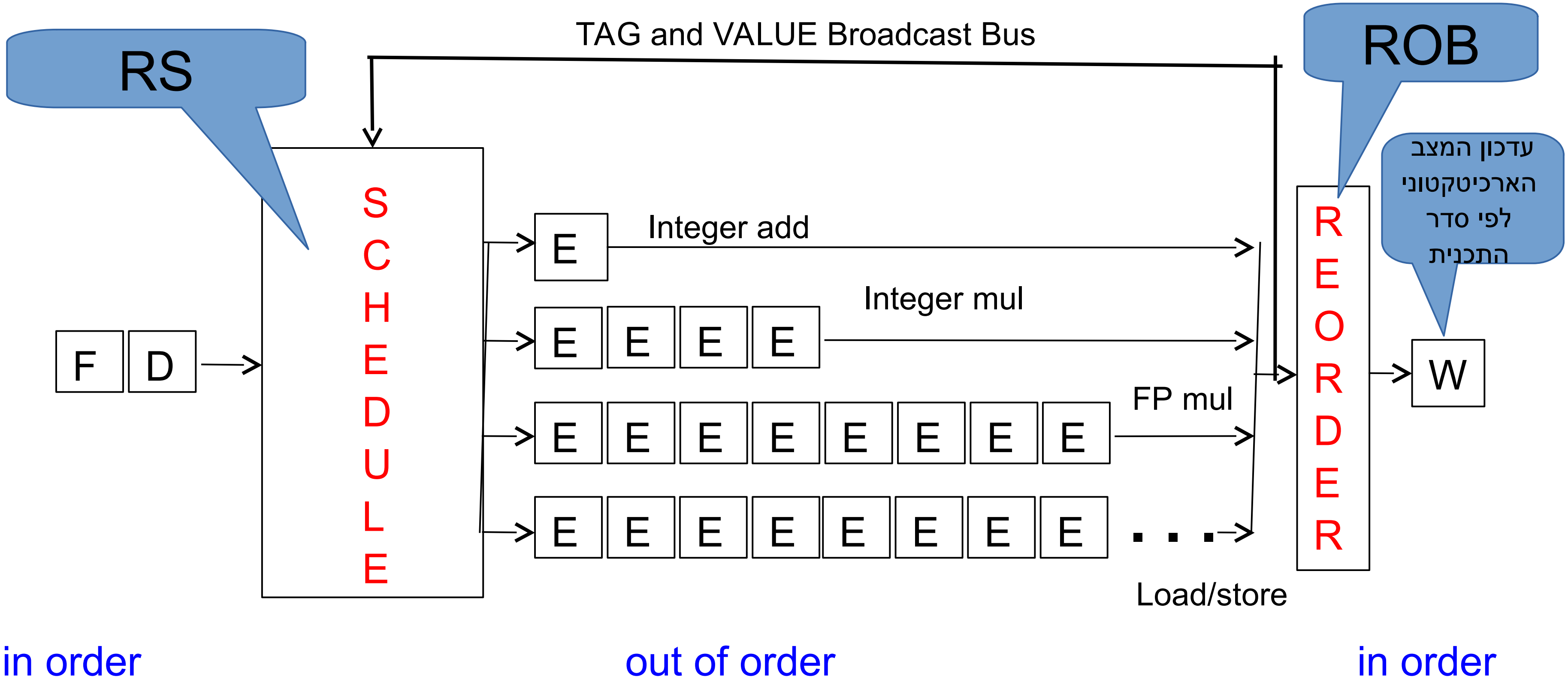
מחיקת ההוראות הצעירות

מחיקת התוכן
ב-RS

Fronend RF:
**R**egister **Al**ias **T**able

העתקת ה- ARF אל ה- RAT

# Out-of-Order Execution with Precise Exceptions

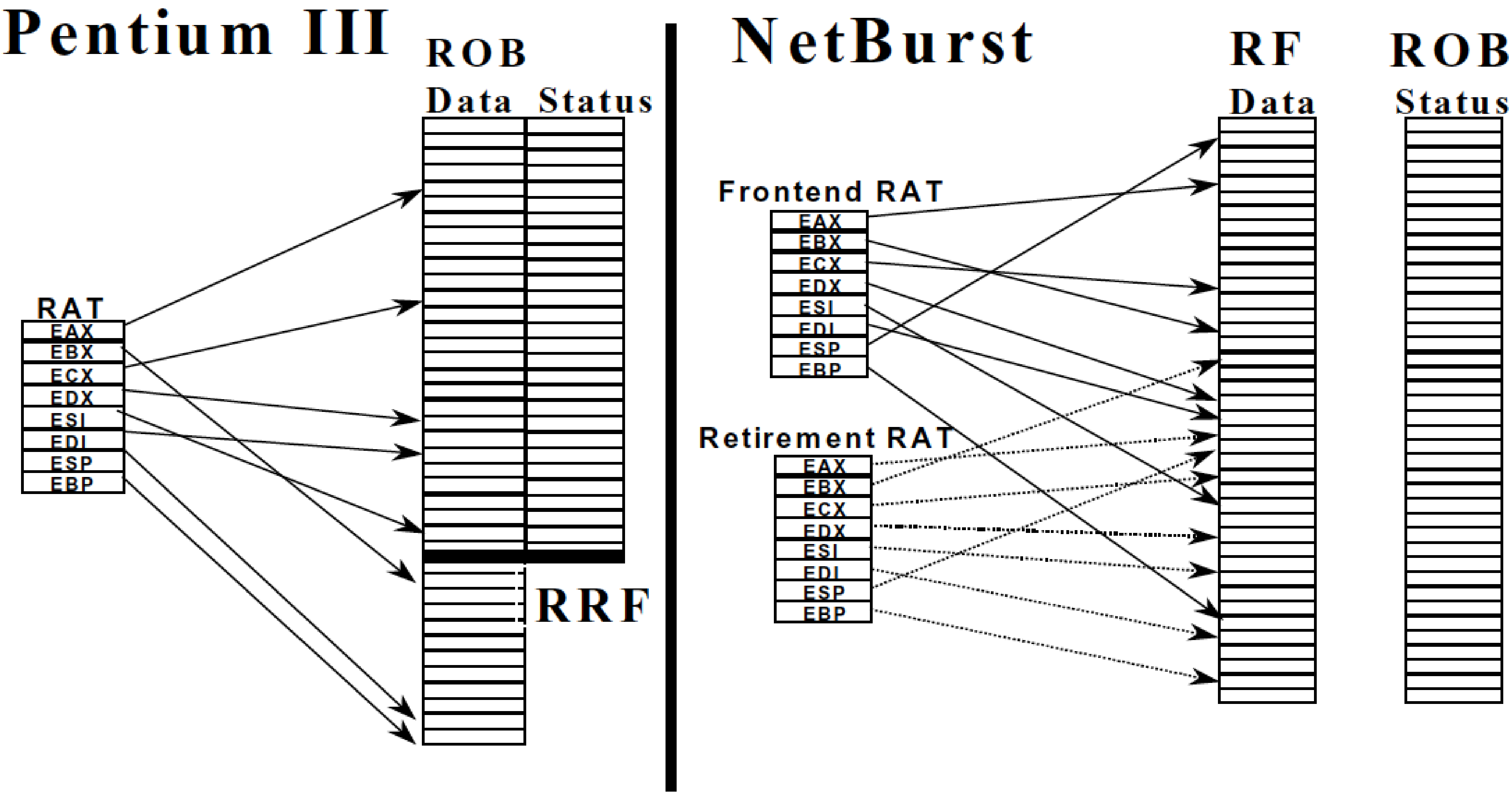


- Hump 1: Reservation stations (scheduling window)
- Hump 2: Reordering (reorder buffer, aka instruction window or active window)

# Modern OoO Execution w/ Precise Exceptions

- Most modern processors use
  - Reorder buffer to support in-order retirement of instructions
  - **A single register file** to store registers (speculative and architectural) – INT and FP are still separate
  - Future register map → used for renaming
  - Architectural register map → used for state recovery

# An Example from Modern Processors



Boggs et al., "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

# Enabling OoO Execution, Revisited

1. Link the consumer of a value to the producer
   - Register renaming: Associate a "tag" with each data value

2. Buffer instructions until they are ready
   - Insert instruction into reservation stations after renaming

3. Keep track of readiness of source values of an instruction
   - Broadcast the "tag" when the value is produced
   - Instructions compare their "source tags" to the broadcast tag → if match, source value becomes ready

4. When all source values of an instruction are ready, dispatch the instruction to functional unit (FU)
   - Wakeup and select/schedule the instruction

# Summary of OoO Execution Concepts

- Register renaming eliminates false dependencies, enables linking of producer to consumers

שינוי שם של רגיסטרים מעלים תלויות מדומות ומאפשר לקשור בין “היצרן” לבין
“הצרכן"

- Buffering enables the pipeline to move for independent ops

שימוש בבאפר מאפשר לקדם הוראות בלתי תלויות בפייפליין

- Tag broadcast enables communication (of readiness of produced value) between instructions

שידור התגים מאפשר תקשורת אודות מוכנות הערכים שחושבו בין ההוראות

- Wakeup and select enables out-of-order dispatch

מנגנון ההתעוררות ובחירת ההוראה מאפשר סיום שלא עפ"י הסדר

# OoO Execution: Restricted Dataflow

- An out-of-order engine dynamically builds the dataflow graph of a piece of the program
  - which piece?

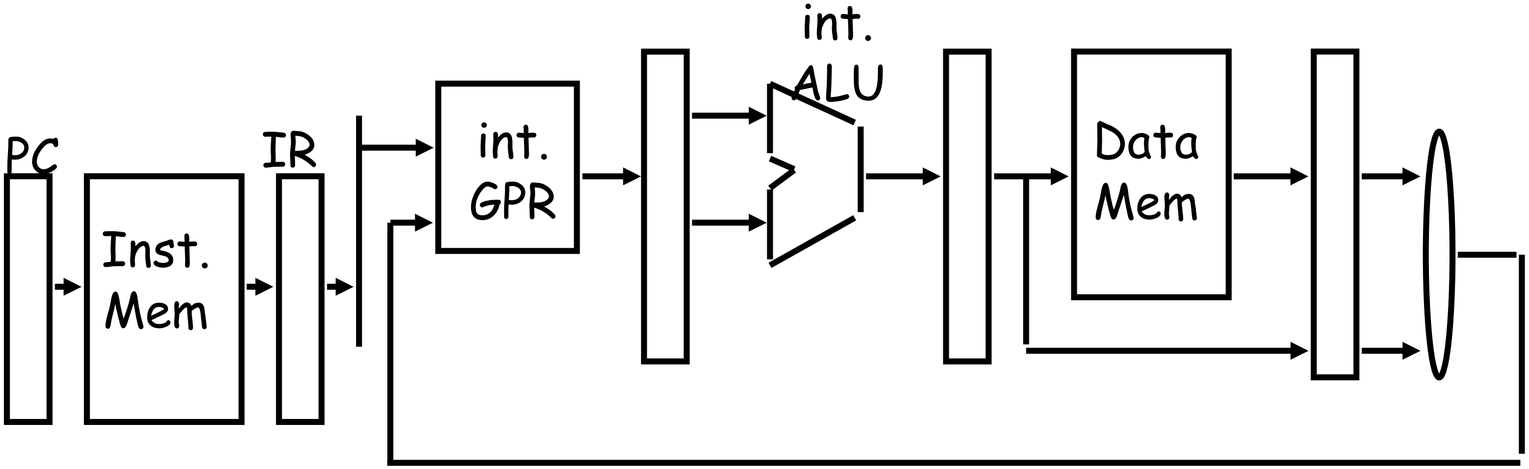
מושג חדש!

- The dataflow graph is limited to the **instruction window**
  - Instruction window: all decoded but not yet retired instructions

- Can we do it for the whole program?
- Why would we like to?
- In other words, how can we have a large instruction window?
- Can we do it efficiently with Tomasulo's algorithm?

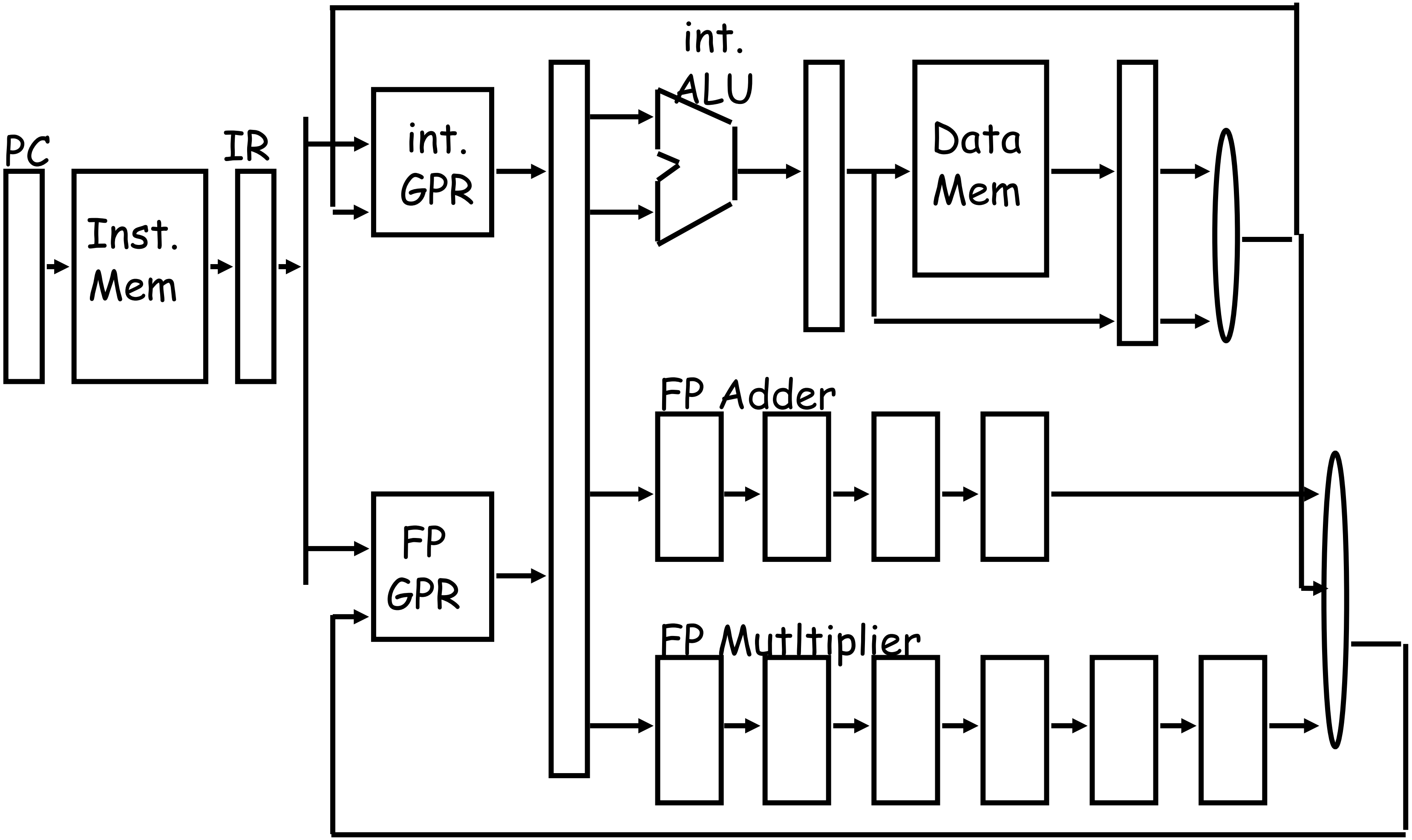
# How we do that in MIPS (next 6 slides)

# Simple integer MIPS



Integer instruction only that are issued in-order
Execute in-order and completed in-order (5 CKs for all inst.)

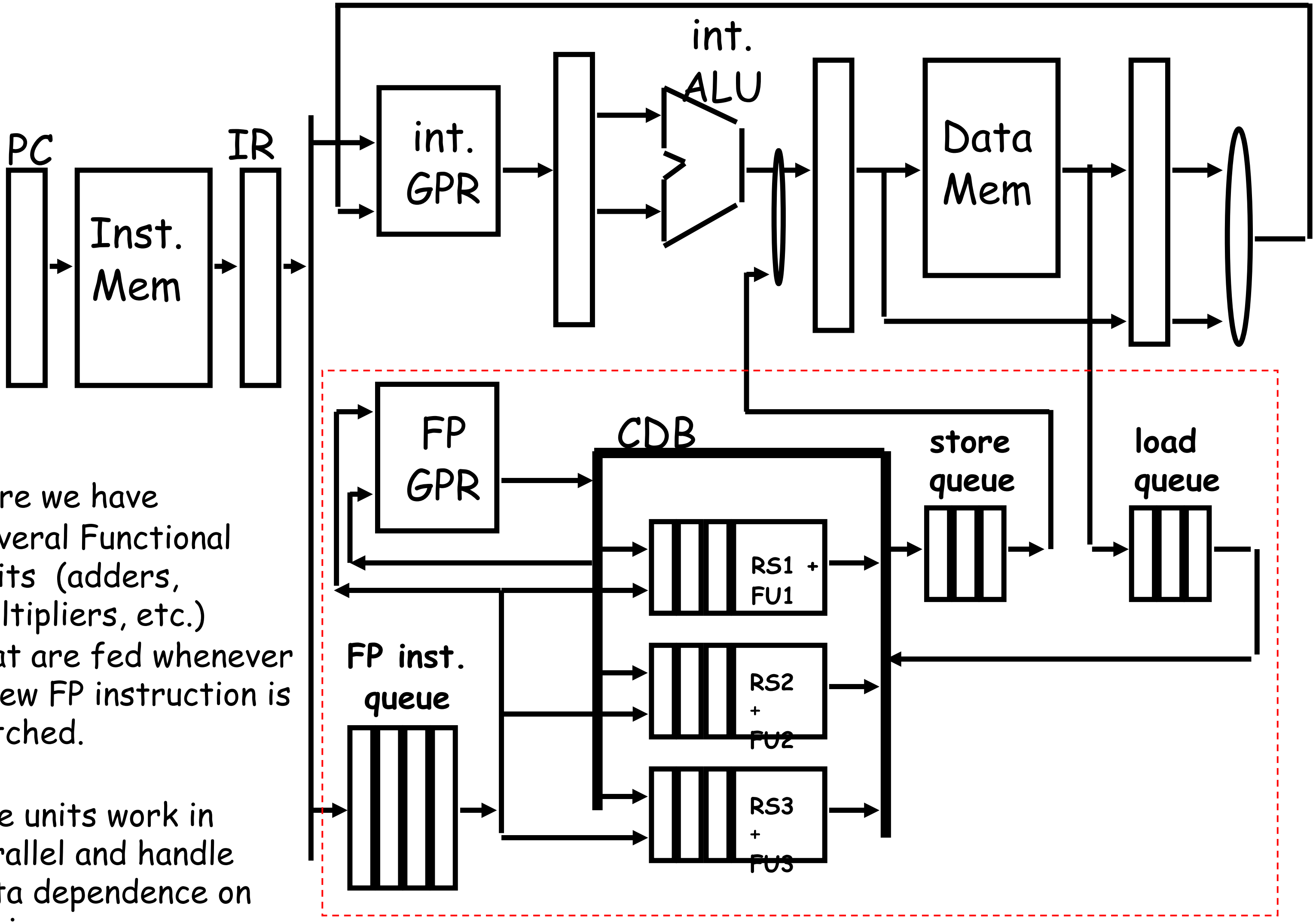
# MIPS with FP pipeline



Integer & FP instruction are issued in-order
Dif. In length causes Out-Of-Order completion

X2אנימציה

# MIPS with Tomasulo style ILP



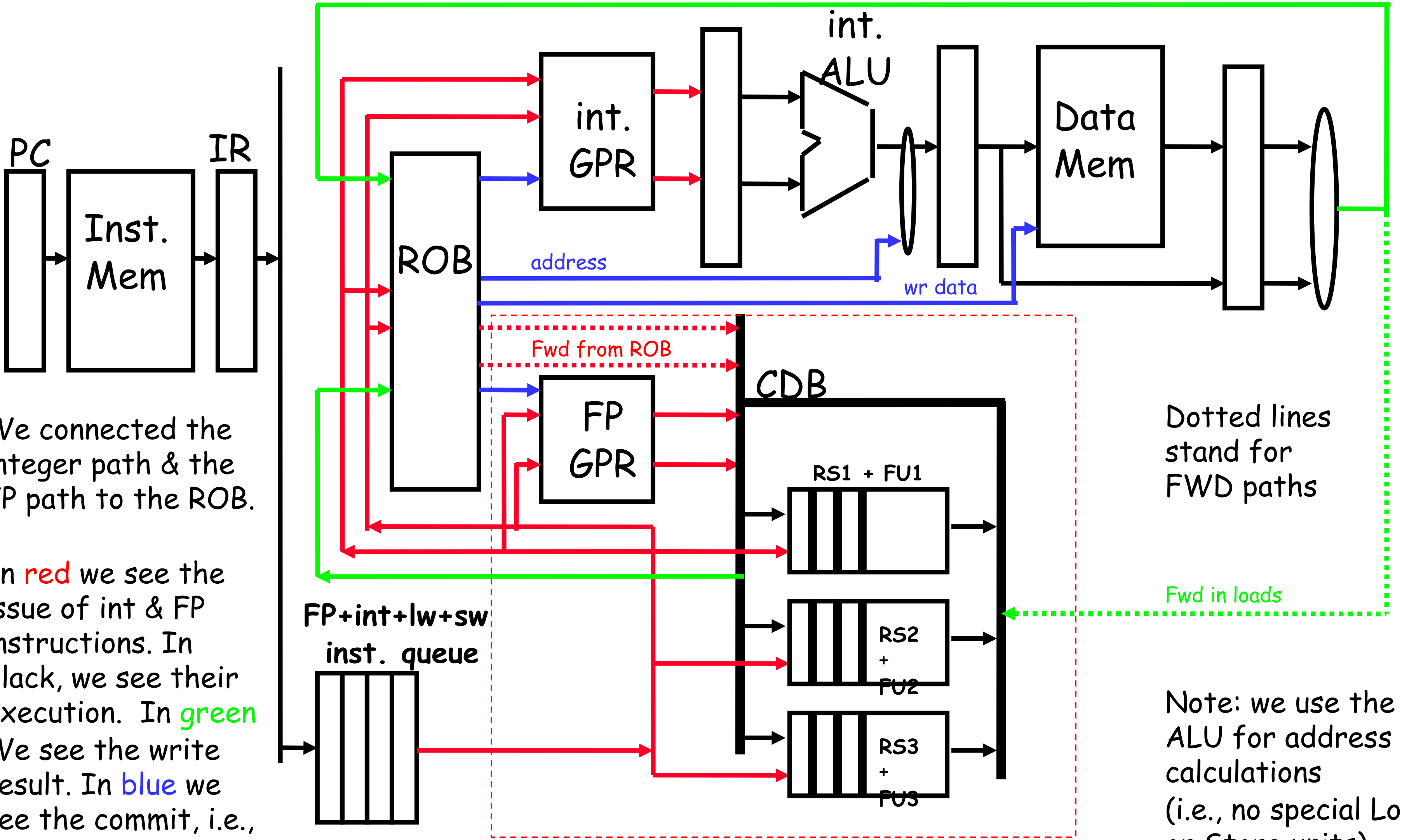
Here we have Several Functional Units (adders, Multipliers, etc.) that are fed whenever a new FP instruction is fetched.

The units work in parallel and handle data dependence on their own.

FP path is separated from Integer path and uses OOO execution and completion

אנימציה x4

# MIPS with speculative Tomasulo ILP



We connected the integer path & the FP path to the ROB.

In red we see the issue of int & FP instructions. In black, we see their execution. In green We see the write result. In blue we see the commit, i.e., copy to GPRs or to Data Mem

Dotted lines stand for FWD paths

Note: we use the int ALU for address calculations (i.e., no special Load or Store units)

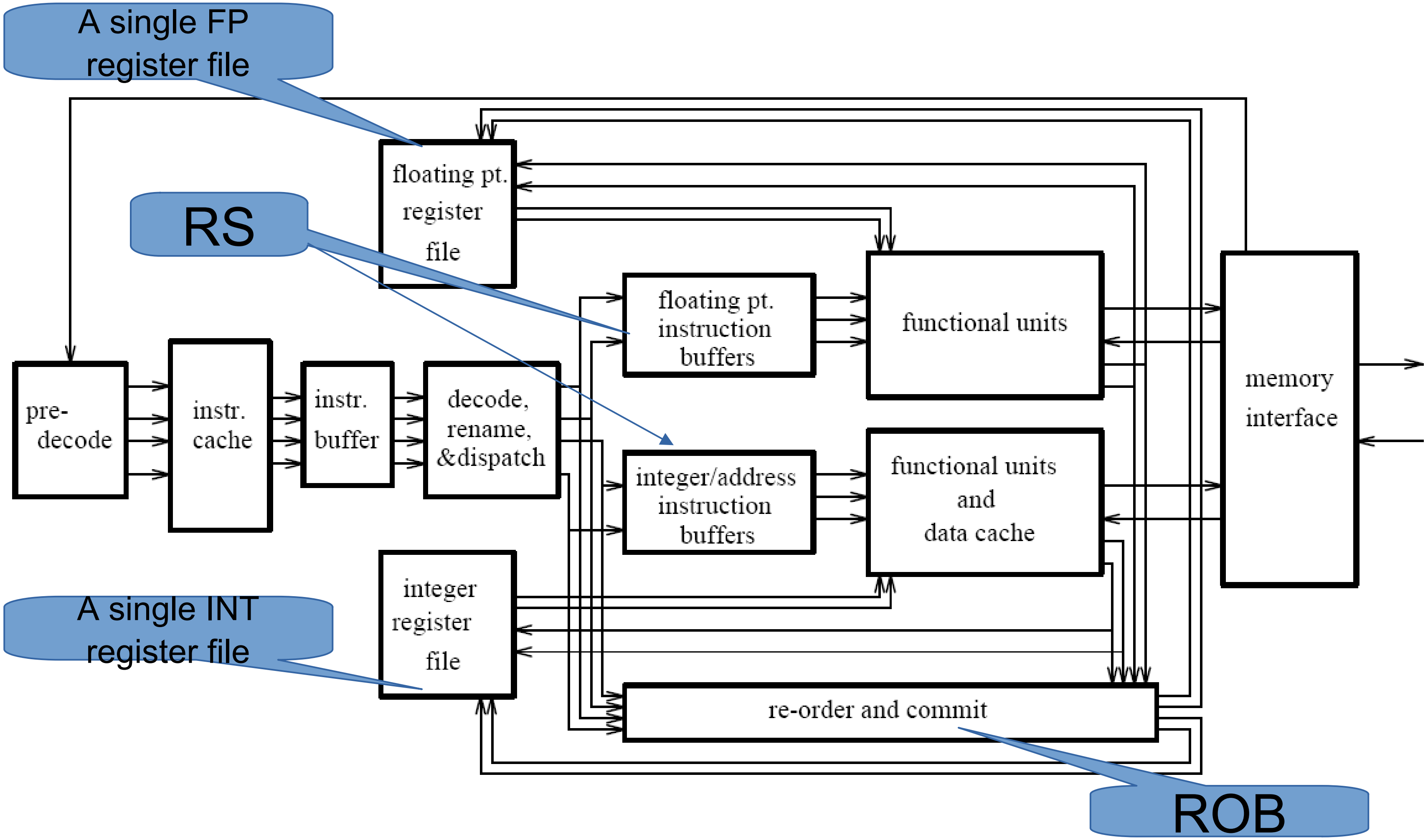
Here we have in-order issue of FP & Int instructions, OOO execution and write result, but in-order completion

# Questions to Ponder

- Why is OoO execution beneficial?
  - What if all operations take a single cycle?
  - Latency tolerance: OoO execution tolerates the latency of multi-cycle operations by executing independent operations concurrently

- What if an instruction takes 1000 cycles?
  - How large of an instruction window do we need to continue decoding?
  - How many cycles of latency can OoO tolerate?
  - What limits the latency tolerance scalability of Tomasulo's algorithm?
    - Instruction window size: how many decoded but not yet retired instructions you can keep in the machine.

# General Organization of an OoO Processor


A single FP register file
RS
A single INT register file
ROB
floating pt. register file
floating pt. instruction buffers
functional units
memory interface
pre-decode
instr. cache
instr. buffer
decode, rename, &dispatch
integer/address instruction buffers
functional units and data cache
integer register file
re-order and commit

- **Smith and Sohi**, "The Microarchitecture of Superscalar Processors," Proc. IEEE, Dec. 1995.

# A Modern OoO Design: Intel Pentium 4

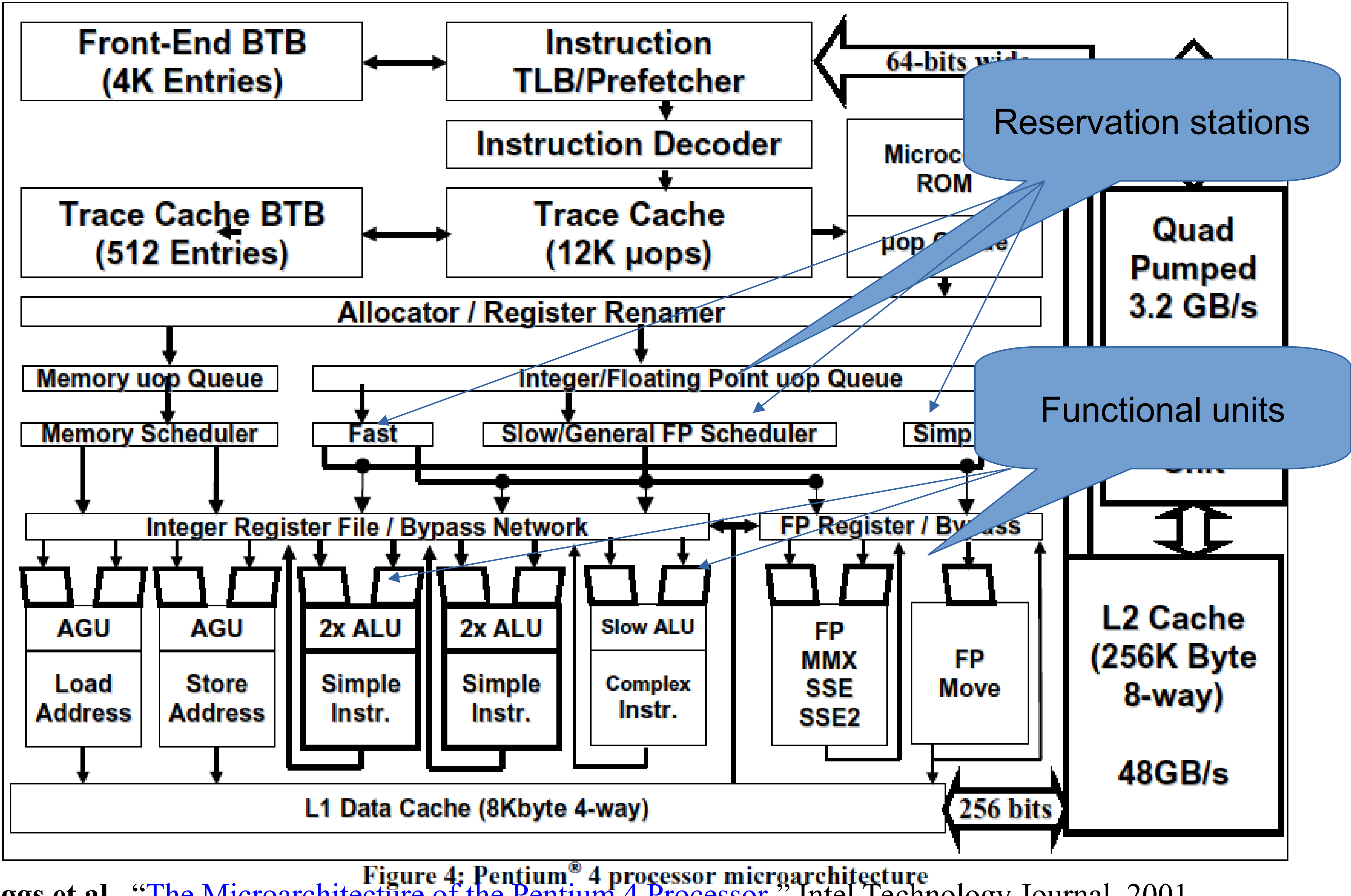
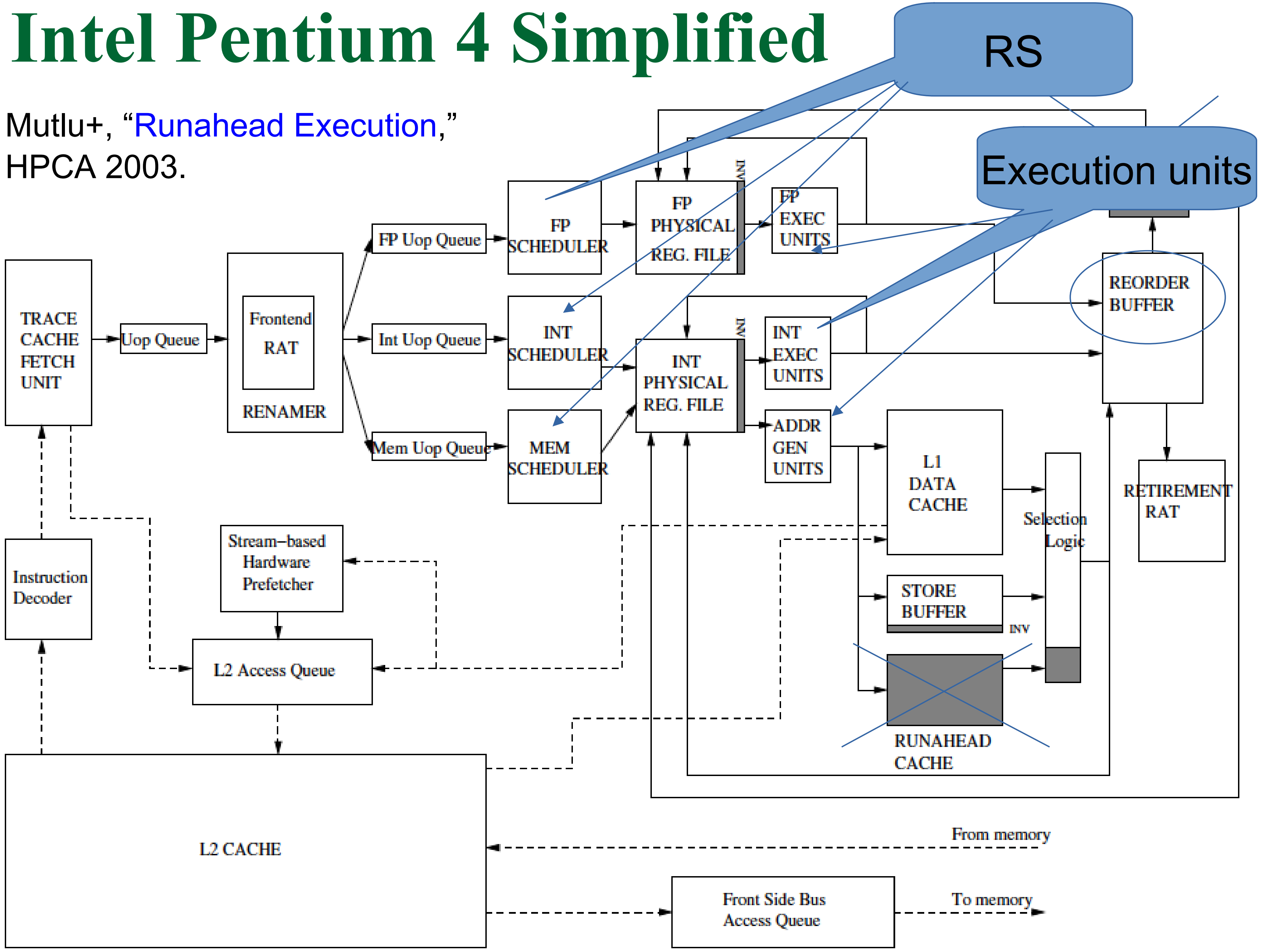


Figure 4: Pentium® 4 processor microarchitecture

**Boggs et al.**, "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

# Intel Pentium 4 Simplified

Mutlu+, “Runahead Execution,” HPCA 2003.



# Alpha 21264

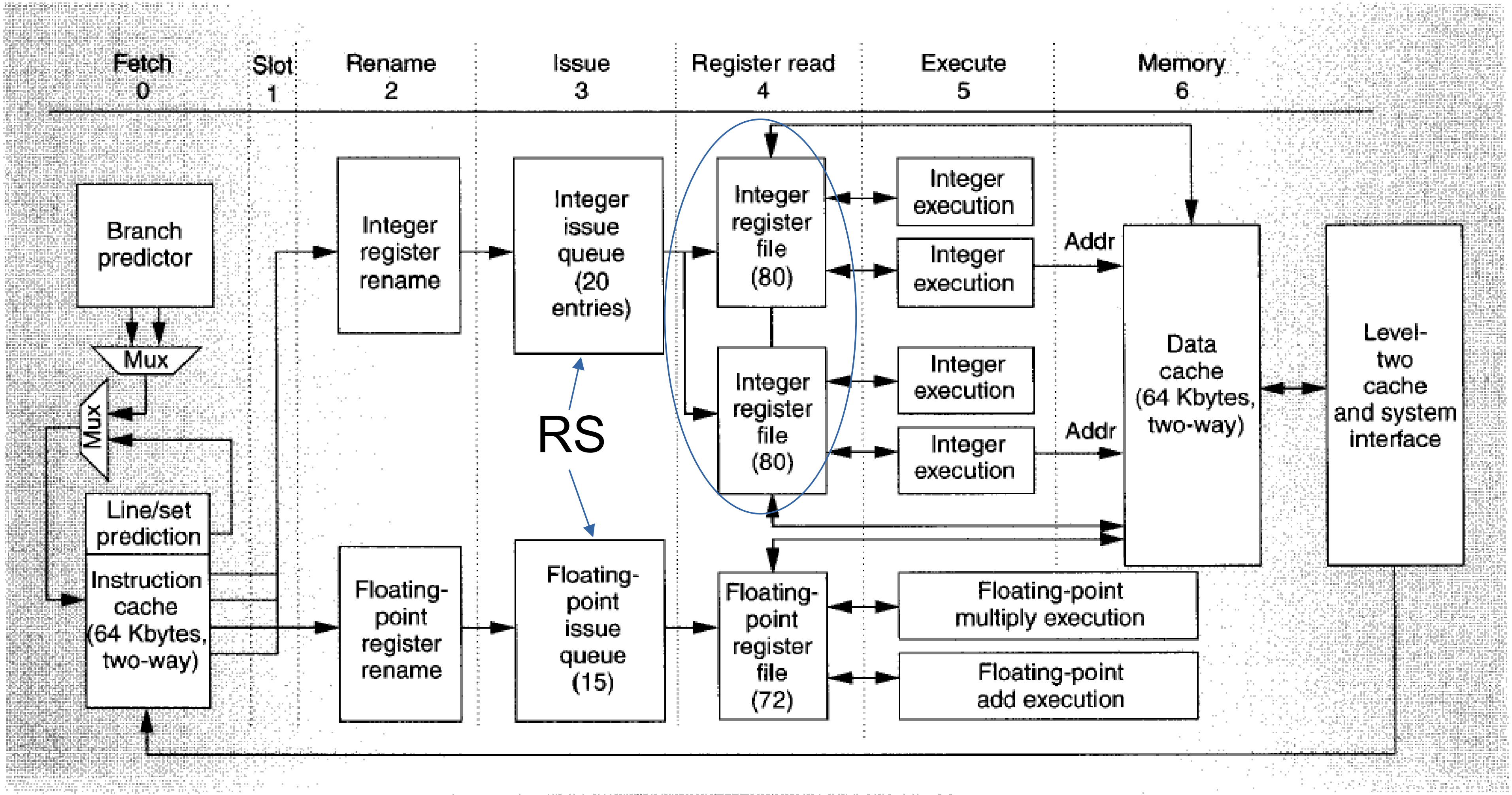

Fetch
0
Slot
1
Rename
2
Issue
3
Register read
4
Execute
5
Memory
6
Branch predictor
Mux
Mux
Line/set prediction
Instruction cache (64 Kbytes, two-way)
Integer register rename
Integer issue queue (20 entries)
RS
Integer register file (80)
Integer register file (80)
Integer execution
Integer execution
Integer execution
Integer execution
Addr
Addr
Data cache (64 Kbytes, two-way)
Level-two cache and system interface
Floating-point register rename
Floating-point issue queue (15)
Floating-point register file (72)
Floating-point multiply execution
Floating-point add execution

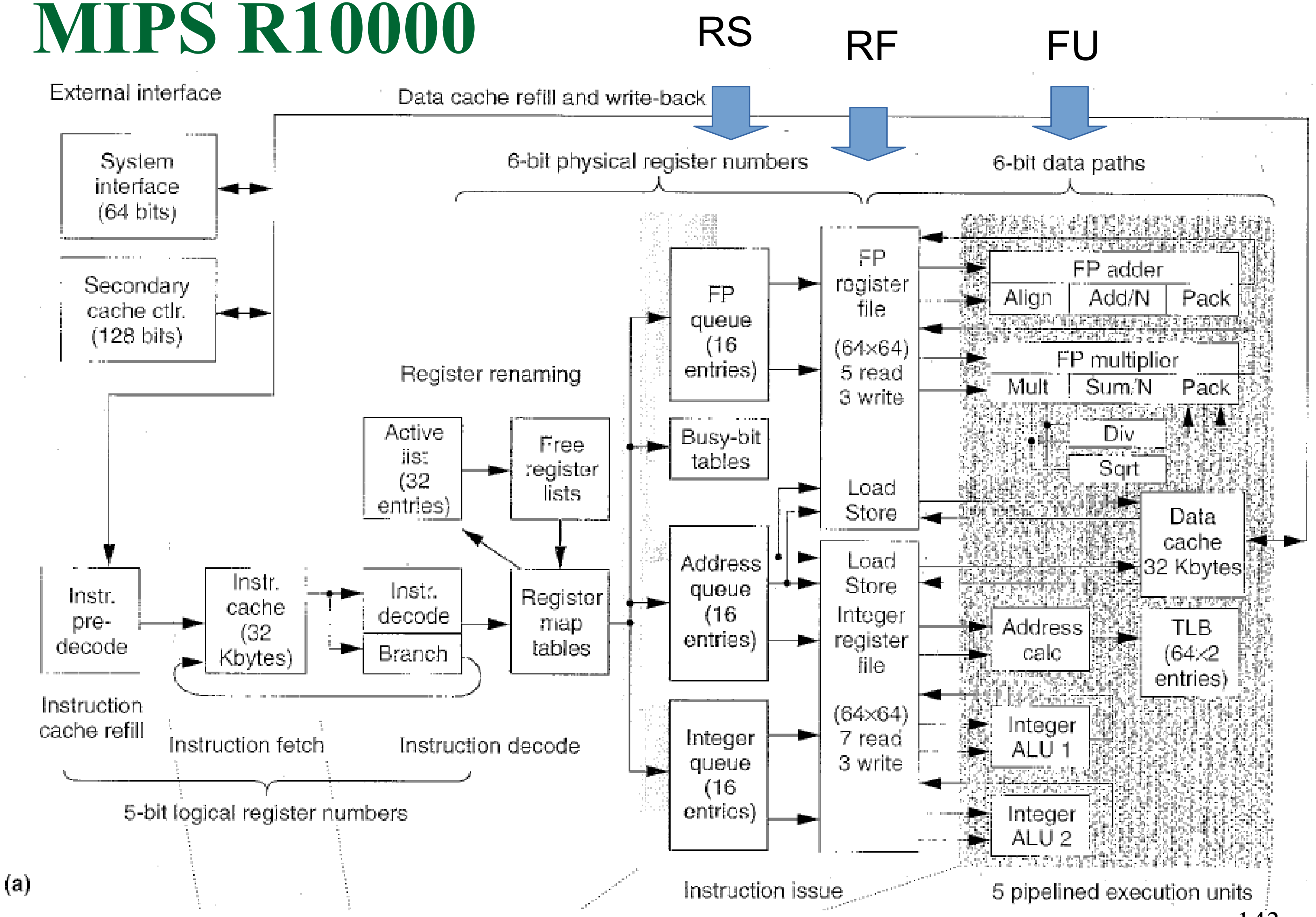
Figure 2. Stages of the Alpha 21264 instruction pipeline.

Kessler, "The Alpha 21264 Microprocessor," IEEE Micro, March-April 1999.

# Alpha 21264

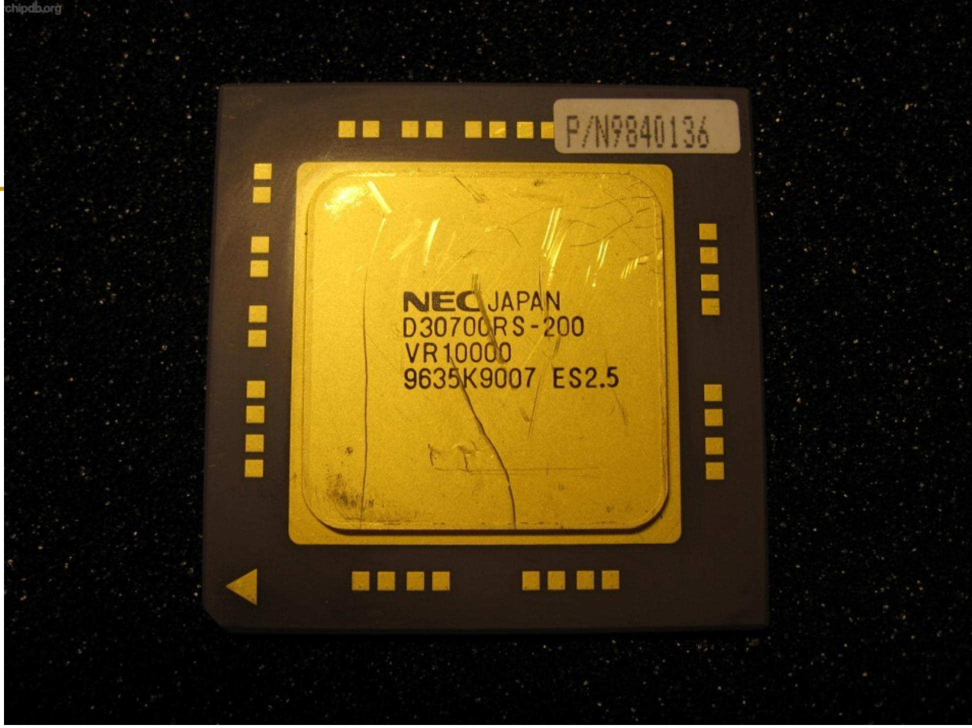
Compaq Alpha Server

# MIPS R10000



Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, April 1996

# MIPS R10000


chipdb.org
P/N9840136
NEC JAPAN
D30700RS-200
VR10000
9635K9007 ES2.5

silicon graphics octane2
sgi
sgi

# IBM POWER 4

Tendler et al., “POWER4 system microarchitecture,” IBM J R&D, 2002.

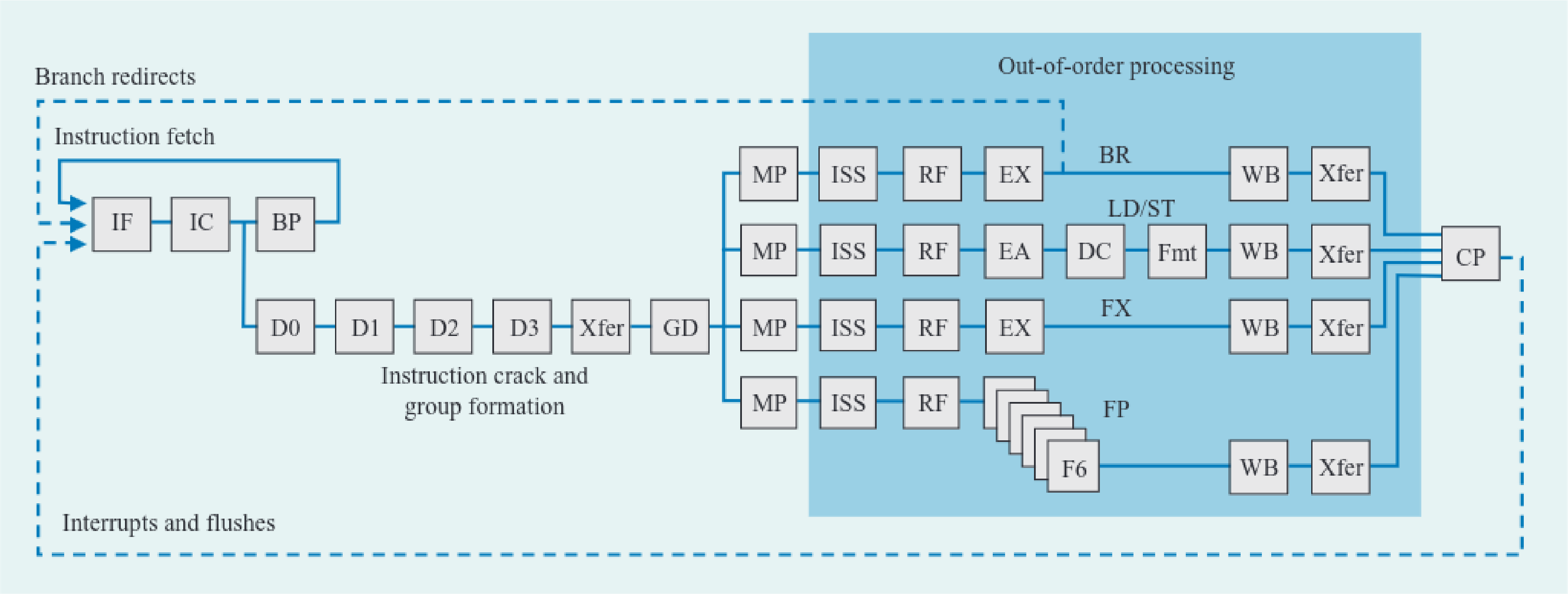


Figure 4

POWER4 instruction execution pipeline.

F = instruction fetch, IC = instruction cache, BP = branch predict, D0 = decode stage 0, Xfer = transfer, GD = group dispatch, MP = mapping, ISS = instruction issue, RF = register file read, EX = execute, EA = compute address, DC = data caches, F6 = six-cycle floating-point execution pipe, Fmt = data format, WB = write back, and CP = group commit

# IBM POWER4

- 2 cores, out-of-order execution
- 100-entry instruction window in each core
- 8-wide instruction fetch, issue, execute
- Large, local+global hybrid branch predictor
- 1.5MB, 8-way L2 cache

# IBM POWER5

- Kalla et al., “IBM Power5 Chip: A Dual-Core Multithreaded Processor,” IEEE Micro 2004.

RS

ROB

SMT: 2 threads

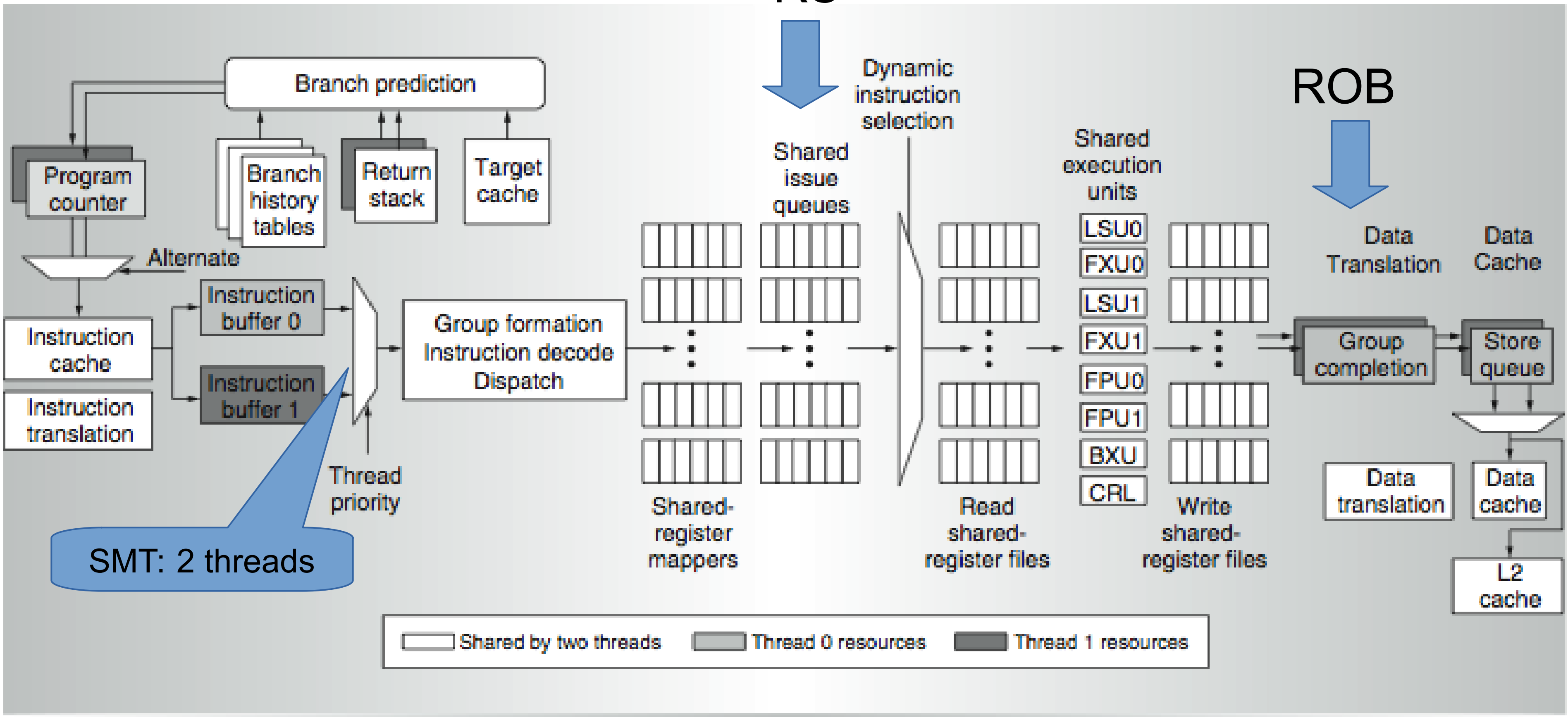


Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

# Handling Out-of-Order Execution of Loads and Stores

# Registers versus Memory

- So far, we considered mainly registers as part of state

- What about memory?

- What are the fundamental differences between registers and memory?
  - Register dependences known statically – memory dependences determined dynamically
  - Register state is small – memory state is large
  - Register state is not visible to other threads/processors – memory state is shared between threads/processors (in a shared memory multiprocessor)

# Memory Dependence Handling (I)

- Need to obey memory dependences in an out-of-order machine
  - and need to do so while providing high performance

- Observation and Problem: Memory address is not known until a load/store executes

- Corollary 1: Renaming memory addresses is difficult
- Corollary 2: Determining dependence or independence of loads/stores has to be handled *after* their (partial) execution
- Corollary 3: When a load/store has its address ready, there may be older/younger stores/loads with unknown addresses in the machine

# Memory Dependence Handling (II)

- When do you schedule a load instruction in an OOO engine?
  - Problem: A younger load can have its address ready before an older store's address is known
  - Known as the memory disambiguation problem or the unknown address problem

- Approaches
  - Conservative: Stall the load until all previous stores have computed their addresses (or even retired from the machine)
  - Aggressive: Assume load is independent of unknown-address stores and schedule the load right away
  - Intelligent: Predict (with a more sophisticated predictor) if the load is dependent on any unknown address store

# Handling of Store-Load Dependences

- **A load's dependence status is not known** until all previous store addresses are available.

- How does the OOO engine detect dependence of a load instruction on a previous store?
  - Option 1: Wait until all previous stores committed (no need to check for address match)
  - Option 2: Keep a list of pending stores in a store buffer and check whether load address matches a previous store address

- How does the OOO engine treat the scheduling of a load instruction wrt previous stores?
  - Option 1: Assume load dependent on all previous stores
  - Option 2: Assume load independent of all previous stores
  - Option 3: Predict the dependence of a load on an outstanding store

# Memory Disambiguation (I)

- Option 1: Assume load is dependent on all previous stores

  + No need for recovery

  -- Too conservative: delays independent loads unnecessarily

- Option 2: Assume load is independent of all previous stores

  + Simple and can be common case: no delay for independent loads

  -- Requires recovery and re-execution of load and dependents on misprediction

- Option 3: Predict the dependence of a load on an outstanding store

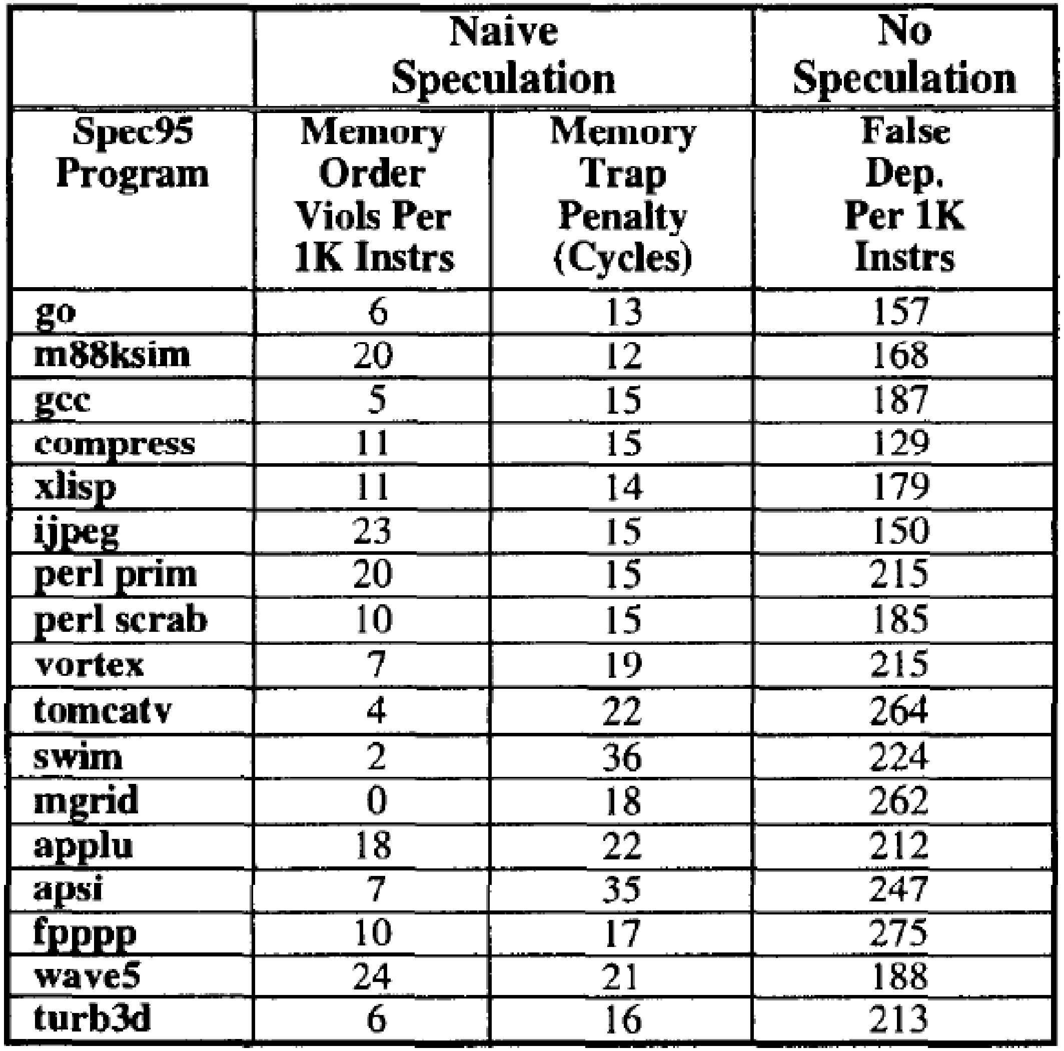
  + More accurate. Load store dependencies persist over time

  -- Still requires recovery/re-execution on misprediction

  - Alpha 21264 : Initially assume load independent, delay loads found to be dependent
  - Moshovos et al., "Dynamic speculation and synchronization of data dependences," ISCA 1997.
  - Chrysos and Emer, "Memory Dependence Prediction Using Store Sets," ISCA 1998.

# Memory Disambiguation (II)

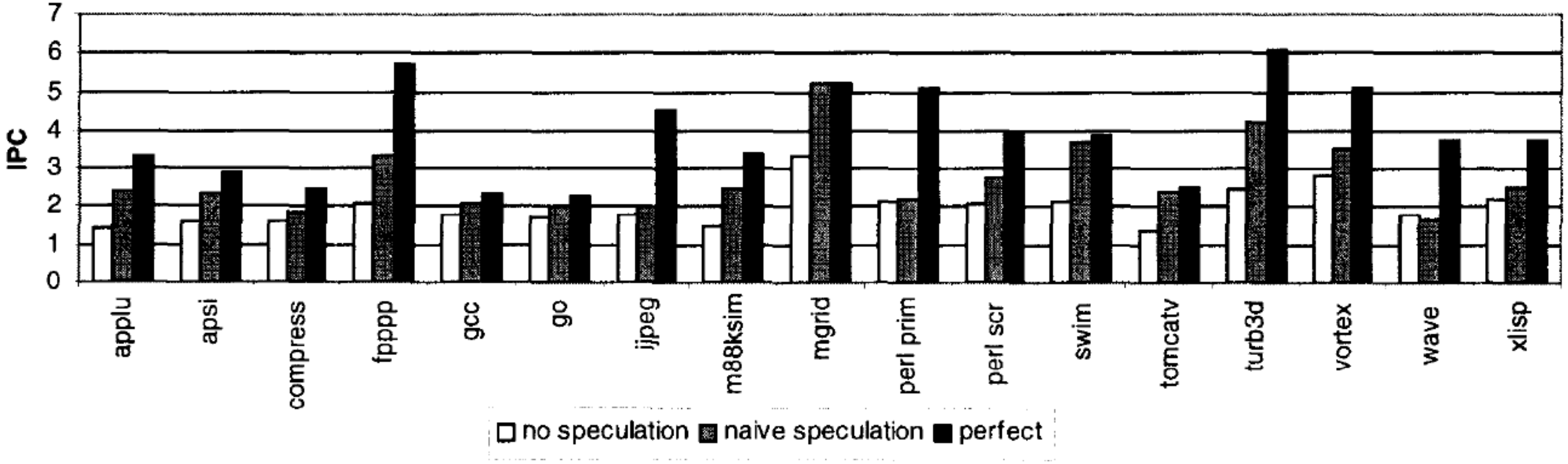
- Chrysos and Emer, "Memory Dependence Prediction Using Store Sets," ISCA 1998.



| | Naive Speculation | | No Speculation |
|---|---|---|---|
| Spec95 Program | Memory Order Viols Per 1K Instrs | Memory Trap Penalty (Cycles) | False Dep. Per 1K Instrs |
| go | 6 | 13 | 157 |
| m88ksim | 20 | 12 | 168 |
| gcc | 5 | 15 | 187 |
| compress | 11 | 15 | 129 |
| xlisp | 11 | 14 | 179 |
| ijpeg | 23 | 15 | 150 |
| perl prim | 20 | 15 | 215 |
| perl scrab | 10 | 15 | 185 |
| vortex | 7 | 19 | 215 |
| tomcatv | 4 | 22 | 264 |
| swim | 2 | 36 | 224 |
| mgrid | 0 | 18 | 262 |
| applu | 18 | 22 | 212 |
| apsi | 7 | 35 | 247 |
| fpppp | 10 | 17 | 275 |
| wave5 | 24 | 21 | 188 |
| turb3d | 6 | 16 | 213 |

# Memory Disambiguation (II)

- Chrysos and Emer, "Memory Dependence Prediction Using Store Sets," ISCA 1998.



- Predicting store-load dependencies important for performance
- Simple predictors (based on past history) can achieve most of the potential performance

# Data Forwarding Between Stores and Loads

- We cannot update memory out of program order

  → Need to buffer all store and load instructions in instruction window

- Even if we **know** all addresses of past stores when we generate the address of a load, two questions still remain:

  1. How do we check whether or not it is dependent on a store
  2. How do we forward data to the load if it is dependent on a store

- Modern processors use a LQ (load queue) and a SQ for this
  - Can be combined or separate between loads and stores
  - A load searches the SQ after it computes its address. Why?
  - A store searches the LQ after it computes its address. Why?

# Out-of-Order Completion of Memory Ops

- When a store instruction finishes execution, it writes its address and data in its reorder buffer entry (or SQ entry)

- When a later load instruction generates its address, it:
  - searches the SQ with its address
  - accesses memory with its address
  - receives the value from the youngest older instruction that wrote to that address (either from ROB or memory)

- This is a complicated "search logic" implemented as a Content Addressable Memory
  - Content is "memory address" (but also need *size* and *age*)
  - Called **store-to-load forwarding logic**

# Store-Load Forwarding Complexity

- Content Addressable Search (based on Load Address)

- Range Search (based on Address and Size of both the Load and earlier Stores)

- Age-Based Search (for last written values)

- Load data can come from a combination of multiple places
  - One or more stores in the Store Buffer (SQ)
  - Memory/cache

# Step by step for

# Dynamic Scheduling by reorder buffer

(13 slides)

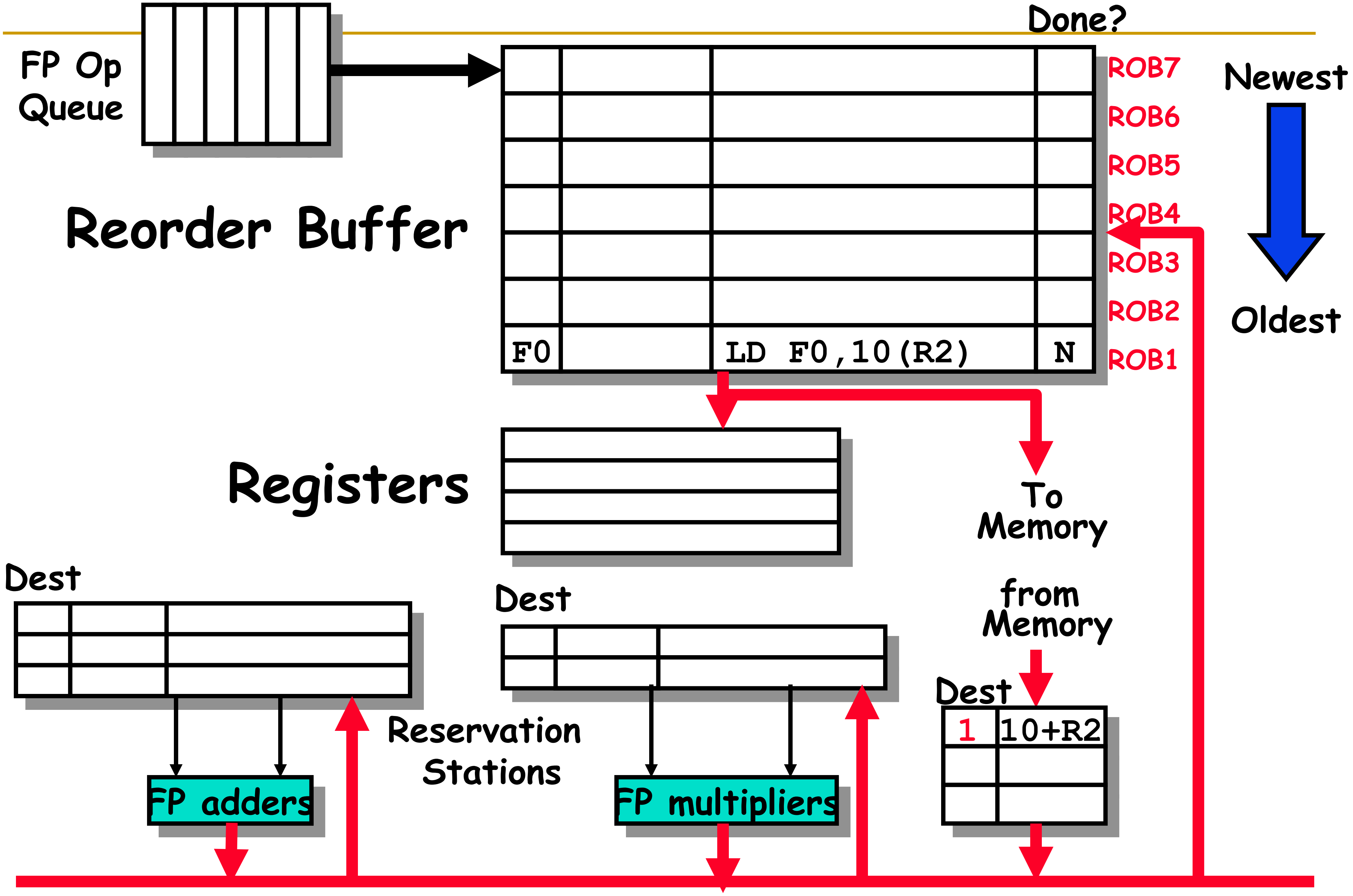
אנימציה

# Four Steps of Speculative Tomasulo Algorithm

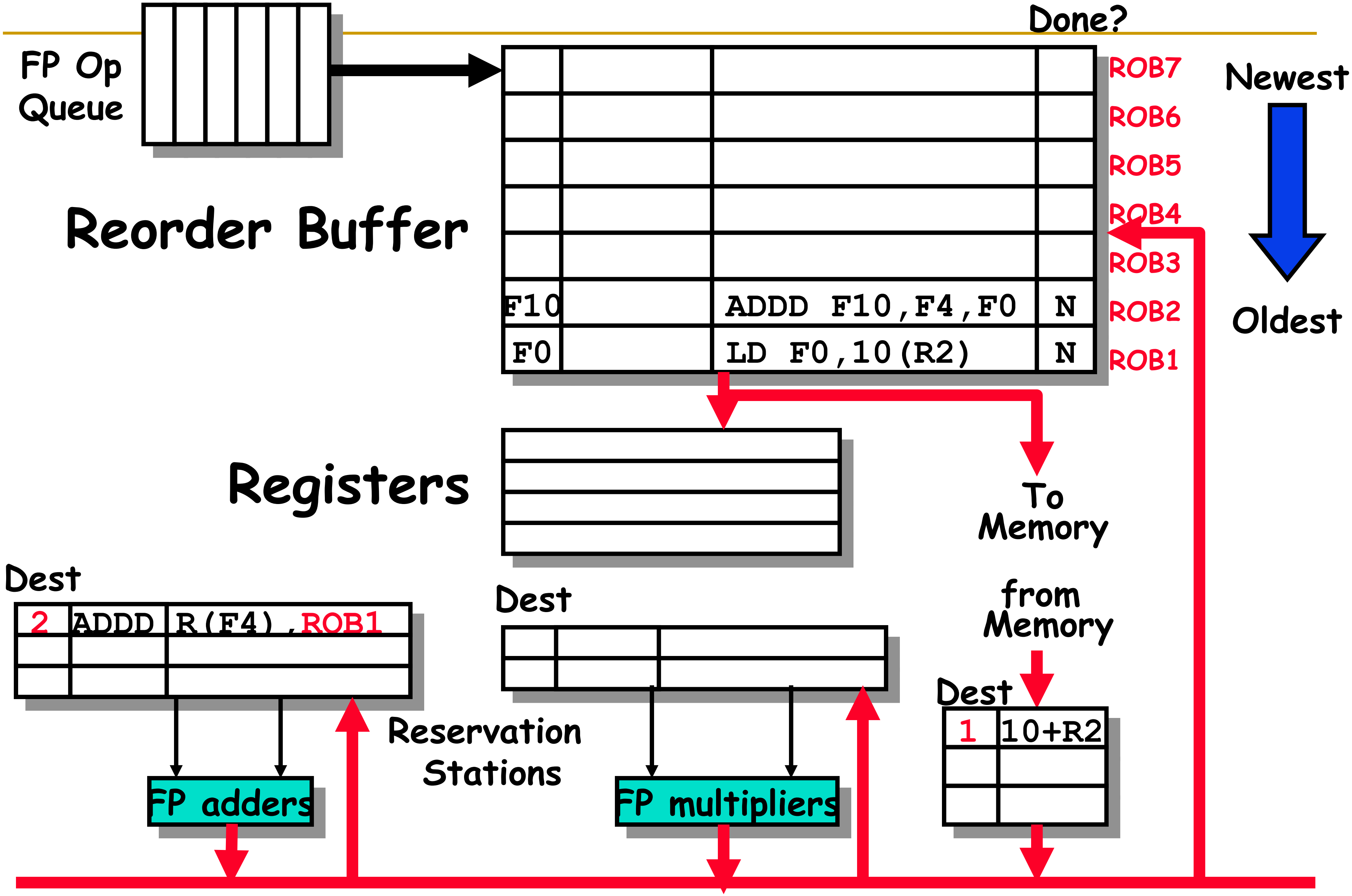
תזכורת:

1. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. Commit—update register with reorder result
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.
   - Mispredicted branch or interrupt flushes reorder buffer (sometimes called "graduation")

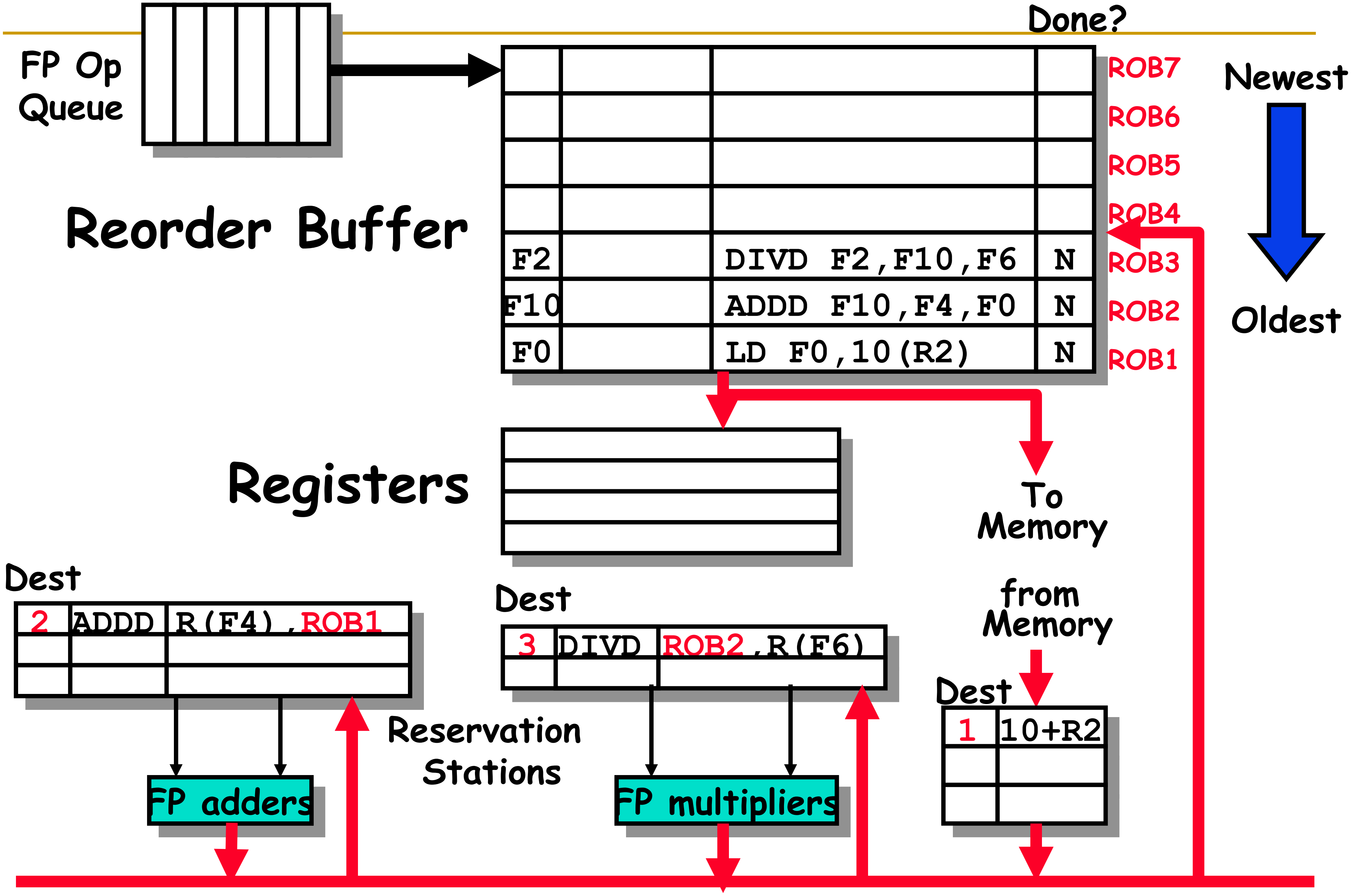
# Tomasulo With Reorder buffer:



# Tomasulo With Reorder buffer:



# Tomasulo With Reorder buffer:



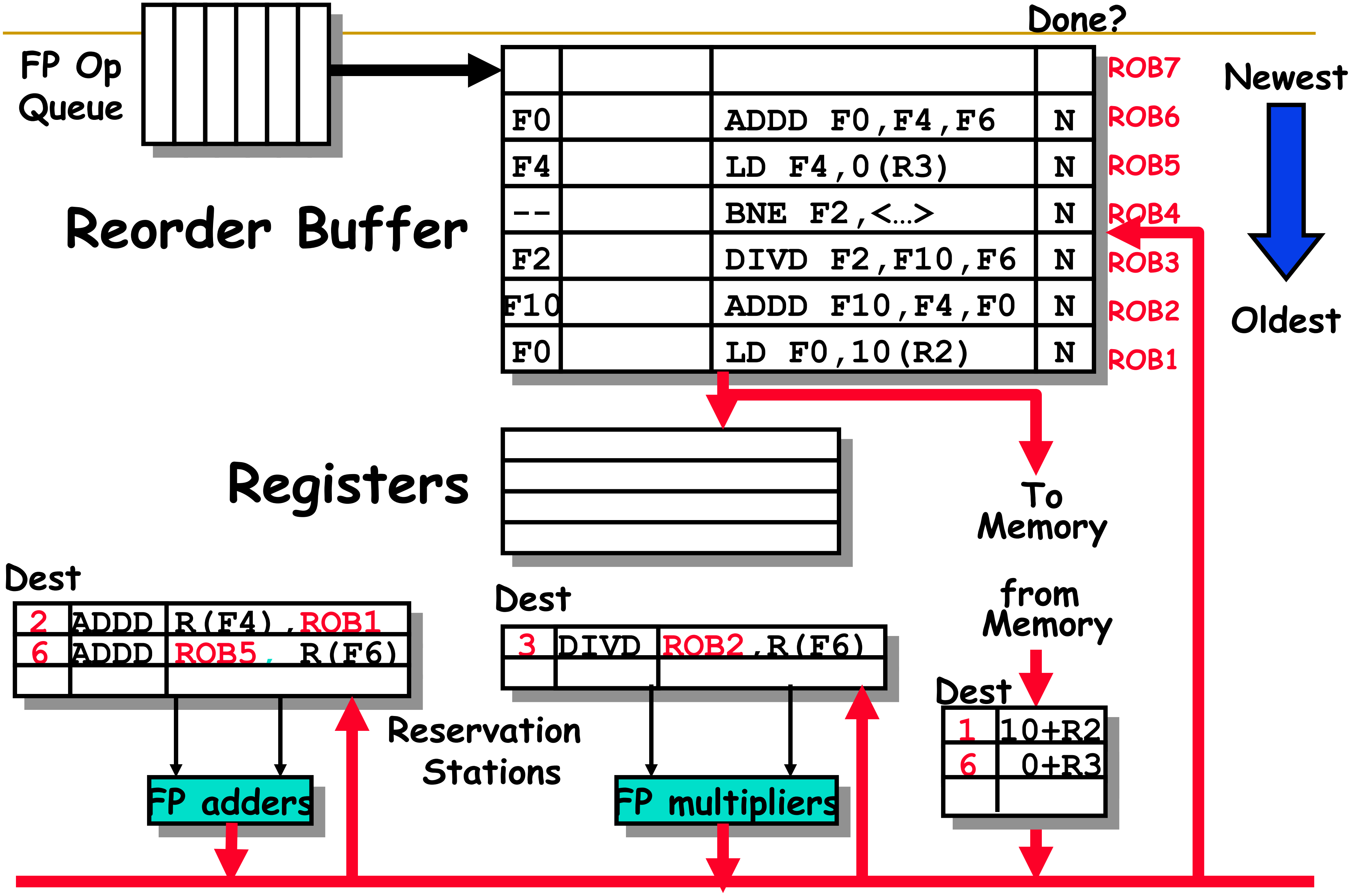
| | | | | |
|---|---|---|---|---|
| | | | | ROB7 |
| | | | | ROB6 |
| | | | | ROB5 |
| | | | | ROB4 |
| F2 | | DIVD F2,F10,F6 | N | ROB3 |
| F10 | | ADDD F10,F4,F0 | N | ROB2 |
| F0 | | LD F0,10(R2) | N | ROB1 |

| Dest | | |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
| | | |
| | | |

| Dest | | |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
| | | |

| Dest | |
|---|---|
| 1 | 10+R2 |
| | |
| | |

# Tomasulo With Reorder buffer:



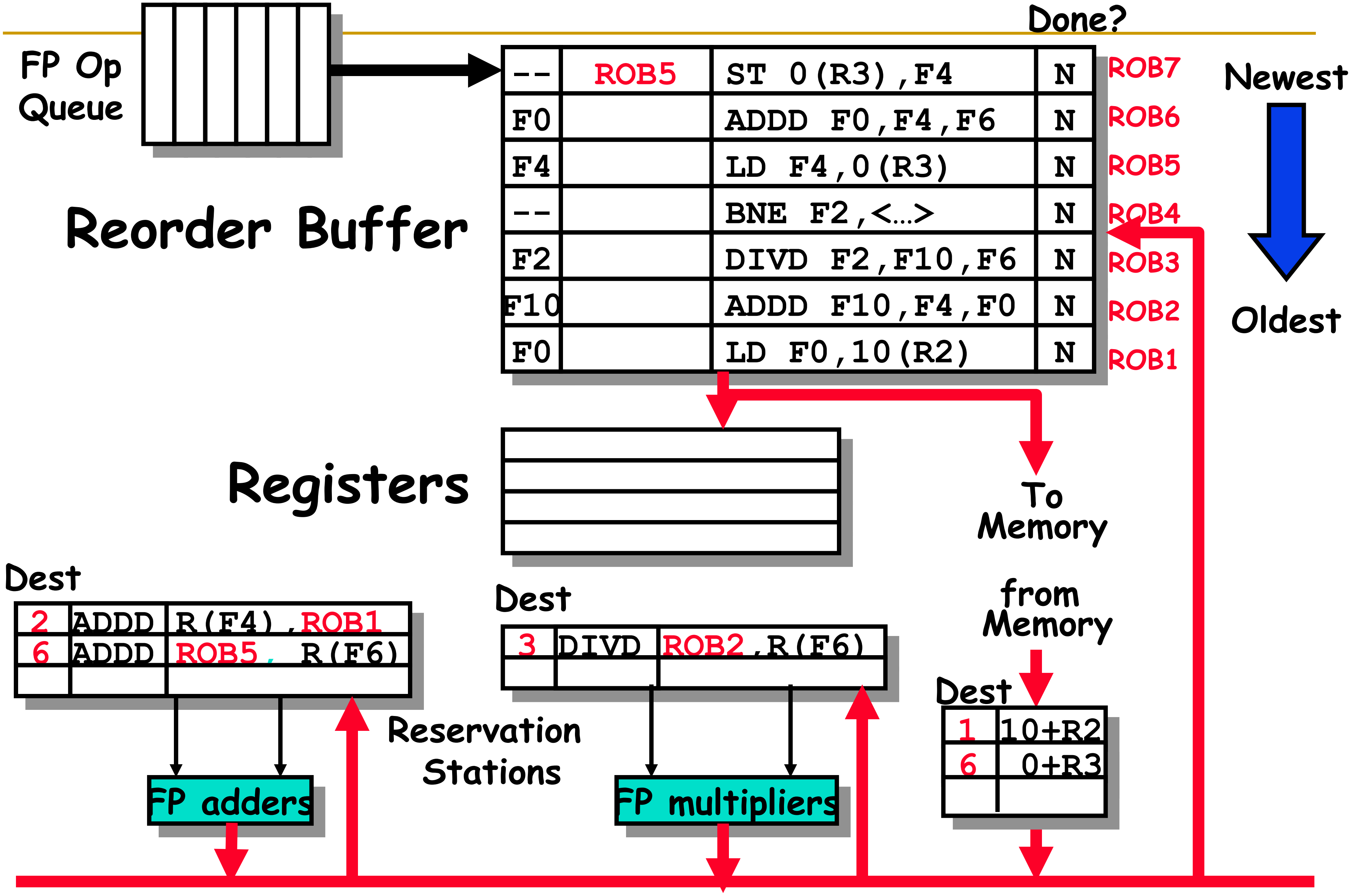
| | | | | Done? |
|---|---|---|---|---|
| ROB7 | | | | |
| ROB6 | F0 | | ADDD F0,F4,F6 | N |
| ROB5 | F4 | | LD F4,0(R3) | N |
| ROB4 | -- | | BNE F2,<…> | N |
| ROB3 | F2 | | DIVD F2,F10,F6 | N |
| ROB2 | F10 | | ADDD F10,F4,F0 | N |
| ROB1 | F0 | | LD F0,10(R2) | N |

| Dest | | |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
| 6 | ADDD | ROB5, R(F6) |
| | | |

| Dest | | |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
| | | |

| Dest | |
|---|---|
| 1 | 10+R2 |
| 6 | 0+R3 |
| | |

# Tomasulo With Reorder buffer:



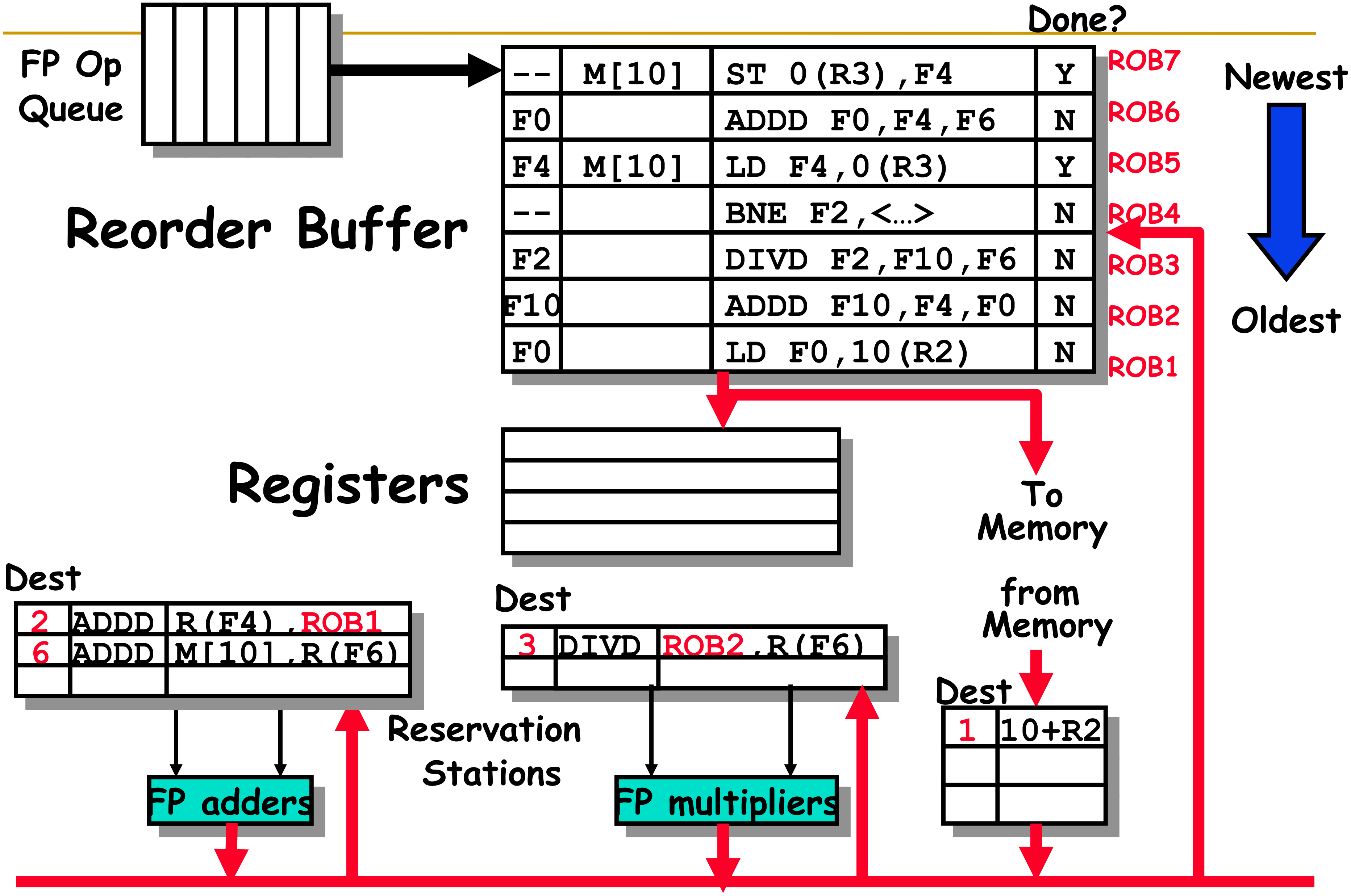
|  |  |  | Done? |  |
|---|---|---|---|---|
| -- | ROB5 | ST 0(R3),F4 | N | ROB7 |
| F0 |  | ADDD F0,F4,F6 | N | ROB6 |
| F4 |  | LD F4,0(R3) | N | ROB5 |
| -- |  | BNE F2,<…> | N | ROB4 |
| F2 |  | DIVD F2,F10,F6 | N | ROB3 |
| F10 |  | ADDD F10,F4,F0 | N | ROB2 |
| F0 |  | LD F0,10(R2) | N | ROB1 |

| Dest |  |  |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
| 6 | ADDD | ROB5, R(F6) |
|  |  |  |

| Dest |  |  |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
|  |  |  |

| Dest |  |
|---|---|
| 1 | 10+R2 |
| 6 | 0+R3 |
|  |  |

# Tomasulo With Reorder buffer:



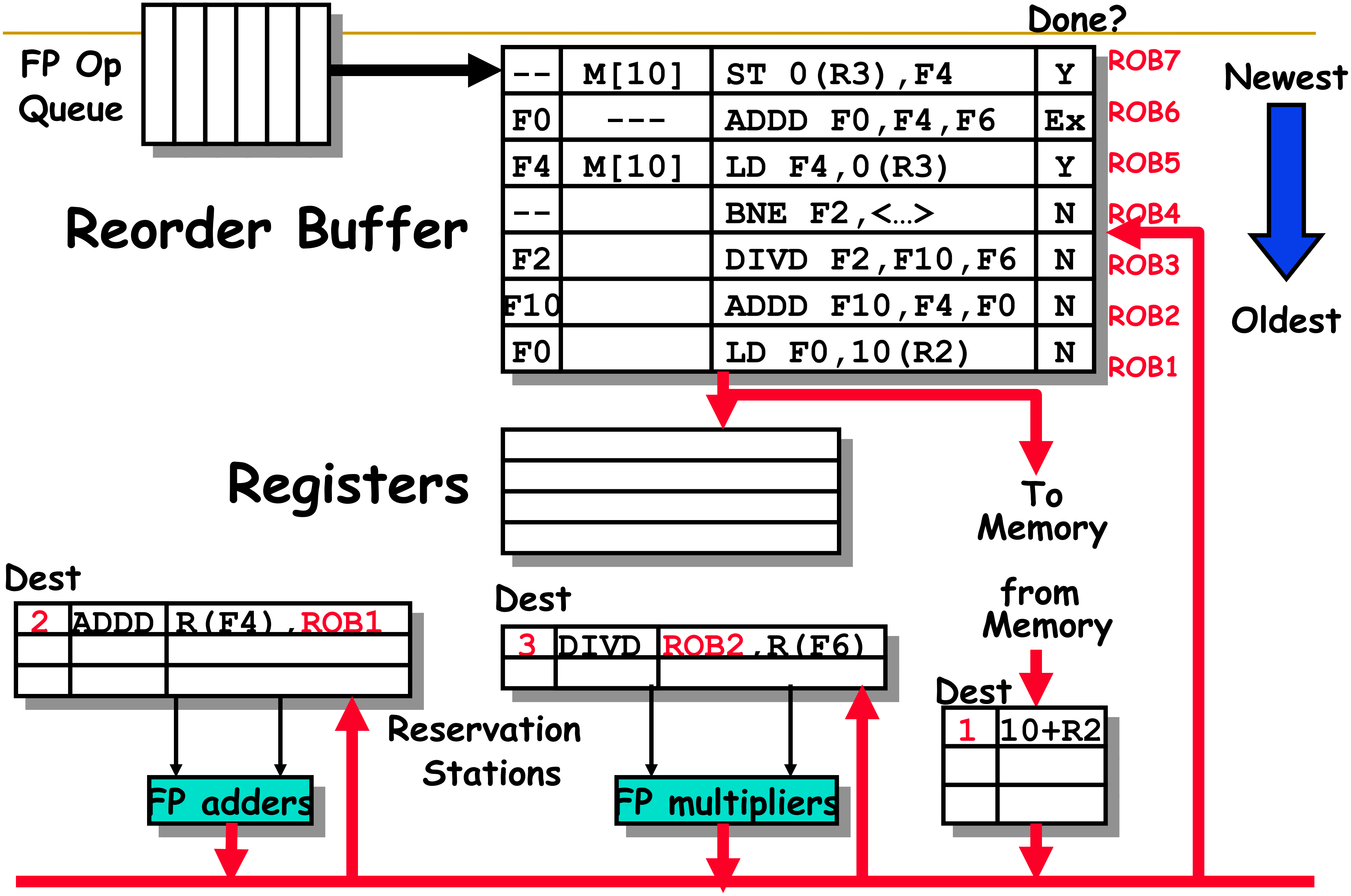
|  |  |  | Done? |  |
|---|---|---|---|---|
| -- | M[10] | ST 0(R3),F4 | Y | ROB7 |
| F0 |  | ADDD F0,F4,F6 | N | ROB6 |
| F4 | M[10] | LD F4,0(R3) | Y | ROB5 |
| -- |  | BNE F2,<…> | N | ROB4 |
| F2 |  | DIVD F2,F10,F6 | N | ROB3 |
| F10 |  | ADDD F10,F4,F0 | N | ROB2 |
| F0 |  | LD F0,10(R2) | N | ROB1 |

| Dest |  |  |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
| 6 | ADDD | M[10],R(F6) |
|  |  |  |

| Dest |  |  |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
|  |  |  |

| Dest |  |
|---|---|
| 1 | 10+R2 |
|  |  |
|  |  |

# Tomasulo With Reorder buffer:



| | | | Done? | |
|---|---|---|---|---|
| -- | M[10] | ST 0(R3),F4 | Y | ROB7 |
| F0 | --- | ADDD F0,F4,F6 | Ex | ROB6 |
| F4 | M[10] | LD F4,0(R3) | Y | ROB5 |
| -- | | BNE F2,<…> | N | ROB4 |
| F2 | | DIVD F2,F10,F6 | N | ROB3 |
| F10 | | ADDD F10,F4,F0 | N | ROB2 |
| F0 | | LD F0,10(R2) | N | ROB1 |

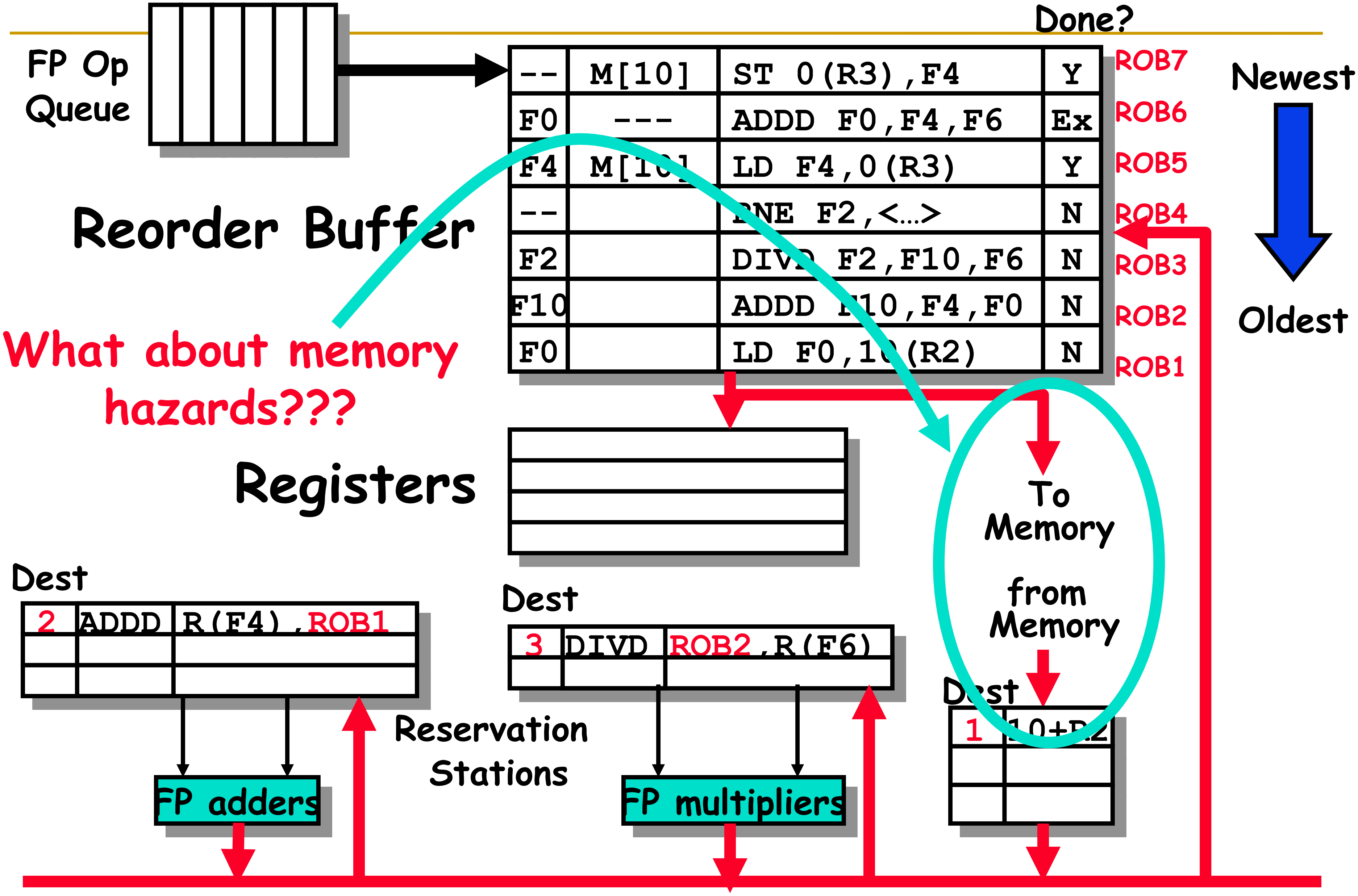
| Dest | | |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
| | | |
| | | |

| Dest | | |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
| | | |

| Dest | |
|---|---|
| 1 | 10+R2 |
| | |
| | |

אנימציה

# Tomasulo With Reorder buffer:



|  |  |  | Done? |  |
|---|---|---|---|---|
| -- | M[10] | ST 0(R3),F4 | Y | ROB7 |
| F0 | --- | ADDD F0,F4,F6 | Ex | ROB6 |
| F4 | M[10] | LD F4,0(R3) | Y | ROB5 |
| -- |  | BNE F2,<…> | N | ROB4 |
| F2 |  | DIVD F2,F10,F6 | N | ROB3 |
| F10 |  | ADDD F10,F4,F0 | N | ROB2 |
| F0 |  | LD F0,10(R2) | N | ROB1 |

What about memory hazards???

| Dest |  |  |
|---|---|---|
| 2 | ADDD | R(F4),ROB1 |
|  |  |  |
|  |  |  |

| Dest |  |  |
|---|---|---|
| 3 | DIVD | ROB2,R(F6) |
|  |  |  |

| Dest |  |
|---|---|
| 1 | 10+R2 |
|  |  |
|  |  |

אנימציה

# Memory Disambiguation: Handling RAW Hazards in memory

Question: Given a load that follows a store in program order, are the two related?

- (Alternatively: is there a RAW hazard between the store and the load)?

```
Eg:     st      0(R2),R5
        ld      R6,0(R3)
```

Can we go ahead and start the load early?

- Store address could be delayed for a long time by some calculation that leads to R2 (divide?).
- We might want to issue/begin execution of both operations in same cycle.

Two techiques:

- No Speculation: we are not allowed to start load until we know *for sure* that address 0(R2) ≠ 0(R3)
- Speculation: We might guess at whether or not they are dependent (called "dependence speculation") and use reorder buffer to fixup if we are wrong.

אנימציה

# Hardware Support for Memory Disambiguation

Need buffer to keep track of all outstanding stores to memory, in program order.

- Keep track of address (when becomes available) and value (when becomes available)
- FIFO ordering: will retire stores from this buffer in program order

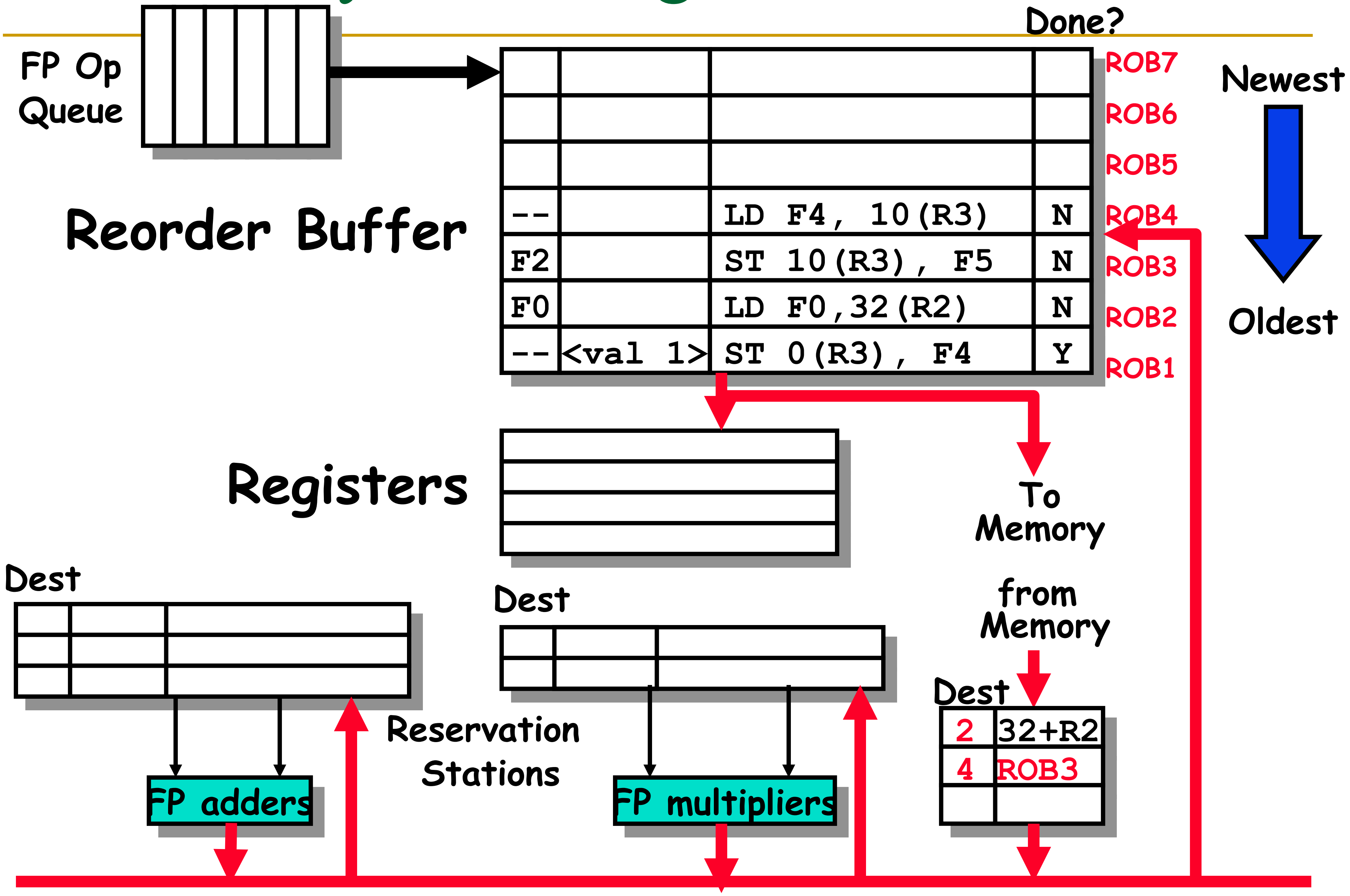
When issuing a load, record current head of store queue (know which stores are ahead of you).

When have address for load, check store queue:

- If *any* store prior to load is waiting for its address, stall load.
- If load address matches earlier store address (associative lookup), then we have a *memory-induced RAW hazard*:
  - store value available ⇒ return value
  - store value not available ⇒ return ROB number of source
- Otherwise, send out request to memory

Actual stores commit in order, so no worry about WAR/WAW hazards through memory.

# Memory Disambiguation:



FP Op Queue

Reorder Buffer

|  |  |  | Done? |  |
|---|---|---|---|---|
|  |  |  |  | ROB7 |
|  |  |  |  | ROB6 |
|  |  |  |  | ROB5 |
| -- |  | LD F4, 10(R3) | N | ROB4 |
| F2 |  | ST 10(R3), F5 | N | ROB3 |
| F0 |  | LD F0,32(R2) | N | ROB2 |
| -- | <val 1> | ST 0(R3), F4 | Y | ROB1 |

Newest

Oldest

To Memory

Registers

from Memory

| Dest |  |
|---|---|
| 2 | 32+R2 |
| 4 | ROB3 |
|  |  |

Dest

Dest

Reservation Stations

FP adders

FP multipliers

# עד כאן מצגת זו