Δυναμική Δρομολόγηση Εντολών (Dynamic Pipeline Scheduling)

Νεκτάριος Κοζύρης, Διονύσης Πνευματικάτος & Κωστής Νίκας

{nkoziris,pnevmati,knikas}@cslab.ece.ntua.gr

8ο εξάμηνο ΣΗΜΜΥ – Ακαδημαϊκό Έτος: 2025-26

https://helios.ntua.gr/course/view.php?id=1039

-Figures are from CA5e/Patterson-Hennessy (Elsevier)

CPI Improving techniques

Pipeline CPI =

Ideal pipeline CPI +

Structural Stalls +

Data Hazard Stalls +

Control Stalls

Superscalar execution

forwarding

register renaming

Dynamic scheduling

loop unrolling

static scheduling, software pipelining

Speculative execution

Branch prediction

delayed branches, branch scheduling

Data dependencies and Hazards

- J is data dependent on I:
  I must write r1 before J reads it, or, equivalently, J cannot read r1 before it is written by I

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- True Dependence
- Information flow (data values)
- Cause Read After Write (RAW) Hazards in the pipeline

Name Dependences, (1): Anti-dependences

- Name dependences: when 2 instructions use the same register or memory location ("name"), without exchanging information (values)
- Anti-dependence: J must write r1 before I reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Cause Write After Read (WAR) hazards in the pipeline

Name Dependences, (2): Output dependences

- Output dependence: J writes r1 before I write r1

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Cause Write After Write (WAW) hazards in the pipeline

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order

```
if p1 {
  S1;
};
if p2 {
  S2;
}
```

- `S1` is control dependent on `p1`, and `S2` is control dependent on `p2` but not on `p1`.

Control Dependence Ignored

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, *if can do so without affecting correctness of the program*
- Instead, 2 properties critical to program correctness are exception behavior and data flow

Exception Behavior

- Preserving exception behavior => any changes in instruction execution order must not change how exceptions are raised in program (=> no new exceptions)
- Example:

```
        DADDU    R2,R3,R4
        BEQZ     R2,L1    // if (p <> NULL)
        LW       R1,0(R2)
    L1:
```

- Problem with moving LW before BEQZ?

Data Dependencies & Hazards

- Dependencies are properties of the program
- The presence of a dependence indicates the possibility of hazard. Whether we will actually have a hazard, and how much stall will introduce is a function of the pipeline
- Significance of dependencies

  1) indicates the possibility of hazard

  2) Determine the expected order of computations

  3) Place an upper limit to the parallelism we can exploit

ILP και Data Hazards

- Due to dependencies we need to preserve "Program order"
- Program order: the execution sequence if the processor executes one instruction at a time according to the instruction sequence in the program
- Our goal (HW/SW): to exploit parallelism while preserving program order where needed, and avoid any changes when the change would impact the produced results of the program

Data Flow

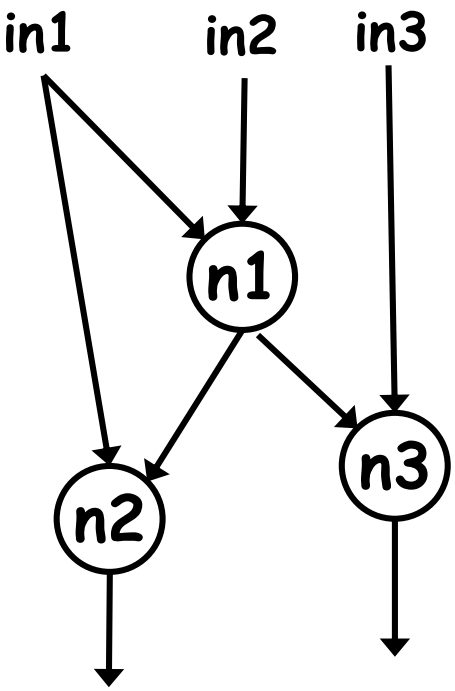
- **Data flow**: actual flow of data values among instructions that produce results and those that consume them
  - branches make flow dynamic, determine which instruction is supplier of data
- Example:

```
    DADDU   R1,R2,R3
    BEQZ    R4,L
    DSUBU   R1,R5,R6
L:  …
    OR      R7,R1,R8
```

- `OR` (**`R1`**) depends on `DADDU` or `DSUBU`?
  Must preserve data flow on execution

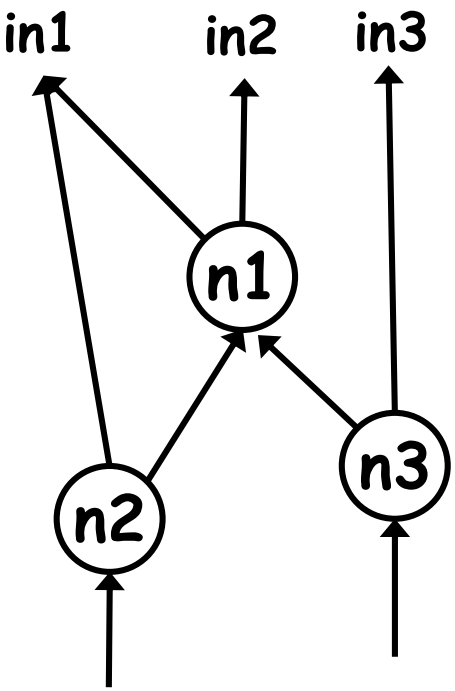
Let's talk about graphs

- Representation: describe *output to* nodes
  - in1 fan-out-to n1, n2
  - in2 fan-out-to n1
  - in3 fan-out-to n3
  - n1 fan-out-to n2, n3



Let's talk about graphs

- Representation: describe *output to* nodes
  - in1 fan-out-to n1, n2
  - in2 fan-out-to n1
  - in3 fan-out-to n3
  - n1 fan-out-to n2, n3



- OR: describe node *inputs*
  - Node n1(in1, in2)
  - Node n2(in1, n1)
  - Node n3(in3, n1)

- Both describe the same graph!
- Why care? Program in not a dependency graph!
  => We have to *build it* along the way!

Dynamic Instruction Scheduling (1)

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F12,F8,F14
```

- dependence μεταξύ DIVD και ADDD
- No dependence on SUBD. Why should it wait “behind” ADDD;

Equivalent code

```
DIVD   F0,F2,F4
SUBD   F12,F8,F14
ADDD   F10,F0,F8
```

- Dynamic Scheduling: change the instruction sequence (out-of-order execution) in HW during execution (runtime). We must:
  - Ensure correct flow of values
  - Ensure correct interrupt behavior

Dynamic Instruction Scheduling (2)

Why dynamic scheduling?

- Can handle cases where dependencies are not known at compile time (e.g., via memory locations)
- Less burden on the compiler
- Allows the efficient execution of programs in newer, different pipelines than the one the compiler optimized for.
- Allows for the speculative execution of instructions

Dynamic Instruction Scheduling (3)

- Properties:
  - *in-order instruction issue*
  - *out-of-order execution*
  - *out-of-order completion*

- The classic ID stage of the 5-stage pipeline is split in two "steps":
  - Issue: Decode the instruction(s) and check for structural hazards (in order issue)
  - Read Operands: Read the source operands when there are no data hazards (instructions wait here for the right moment to read their sources. Other instructions can bypass them and read operands earlier in ooo execution

Dynamic Instruction Scheduling Concerns

- Possible WAR και WAW hazards

```
1.DIVD   F0,F2,F4
2.ADDD   F6,F0,F8
3.SUBD   F8,F10,F14
4.MULD   F6,F10,F8
```

- antidependence: (2) -> (3)
  - If SUBD executes before ADDD we have a WAR
- output dependence: (2) -> (4)
  - If MULD executes before ADDD we have a WAW

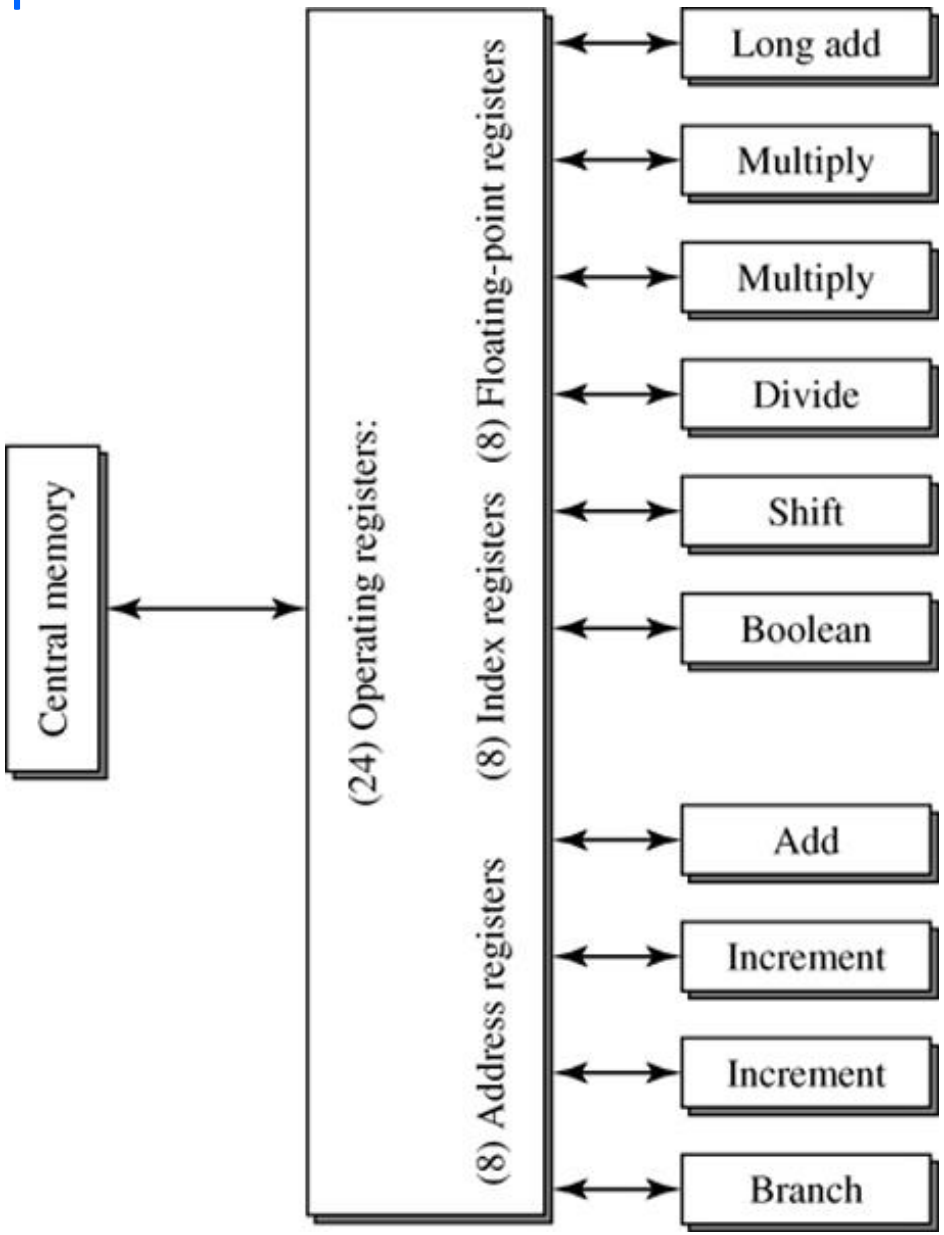
Dynamic Instruction Scheduling

- Scoreboard
  - CDC6600 [1963]
  - Control via the "scoreboard", a centralized structure
  - Deal with WAR hazards
    - » *Stall* the WB until registers are read
    - » Register reads happen at the Read Operands stage
  - Deal with WAW hazards
    - » Detect hazard and *stall* issue until earlier instruction is finished
- Tomasulo's algorithm
  - IBM360/91 [1966]
  - Deal with WAR and WAW hazards using *register renaming*
  - More effective technique

Scoreboard: CDC 6600-1964

- A super-computer class machine for the time!
- 1964 (before RISC processors and pipeline): featured 10 functional units, with different operation latency
- Goal: complete 1 instruction per cycle
- 8 address registers (18 bits)
- 8 index registers (18 bits)
- 8 fp registers (60 bits)

- Scoreboard technique: 1963



https://www.icsa.inf.ed.ac.uk/research/groups/hase/models/6600/index.html

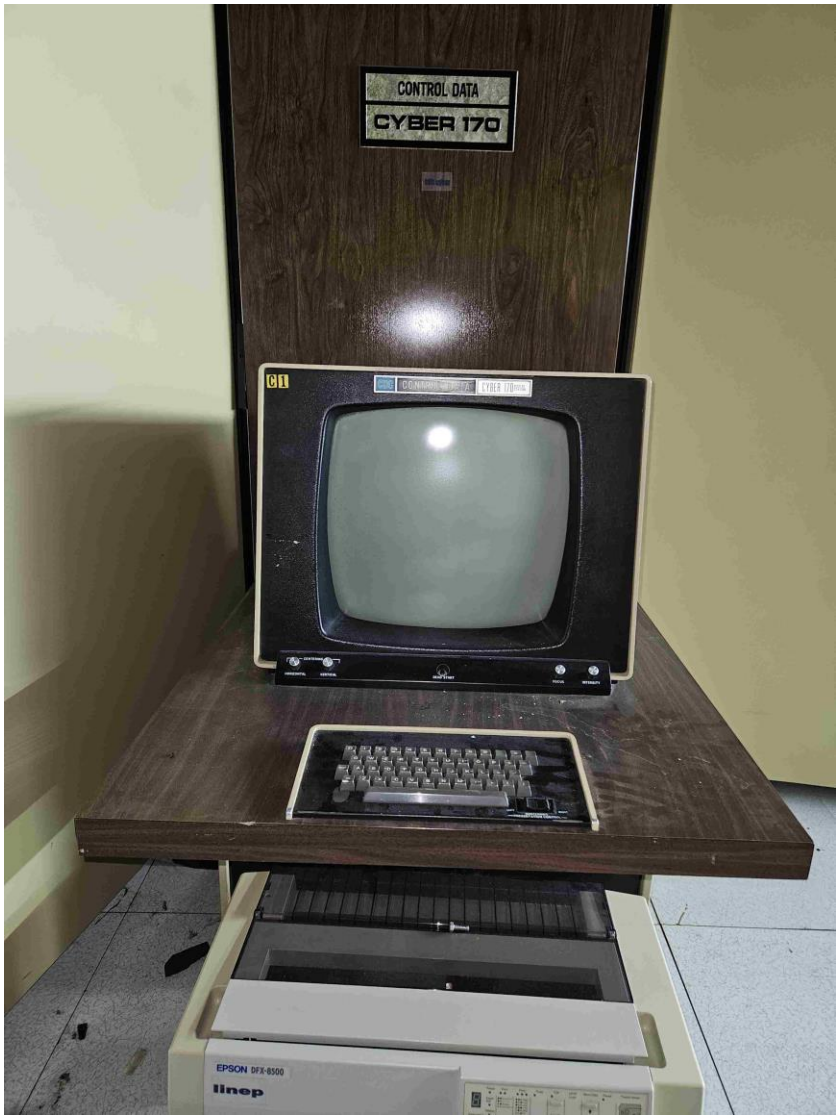
CDC 6600

- 10 functional units/non pipelined/variable exec latency
- 1x Fixed-point adder (18 bits)-3 cycles
- 1 x Floating-point adder (60 bits) – 4 cycles
- 2 x Multiply unit (60 bits)-10 cycles
- Divide unit (60 bits)-29 cycles
- Shift unit (60 bits)
- Logical unit (60 bits)
- 2 x increment units
- Branch unit

20

Cyber 170 (CDC 7600) @NTUA

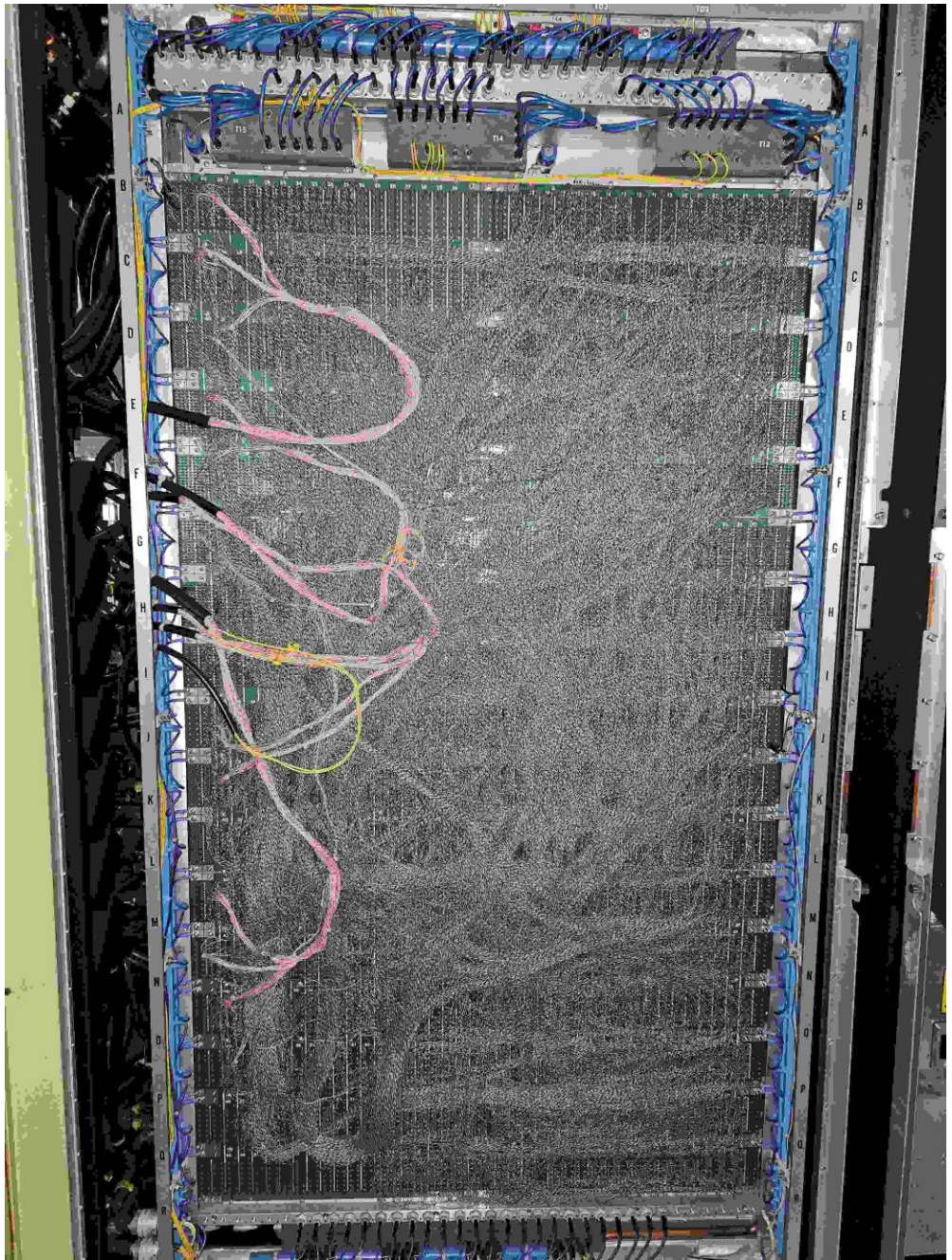
- Clock 36.4 MHz
- 60-bit word size
- 65 Kword magnetic core memory
- Up to 512 Kword secondary memory
- 10 MFLOPS on hand-compiled code
- Peak: 36 MFLOPS



National Technical University of Athens
CSLab

Cyber 170 (CDC 7600) @NTUA

Cyber 170 (CDC 7600) @NTUA



60459300

CONTROL DATA CORPORATION

NOS VERSION 2
SYSTEM MAINTENANCE
REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
CYBER 170
CYBER 70
MODELS 71, 72, 73, 74
6000

National Technical University of Athens CSLab

Dynamic Scheduling Step 1

- Simple (linear) pipeline uses one stage to check *both structural and data hazards*: Instruction Decode (ID) (or Instruction *Issue*)
- Split the ID pipe stage of simple 5-stage pipeline into 2 stages:
  - *Issue*—Decode instructions, check for structural hazards (and WAW hazards)
  - *Read operands*—Wait until no (RAW) data hazards, then read operands

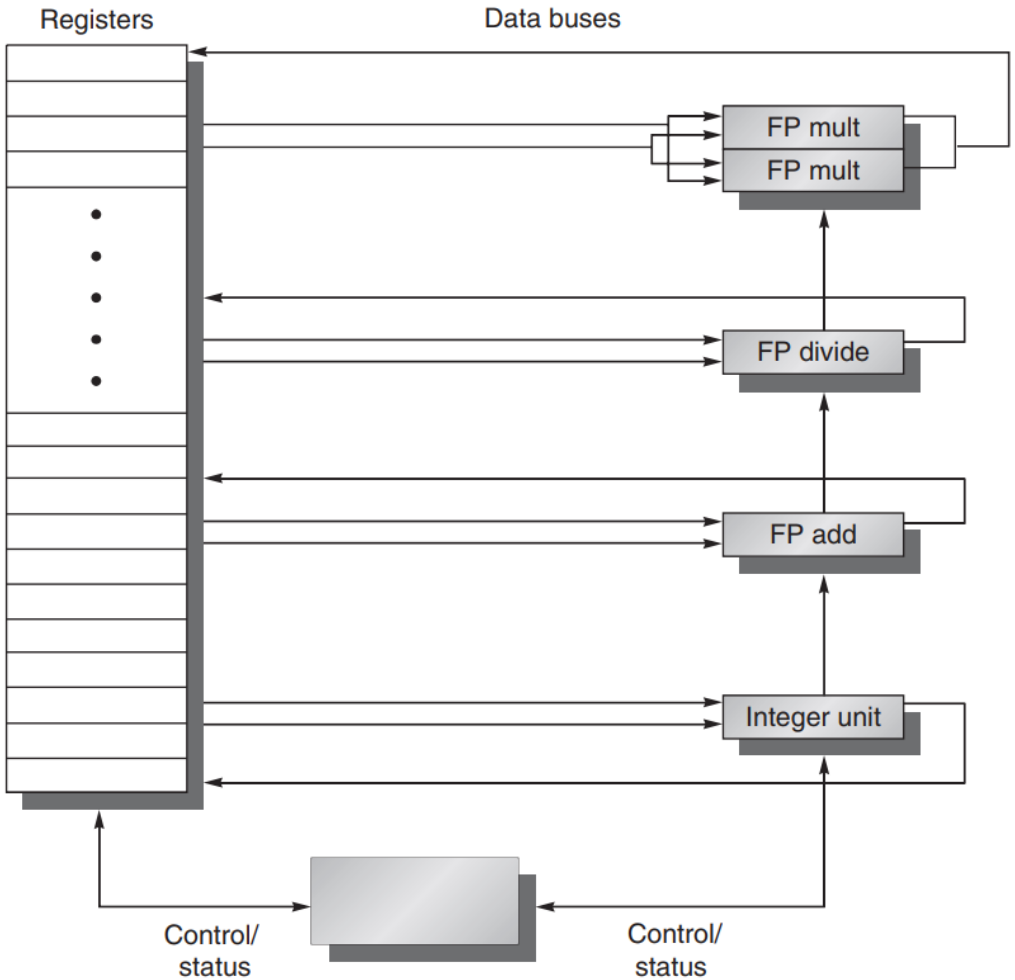


Figure C.54 The basic structure of a MIPS processor with a scoreboard. The scoreboard's function is to control instruction execution (vertical control lines). All of the data flow between the register file and the functional units over the buses (the horizontal lines, called *trunks* in the CDC 6600). There are two FP multipliers, an FP divider, an FP adder, and an integer unit. One set of buses (two inputs and one output) serves a group of functional units. The details of the scoreboard are shown in Figures C.55 to C.58.

-Figure from CA5e/Patterson-Hennessy (Elsevier)

1. *Instruction status*—Indicates which of the four steps the instruction is in.
2. *Functional unit status*—Indicates the state of the functional unit (FU). There are nine fields for each functional unit:
   - Busy—Indicates whether the unit is busy or not.
   - Op—Operation to perform in the unit (e.g., add or subtract).
   - Fi—Destination register.
   - Fj, Fk—Source-register numbers.
   - Qj, Qk—Functional units producing source registers Fj, Fk.
   - Rj, Rk—Flags indicating when Fj, Fk are ready and not yet read. Set to No after operands are read.
3. *Register result status*—Indicates which functional unit will write each register, if an active instruction has the register as its destination. This field is set to blank whenever there are no pending instructions that will write that register.

-Figure from CA5e/Patterson-Hennessy (Elsevier)

1. Issue
2. Read operands
3. Execute
4. Write result

Instruction status	Wait until	Bookkeeping
Issue	Not busy [FU] and not result [D]	Busy[FU]←yes; Op[FU]←op; Fi[FU]←D; Fj[FU]←S1; Fk[FU]←S2; Qj←Result[S1]; Qk← Result[S2]; Rj← not Qj; Rk← not Qk; Result[D]←FU;
Read operands	Rj and Rk	Rj← No; Rk← No; Qj←0; Qk←0
Execution complete	Functional unit done	
Write result	∀*f*((Fj[*f*] ¦ Fi[FU] or Rj[*f*] = No) & (Fk[*f*] ¦ Fi[FU] or Rk[*f*] = No))	∀*f*(if Qj[*f*]=FU then Rj[*f*]←Yes); ∀*f*(if Qk[*f*]=FU then Rk[*f*]←Yes); Result[Fi[FU]]← 0; Busy[FU]← No

Figure C.58 Required checks and bookkeeping actions for each step in instruction execution. FU stands for the functional unit used by the instruction, D is the destination register name, S1 and S2 are the source register names, and op is the operation to be done. To access the scoreboard entry named Fj for functional unit FU we use the notation Fj[FU]. Result[D] is the name of the functional unit that will write register D. The test on the write result case prevents the write when there is a WAR hazard, which exists if another instruction has this instruction's destination (Fi[FU]) as a source (Fj[*f*] or Fk[*f*]) and if some other instruction has written the register (Rj = Yes or Rk = Yes). The variable *f* is used for any functional unit.

-Figure from CA5e/Patterson-Hennessy (Elsevier)

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

Functional unit status									
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status									
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Integer			Add	Divide			

-Figure from CA5e/Patterson-Hennessy (Elsevier)

RESERVATION CONTROL
(SCOREBOARD)

The need for reservation control logic in the 6600 Central Processor arises due to the parallel processing concept of the CPU. This capability necessitates an orderly means of utilizing the functional units, operating registers and memory circuitry, since it is possible that several instructions require the same functional unit, operating register, etc. The scoreboard , then, makes the required reservations of each instruction and provides a means for the orderly handling of conflicts which may occur between instructions.

Conflicts are categorized into three groups - first, second and third order. The types of conflicts are defined as follows:

1) FIRST ORDER: A conflict between two instructions that require the same functional unit or the same result registers.

EXAMPLE 1: Functional Unit Conflict

F X 6 = X1 ⊕ X2
F X 5 = X3 ⊕ X4

Both instructions need the Floating Add functional unit for their calculation. Since only one such unit exists, the second instruction must wait until the first is finished, before it can be executed. Note that if two Multiply instructions are coded in sequence, no functional unit conflict occurs since two multiply units are provided.

EXAMPLE 2: Result Register Conflict

F X6 = X1 + X2
F X6 = X4 * X5

Both instructions require X6 for their result. In this case, the Floating Add result would be returned to X6 before the Multiply result was desired.

There are then, two types of First Order Conflicts - functional unit and result register. In all cases of first order conflicts, issuance of instructions stops until the conflict is resolved. In other words, no further instructions are initiated (including the one which "sees" the conflict) until the first of the conflicting instructions has completed. In conclusion, first order conflicts temporarily stop issuance of instructions at the point of conflict.

2) SECOND ORDER: A conflict that occurs when an instruction requires the result register of a previously initiated instruction as a source operand.

CONTROL DATA INSTITUTE CONTROL DATA CORPORATION

6600
CENTRAL PROCESSOR

Volume I
CONTROL & MEMORY

online from:
http://ygdes.com/CDC/020167_6600_CP_Vol1_Mar67.pdf

National Technical University of Athens CSLab

EXAMPLE:

F X6 = X1 + X2
F X7 = X5 / X6

In this case, the Divide unit needs X6, which is the result of the Add instruction, as one of its source operands. The Divide Unit must obviously wait for the Add unit to time out, but instruction issue will not stop. Instead, the Scoreboard will delay the start of the Divide instruction until the Add unit has stored its result. Subsequent instructions may be issued as long as no First Order conflicts exist.

The result of a Second Order Conflict is to delay the execution of the conflicting instruction only.

3) THIRD ORDER: A conflict that occurs when one instruction must store its result in a register which is to be used as a source operand for a previously issued instruction.

EXAMPLE:

F X3 = X1 / X2
F X5 = X4 * X3
F X4 = X0 + X6

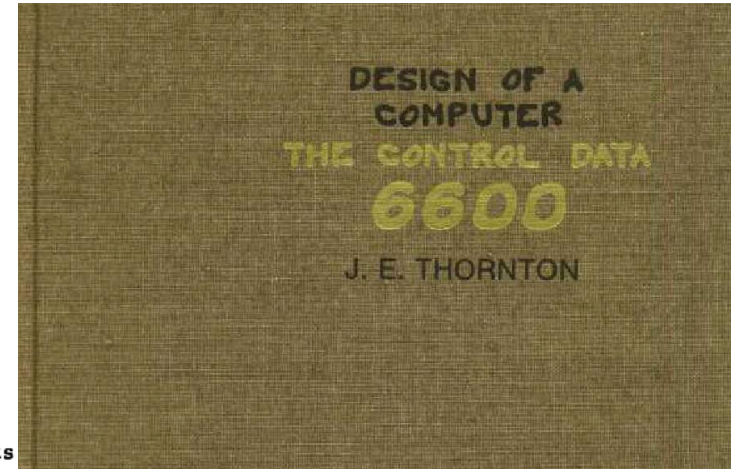
In this example, due to the relatively long execution times of the Divide and Multiply op. codes and the second order conflict (X3) of these units, the Add instruction will complete its calculation before the Multiply unit has read its operands (both operands are always read at the same time; therefore, all second order conflicts must be resolved). Since the Multiply instruction is intended to read X4 before it is changed by the Add instruction, storage of the Add result must be delayed until the Multiply Unit begins its calculation. Thus, third order conflicts do not delay issue or calculation, but rather the storage of a result operand.

The following discussion explains, at the block diagram level, how these conflicts are handled by the scoreboard. Figure 1-8 should be used in following the explanation.

FIRST-ORDER CONFLICTS

First-order conflicts are defined as either functional unit or result register conflicts. If either type exists, no issues can be generated.

Functional unit conflicts are determined by checking the Unit Request flip-flops against the Unit Busy flip-flops. Recall that Unit Requests are set at the



http://www.bitsavers.org/pdf/cdc/cyber/books/DesignOfAComputer_CDC6600.pdf

Scoreboard Structures

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

Functional unit status:

Time Name	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register result status:

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

Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D) **Structural + WAW**	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	**RAW** Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	**WAR** ∀f(not(Fj(f)=Fi(FU) and Rj(f)=Yes) & not(Fk(f)=Fi(FU) and Rk(f)=Yes))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rk(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No **Γιατί όχι RAW?**

(e.g. Add.d Fi, Fj, Fk)

Si = source registers, D destination register,

op = operation FU = Functional Unit for this instruction

Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f)≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rk(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

Equivalent!

(e.g. Add.d Fi, Fj, Fk)

Si = source registers, D destination register,

op = operation FU = Functional Unit for this instruction

Scoreboard Example: Cycle 1

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write 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				

Functional unit status:

Time Name	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
Integer	Yes	Load	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register result status:

Clock **1**

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
FU				Integer					

Scoreboard Example: Cycle 2

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*				Integer					

- **Issue 2nd LD?**

Scoreboard Example: Cycle 3

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	*FU*				Integer					

- **Issue MULT?**

Scoreboard Example: Cycle 4

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*				Integer					

Scoreboard Example: Cycle 5

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*		Integer							

Scoreboard Example: Cycle 6

Instruction status:

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

Functional unit status:

Time Name	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
Integer	~~Yes~~	~~Load~~	~~F2~~		~~R3~~				~~Yes~~
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	~~No~~								
Add	No								
Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
6	*FU*	Mult1	Integer							

Scoreboard Example: Cycle 7

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
7	*FU*	Mult1	Integer			Add				

- **Read multiply operands?**

Scoreboard Example: Cycle 8a
(First half of clock cycle)

Instruction status:

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

Functional unit status:

Time Name	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
Integer	Yes	Load	F2		R3				No
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
8	*FU*	Mult1	Integer			Add	Divide			

Scoreboard Example: Cycle 8b (Second half of clock cycle)

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
8	*FU*	Mult1				Add	Divide			

Scoreboard Example: Cycle 9

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
9	*FU*	Mult1				Add	Divide			

- **Issue ADDD? Read operands for MULT & SUB?**

Scoreboard Example: Cycle 9

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
→ ADDD	F6	F8	F2				

Functional unit status:

Time Remaining →

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
9	*FU*	Mult1				Add	Divide			

- ~~Issue ADDD?~~ Read operands for MULT & SUB? ✓

Scoreboard Example: Cycle 10

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
→ ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
9	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
10	*FU*	Mult1				Add	Divide			

Scoreboard Example: Cycle 11

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
→ ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
11	*FU*	Mult1				Add	Divide			

Scoreboard Example: Cycle 12

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
→ ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
12	*FU*	Mult1					Divide			

- **Read operands for DIVD? Issue ADDD?**

Scoreboard Example: Cycle 13

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
13	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 14

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
14	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 15

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
15	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 16

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
16	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 17

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

WAR Hazard!

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
17	*FU*	Mult1			Add		Divide			

- **Why not write result of ADD???**

Scoreboard Example: Cycle 18

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
18	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 19

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
19	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 20

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
20	*FU*				Add		Divide			

Scoreboard Example: Cycle 21

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
21	*FU*				Add		Divide			

- **WAR hazard is now gone...**

Scoreboard Example: Cycle 22

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
39	Divide	Yes	Div	F10	F0	F6			No	No

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
22	*FU*						Divide			

Faster than light computation
(skip a couple of cycles)

Scoreboard Example: Cycle 61

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			No	No

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
61	*FU*						Divide			

Scoreboard Example: Cycle 62

Instruction status:

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

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
62	*FU*									

Review: Scoreboard Example: Cycle 62

Instruction status:

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

Functional unit status:

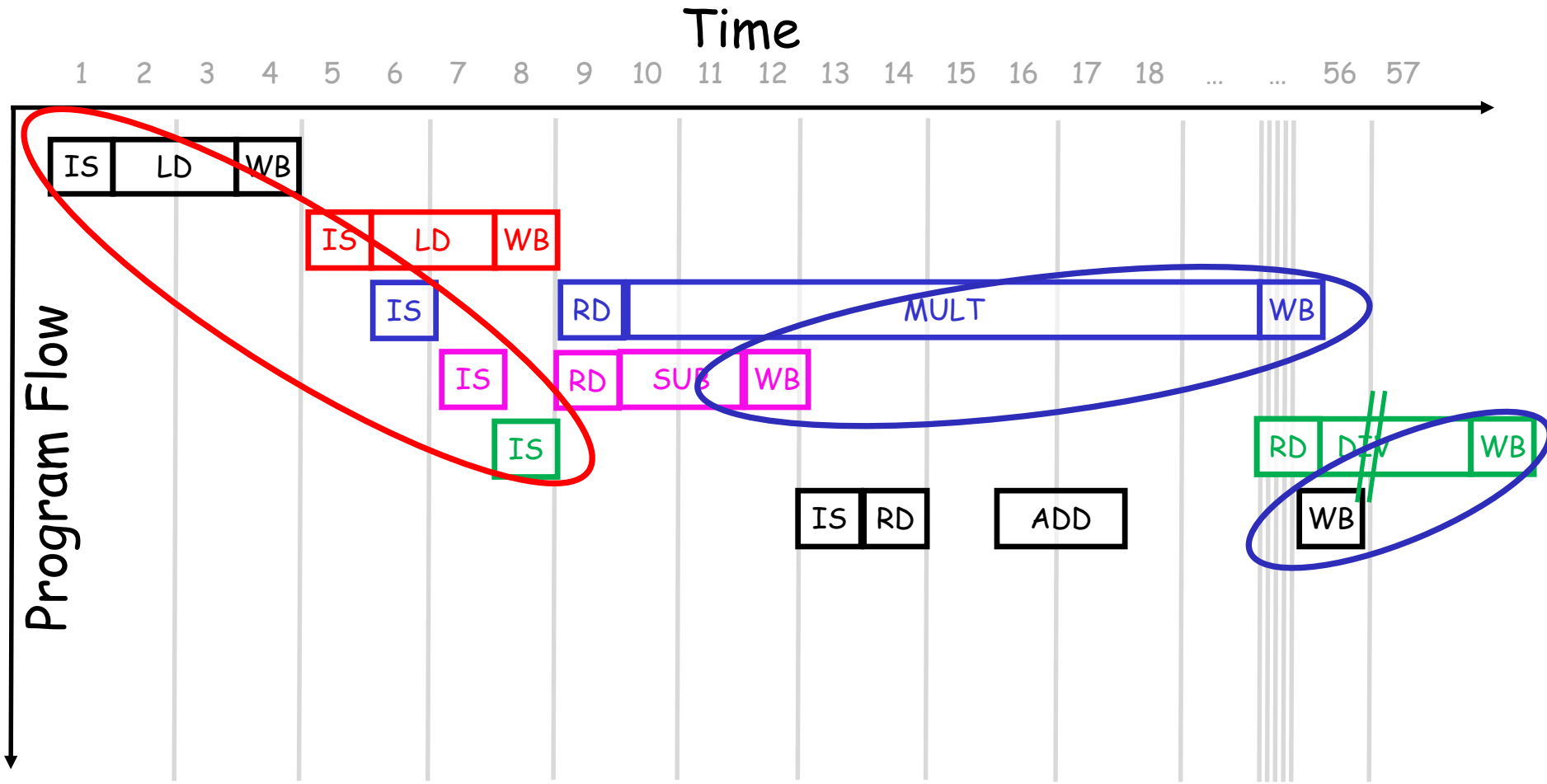
Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
62	*FU*									

- **In-order issue; out-of-order execute & commit**

Scoreboard Dynamic Execution



CDC 6600 Scoreboard

- Speedup 1.7 using the compiler (1964)
- Speedup 2.5 by hand (assembly) BUT slow memory (no cache) limits potential benefits
- Limitations of 6600 scoreboard:
  - Many read ports in the Register File!!!!
  - No forwarding hardware
  - Limited to instructions in basic block (small *window*)
  - Small number of functional units (structural hazards), especially integer/load store units (or added cost!!!)
  - Do not issue on structural hazards
  - Wait for WAR hazards
  - Stall instruction issue on WAW hazards

Avoiding WAR and WAW hazards with Register Renaming

DIV.D F0,F2,F4

ADD.D F6,F0,F8

S.D F6,0(R1)

SUB.D F8,F10,F14

MUL.D F6,F10,F8

Avoiding WAR and WAW hazards with Register Renaming

```
DIV.D   F0,F2,F4
ADD.D F6,F0,F8
S.D     F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8
```

WAR

WAW

```
DIV.D   F0,F2,F4
ADD.D F6,F0,F8
S.D     F6,0(R1)
SUB.D T,F10,F14
MUL.D F6,F10,T
```

```
DIV.D   F0,F2,F4
ADD.D S,F0,F8
S.D     S,0(R1)
SUB.D T,F10,F14
MUL.D F6,F10,T
```

No WAR or WAW hazards

```
DIV.D.  F0,F2,F4
ADD.D S,F0,F8
S.D     S,0(R1)
```

```
SUB.D T,F10,F14
MUL.D F6,F10,T
```

IBM's reaction to scoreboard ☺

MEMORANDUM

August 28, 1963

Memorandum To: Messrs. A. L. Williams
T. V. Learson
H. W. Miller, Jr.
E. R. Piore
O. M. Scott
M. B. Smith
A. K. Watson

Last week CDC had a press conference during which they officially announced their 6600 system. I understand that in the laboratory developing this system there are only 34 people, "including the janitor." Of these, 14 are engineers and 4 are programmers, and only one person has a Ph. D., a relatively junior programmer. To the outsider, the laboratory appeared to be cost conscious, hard working and highly motivated.

Contrasting this modest effort with our own vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer. At Jenny Lake, I think top priority should be given to a discussion as to what we are doing wrong and how we should go about changing it immediately.

TJW, Jr:jmc

T. J. Watson, Jr.

cc: Mr. W. B. McWhirter

Αλγόριθμος Tomasulo

- Reservation Stations (RS)
  - Store the instructions waiting for execution and their operand *values*
  - Distributed along with the Functional Units (FUs)
- The source registers of each instruction are replaced with the name of the RS that will produce the required input
  ⇒ register renaming
  - Avoids WAR, WAW hazards
  - Can have more RS (names) than actual registers ensuring better results than the compiler would achieve
- "forwarding" of results from RS to FUs, not via the register file, but via the Common Data Bus that broadcasts results to all FUs
- Memory Load & Stores are also treated as FUs with RS

IBM 360/91

- IBM wanted high floating-point performance from an instruction set and from compilers designed for the entire 360 computer family.

- IBM 360 architecture had only four double-precision floating-point registers.

- IBM 360/91 had long memory accesses and long floating-point delays.

Robert Tomasulo (IBM) invented the dynamic scheduling algorithm for OoO instruction execution, using register renaming (reservation stations) in 1966.

Tomasulo for MIPS floating point + load-store unit

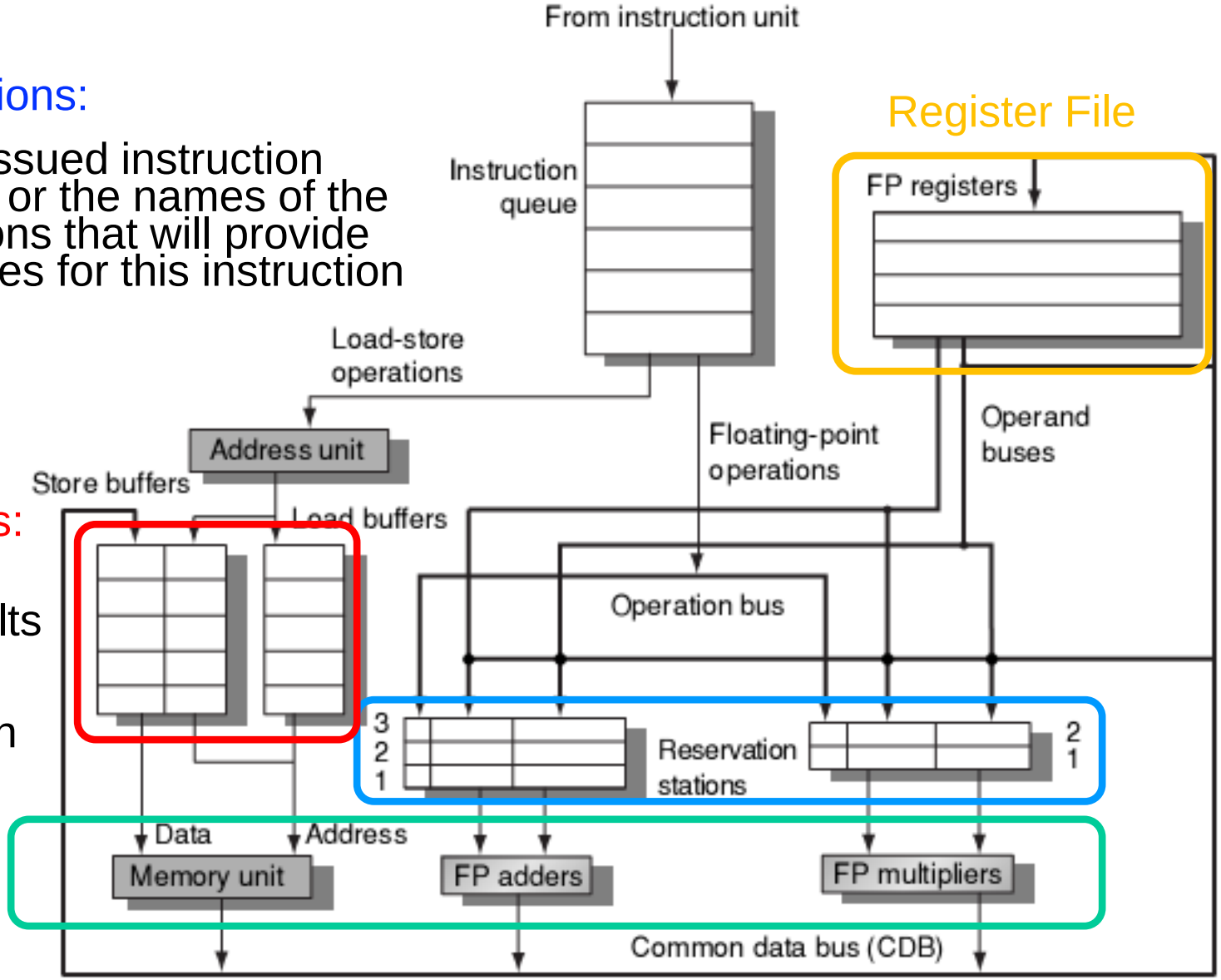
Reservation Stations:

contain already issued instruction and its operands or the names of the reservation stations that will provide the operand values for this instruction

Load/store buffers:

hold the effective address, the results of the completed loads, track loads that are waiting on the memory

Functional Units:



Common Data Bus

Tomasulo's algorithm steps

Issue: gets the next instruction from the instruction queue

If there is a free RS (no structural hazard), issue the instruction on this together with its operands to this RS. Destination register renaming happens here.

Execute: perform the computation in the FU (EX)

When both operands are available, perform the operation on them. If one of both operands are missing, monitor the CDB for the required value(s)

Write result: (WB)

Broadcast the result via the CDB to all RS that are waiting for it (this includes the destination register). Mark the RS as free

Tomasulo Structures (1)

- Reservation Station fields
  - Op: operation to perform (π.χ. +, -, *, …)
  - Vj, Vk: source operands *values*
  - Qj, Qk: which RS will produce the value of the source operands
    - » *At any given time either Q or V are valid for each operand*
  - Busy: is this RS assigned or not?

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

Tomasulo Structures (2)

- Register Result status
  - Qi : which RS (and corresponding instruction) will produce the value for register i

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

- Load/Store Buffer fields
  - A: the effective address of the memory location to read/write
  - Busy: is this buffer entry valid or free?

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Tomasulo Structures (3)

- Common Data Bus
  - Typical data bus: data + destination (“go to” bus)
  - CDB: data + source (“come from” bus) CDB.{source, value}
  - E.g. 64 bits of data + 4 bits #RS
  - If the CDB.source is equal to the Q field of an RS, write the CDB value to the corresponding V field of the RS
  - Broadcast: one “talks”, all “listen”

Instruction execution steps in Tomasulo

1. Issue — get instruction from FP Op Queue

   r = find free RS for this type of instructions (if no free RS => stall)

   RS[r].op=op; RS[r].busy=1;

   RS[r].Vj = RF[rs].V; RS[r].Qj = RF[rs].Q;

   RS[r].Vk = RF[rt].V RS[r].Qk = RF[rt].Q

   RF[rd].Q = r

2. Wait — collect “interesting” results from CDB until ready

   ∀r, if (r.Qj = CDB.Q and CDB.valid) { r.Vj = CDB.V; r,Qj = 0}

   ∀r, if (r.Qk = CDB.Q and CDB.valid) { r.Vk = CDB.V; r,Qk = 0}

   ∀RFe, if (RFe.Q = CDB.Q and CDB.valid) { RFe.Vk = CDB.V; r,Qk = 0}

2. Execute — when ready, operate on operands (EX)

   ∀r, if (r.Qj = 0 and r,Qk = 0) and FU is free, start operation

3. Write result — finish execution (WB)

   If CDB is free: {CBD.Q = r; CDB.V = value; CDB.valid = 1}

Instruction state	Wait until	Action or bookkeeping

Issue
FP operation
	Station r empty	`if (RegisterStat[rs].Qi≠0)`
		`  {RS[r].Qj ← RegisterStat[rs].Qi}`
		`else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0};`
		`if (RegisterStat[rt].Qi≠0)`
		`  {RS[r].Qk ← RegisterStat[rt].Qi`
		`else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`
		`RS[r].Busy ← yes; RegisterStat[rd].Q ← r;`

Load or store	Buffer r empty	`if (RegisterStat[rs].Qi≠0)`
		`  {RS[r].Qj ← RegisterStat[rs].Qi}`
		`else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0};`
		`RS[r].A ← imm; RS[r].Busy ← yes;`

Load only		`RegisterStat[rt].Qi ← r;`

Store only		`if (RegisterStat[rt].Qi≠0)`
		`  {RS[r].Qk ← RegisterStat[rs].Qi}`
		`  else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`

Instruction state	Wait until	Action or bookkeeping
Execute FP operation	(RS[r].Qj = 0) and (RS[r].Qk = 0)	Compute result: operands are in Vj and Vk
Load-store 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	Load step 1 complete	Read from Mem[RS[r].A]
Write Result FP operation or load	Execution complete at r & CDB available	∀x(if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj ← result;RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk ← result;RS[x].Qk ← 0}); RS[r].Busy ← no;
Store	Execution complete at r & RS[r].Qk = 0	Mem[RS[r].A] ← RS[r].Vk; RS[r].Busy ← no;

Tomasulo Example

Instruction stream

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:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS 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

0

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

Clock cycle counter

(load: 2 cycles, add: 2 cycles, mult: 10 cycles, divide 40 cycles)

Tomasulo Example Cycle 1

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write 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:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS 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

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write 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:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS 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					

Tomasulo Example Cycle 3

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:

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

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- When an instruction is issued in an RS, the names of the source registers are replaced (i.e. renamed) via the V or Q fields of the RS
- Load1 instruction is completed - who is waiting for its result?

Tomasulo Example Cycle 4

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)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

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

- Load2 instruction is completed - who is waiting for its result?

Tomasulo Example Cycle 5

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)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

- Count down for Add1, Mult1 (load: 1 cycle, add: 2 cycles, mult: 10 cycles, divide 40 cycles)

Tomasulo Example Cycle 6

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	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

- ADDD is issue despite its name dependency on F6

Tomasulo Example Cycle 7

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:

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

Register result status:

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

- Add1 (SUBD) instruction is completed - who is waiting for its result?

Tomasulo Example Cycle 8

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:

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

Register result status:

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

Tomasulo Example Cycle 9

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:

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

Register result status:

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

Tomasulo Example Cycle 10

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	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

- Add2 (ADDD) instruction is completed - who is waiting for its result?

Tomasulo Example Cycle 11

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					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

- ADDD writes its result

Tomasulo Example Cycle 12

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					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 13

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					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 14

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)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 15

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)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

- Mult1 (MULTD) instruction is completed - who is waiting for its result?

Tomasulo Example Cycle 16

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		
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					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

- ...only DIVD is outstanding (div: 40 cycles)

After a few cycles…

Tomasulo Example Cycle 55

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		
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					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

Tomasulo Example Cycle 56

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	
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					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

- Mult2 (DIVD) instruction is completed – who is waiting for its result?

Tomasulo Example Cycle 57

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:

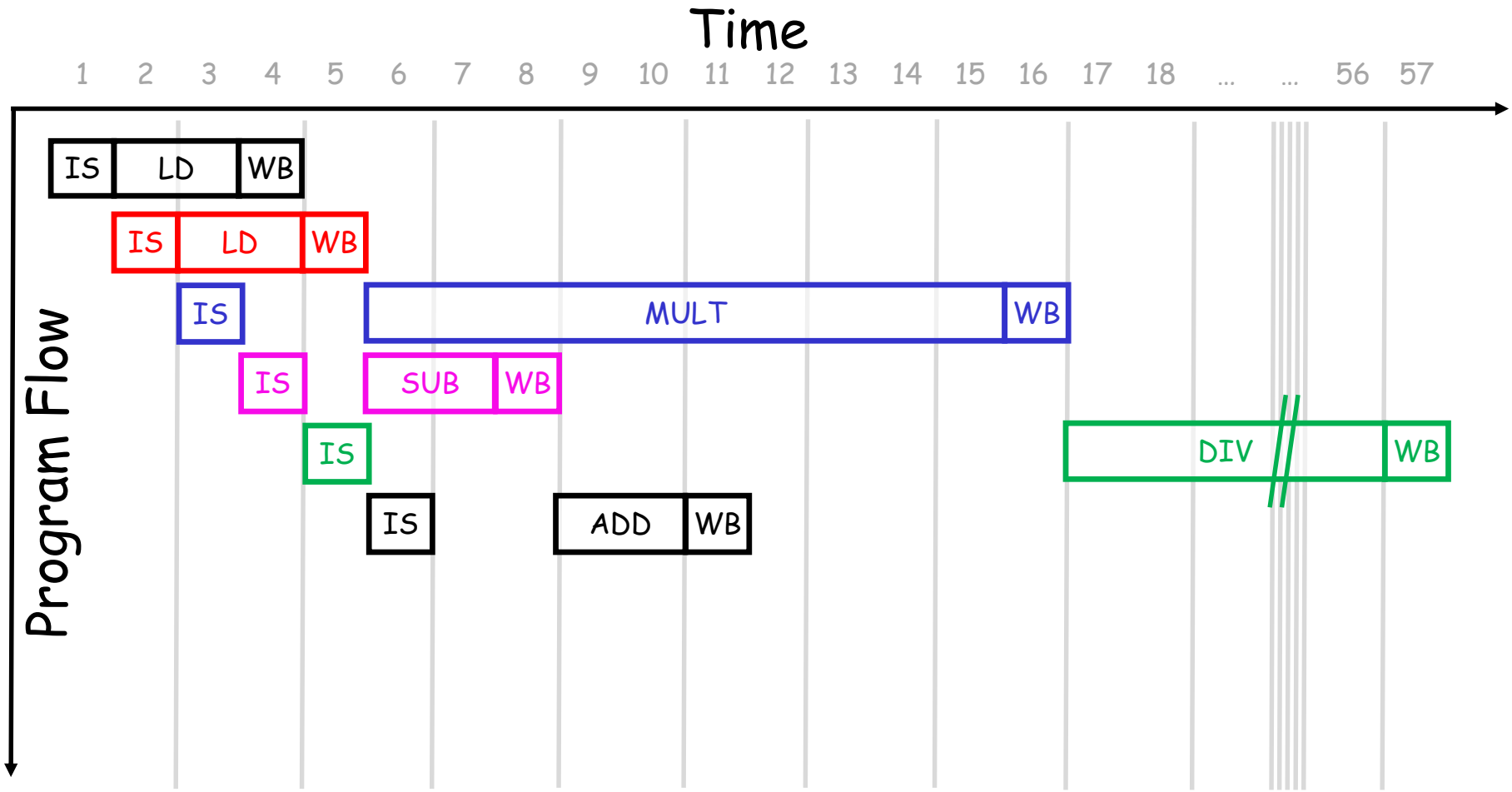
Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

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

Tomasulo Dynamic Execution



Tomasulo Loop Example

```
Loop: LD      F0    0     R1
      MULTD   F4    F0    F2
      SD      F4    0     R1
      SUBI    R1    R1    #8
      BNEZ    R1    Loop
```

- mult: 4 cycles
- 1st load: 8 cycles (L1 cache miss)
- 2nd load: 4 cycles (hit)
- branch is predicted as TAKEN

Loop Example

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1			
1	MULTD	F4	F0	F2			
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

Iter-ation Count

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Added Store Buffers

Reservation Stations:

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

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Instruction Loop

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
0	80	Fu									

Register value to be used for addresses and loop count

Loop Example Cycle 1

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

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

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
1	80	Fu	Load1								

Loop Example Cycle 2

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
2	80	Fu	Load1		Mult1						

Loop Example Cycle 3

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
3	80	Fu	Load1		Mult1						

Loop Example Cycle 4

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
4	80	Fu	Load1		Mult1						

- (the SUBI instruction –not in the FP queue- is dispatched)

Loop Example Cycle 5

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
5	72	Fu	Load1		Mult1						

- (the same for BNEZ)

Loop Example Cycle 6

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
6	72	Fu	Load2		Mult1						

- F0 never “sees” the value from address 80

Loop Example Cycle 7

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Fu	Load2		Mult2						

- The register file is decoupled from computations
- The 1st and 2nd loop iterations are overlapped

Loop Example Cycle 8

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
8	72	Fu	Load2		Mult2						

Loop Example Cycle 9

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
9	72	Fu	Load2		Mult2						

- Instruction in Load1 is competed – who is waiting for the result?
- (SUBI is dispatched)

Loop Example Cycle 10

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd	M[80]	R(F2)		
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
10	64	Fu	Load2		Mult2						

- The instruction in Load2 is finished – who waits for the result?
- (BNEZ is dispatch)

Loop Example Cycle 11

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	Multd	M[80]	R(F2)		
4	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
11	64	Fu	Load3		Mult2						

- next load in the sequence

Loop Example Cycle 12

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	Multd	M[80]	R(F2)		
3	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	**64**	*Fu*	Load3		Mult2						

- Why not issue the third mult as well?

Loop Example Cycle 13

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	Multd	M[80]	R(F2)		
2	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
13	64	Fu	Load3		Mult2						

- Why not issue the third store?

Loop Example Cycle 14

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	Multd	M[80]	R(F2)		
1	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		F0	F2	F4	F6	F8	F10	F12	...	F30
14	**64**	Fu	Load3		Mult2						

- The instruction in Mult1 is finished – who waits for the results?

Loop Example Cycle 15

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Fu	Load3		Mult2						

- The instruction in Mult2 is finished – who waits for the results?

Loop Example Cycle 16

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
16	64	Fu	Load3		Mult1						

- The 3rd MULTD is issued

Loop Example Cycle 17

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
17	64	Fu	Load3		Mult1						

- …can issue the 3rd SD

Loop Example Cycle 18

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
18	64	Fu	Load3		Mult1						

- …The 1st SD is completed

Loop Example Cycle 19

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
19	56	Fu	Load3		Mult1						

- …the 2nd SD is completed

Loop Example Cycle 20

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	20

	Busy	Addr	Fu
Load1	Yes	56	
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

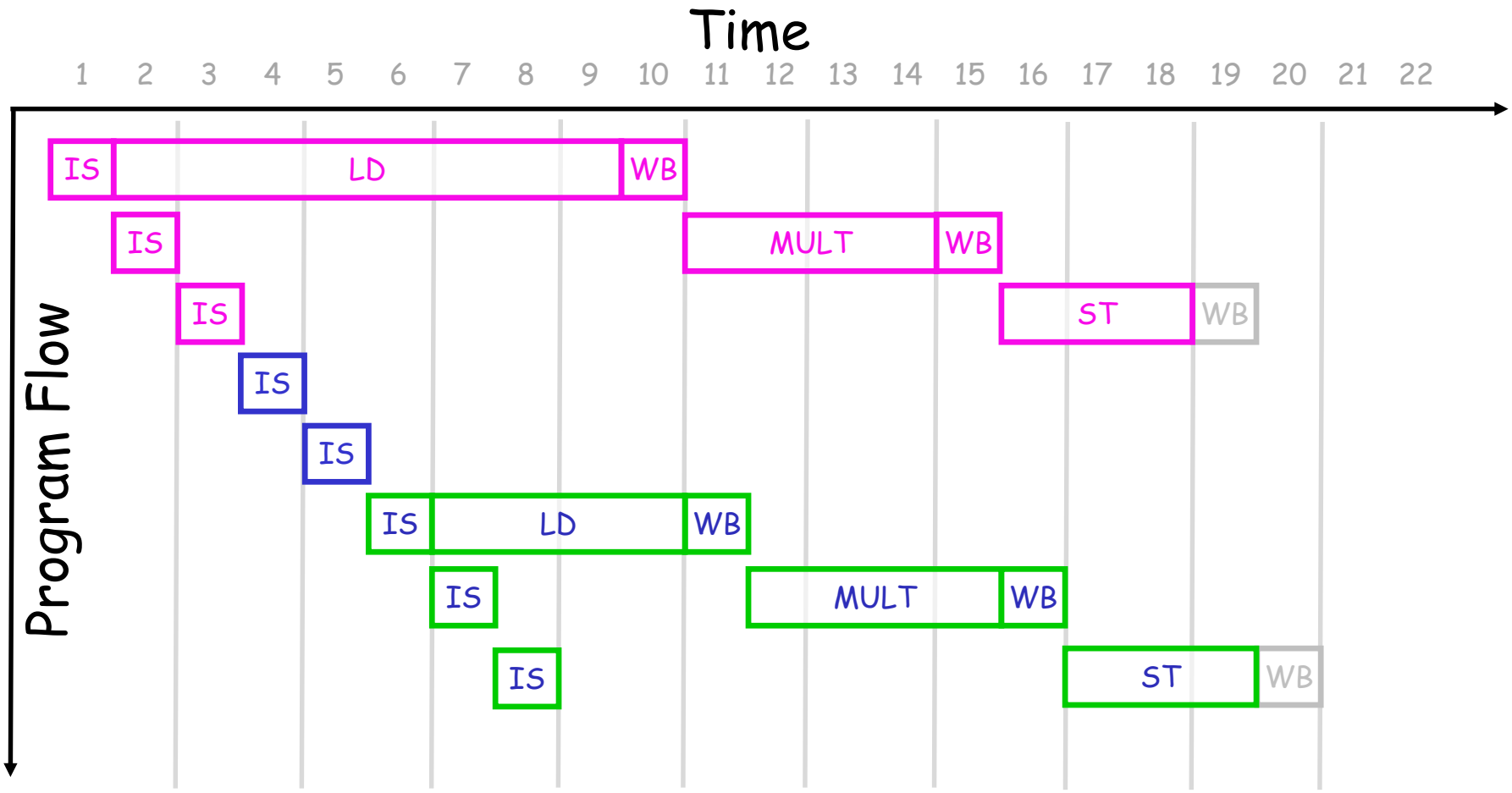
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Fu	Load1		Mult1						

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

Tomasulo Dynamic Execution



How does Tomasulo overlaps different iterations?

- Register renaming
  - Successive iterations use different “physical locations” for destination registers (dynamic loop unrolling)

- Reservation stations
  - Allows the instruction issue to proceed ahead of instruction execution (subject to true data and control dependencies)
  - Buffer old register values – avoids stalls due to WAR hazards

Tomasulo Basic Advantages

- Distributed logic for hazard detection
  - At the reservation station level
  - If multiple instructions are waiting for the same result (and the other operand is ready), they can be released for execution at the same time when the result they waiting for is broadcast on the CDB
    - » If we use a central register file, we need to read the RF (consuming register file read ports)
- Avoid stalls for WAW and WAR hazards
  - Register renaming: renamed “name” is the reservation station location

Tomasulo for MIPS floating point + load-store unit

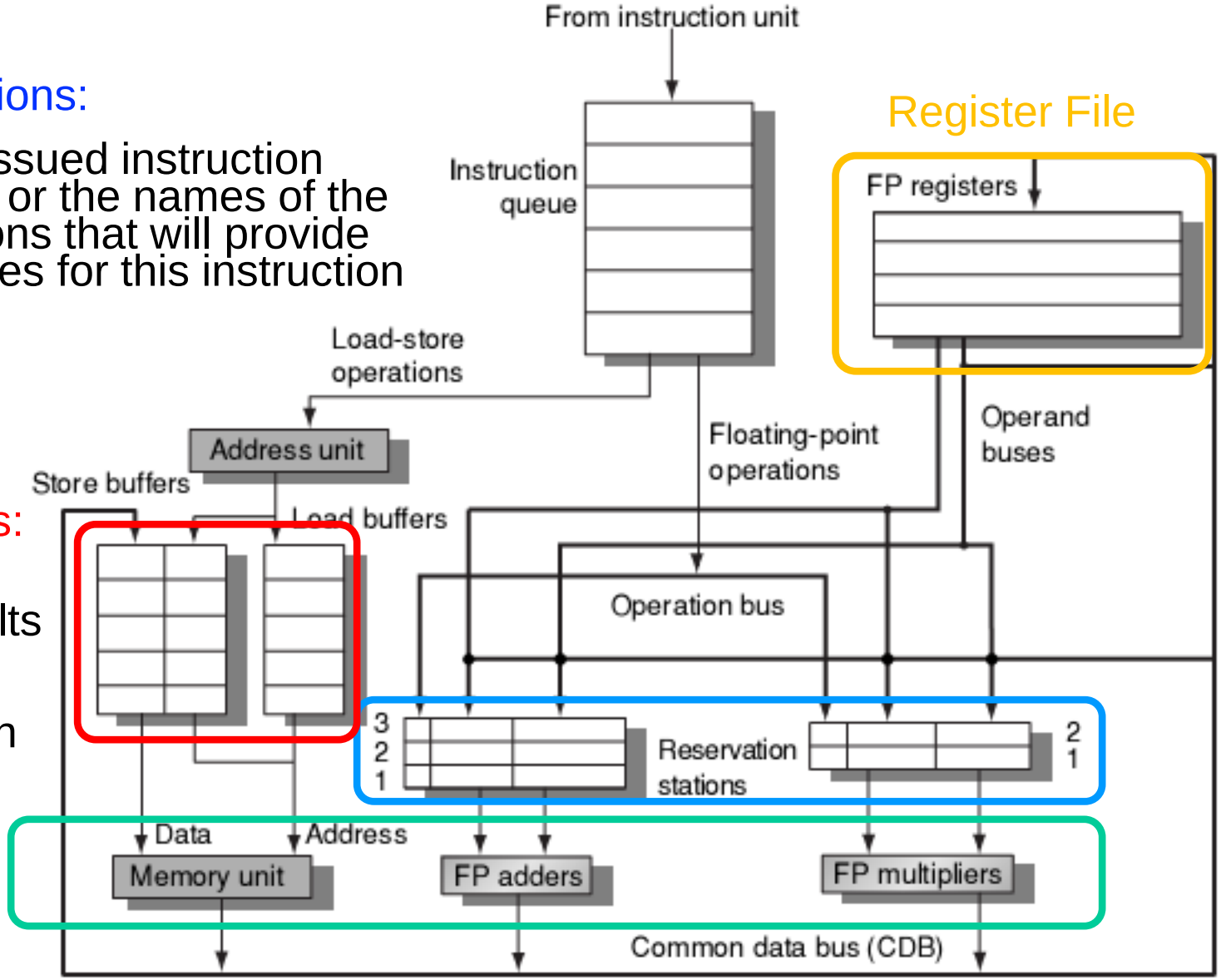
Reservation Stations:

contain already issued instruction and its operands or the names of the reservation stations that will provide the operand values for this instruction

Load/store buffers:

hold the effective address, the results of the completed loads, track loads that are waiting on the memory

Functional Units:



Common Data Bus

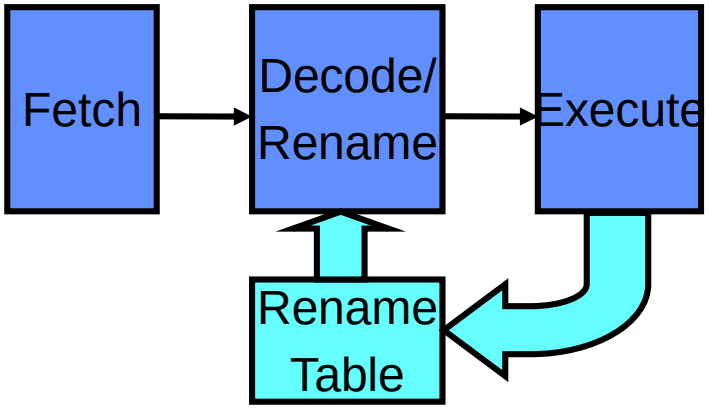
-Figure from CA5e/Patterson-Hennessy (Elsevier)

Explicit Register Renaming

- Idea : Why not have “infinite” registers or easier and more effective register renaming;
- We need:
  - A physical register file (PRF) with more “physical” registers than the ISA defines
  - (fast access) Translation Table
  - Concepts: logical register and physical location
  - Need mechanism to track free physical locations

Explicit Register Renaming(2)

- Assume a classic 5-stage pipeline



  - At decode each ISA register referenced in an instruction is mapped to a physical register
    - » target: one of the free registers is selected and the corresponding mapping is stored in the Register Map Table (RMT)
    - » source: The latest mapping is read from the RMT to be used for the source registers
  - Each physical register not used by any instruction is considered free

Example

Instruction Stream

```
DIV   R5,R4,R2
ADD   R7,R5,R1
SUB   R5,R3,R2
LD    R7,1000(R5)
```

Register Map Table

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
...	...

Free Registers

PR37,PR4,PR42,PR19,...

Example (1)

Instruction Stream

```
DIV    R5,R4,R2
ADD    R7,R5,R1
SUB    R5,R3,R2
LD     R7,1000(R5)
```

Register Map Table

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
...	...

Free Registers

PR37,PR4,PR42,PR19,...

```
DIV    PR37,PR45,PR2
```

Example (2)

Instruction Stream

```
DIV   R5,R4,R2
ADD   R7,R5,R1
SUB   R5,R3,R2
LD    R7,1000(R5)
```

Register Map Table

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	PR30
...	...

Free Registers

PR4,PR42,PR19,...

```
DIV   PR37,PR45,PR2
ADD   PR4  ,PR37,PR23
```

Example (3)

Instruction Stream

```
DIV   R5,R4,R2
ADD   R7,R5,R1
SUB   R5,R3,R2
LD    R7,1000(R5)
```

Register Map Table

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	PR4
...	...

Free Registers

PR42,PR19,...

```
DIV   PR37,PR45,PR2
ADD   PR4  ,PR37,PR23
SUB   PR42,PR17,PR2
```

Example (4)

Instruction Stream

```
DIV   R5,R4,R2
ADD   R7,R5,R1
SUB   R5,R3,R2
LD    R7,1000(R5)
```

Register Map Table

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR4
...	...

Free Registers

PR19,...

```
DIV   PR37,PR45,PR2
ADD   PR4 ,PR37,PR23
SUB   PR42,PR17,PR2
LD    PR19,1000(PR42)
```

Advantages

- No need for reservation stations
- Decouple register renaming from instruction scheduling
  - The pipeline can be simple (e.g. classic 5-stage pipeline) or complex
- Data transfers via the single physical register file
- Avoid WAR & WAW hazards
- Allows out-of-order completion (like Tomasulo)
- Many contemporary architectures use the explicit register renaming + OOO scheduling (similar to Tomasulo)
- Scoreboard + Renaming == Tomasulo ?