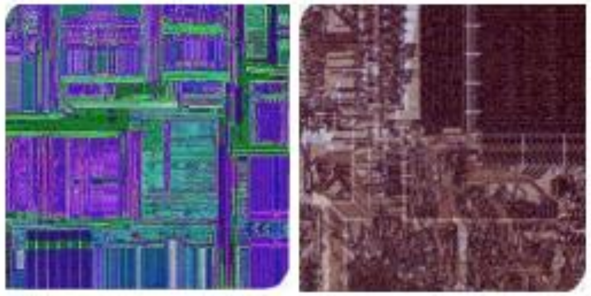


Dynamic Scheduling

Dynamic Scheduling

- simple pipelining techniques use in-order instruction issue and execution
- If an instruction is stalled in the pipeline no later instructions can proceed
- If there is a dependence between two closely spaced instructions in the pipeline, this will lead to a hazard and a stall will result
- In Dynamic scheduling hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behavior.

Dynamic Scheduling

- Advantages
  - Allows code compiled with one pipeline to run in a different pipeline – eliminates recompile for a different microarchitecture
  - Enables handling some cases when dependences are unknown at compile time
  - Allows the processor to tolerate unpredictable delays, such as cache misses, by executing other code while waiting for the miss to resolve

- dynamically scheduled processor cannot change the data flow, tries to avoid stalling when dependences are present

Dynamic Scheduling

- Example

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

- **`SUB.D`** instruction cannot execute because the dependence of **`ADD.D`** on **`DIV.D`** causes the pipeline to stall
- This hazard creates a performance limitation that can be eliminated by not requiring instructions to execute in program order.
- To allow execution of **`SUB.D`**, the issue process should be separated into two parts: checking for any structural hazards and waiting for the absence of a data hazard – instruction begins execution as soon as its data operands are available - out-of-order execution, which implies out-of-order completion.

Dynamic Scheduling

- Out-of-order execution introduces the possibility of WAR and WAW hazards

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8
SUB.D F8,F10,F14
MUL.D F6,F10,F8
```

- antidependence between the `ADD.D` and the `SUB.D`, and if the pipeline executes the `SUB.D` before the `ADD.D` - results a WAR hazard.
- Output dependence between `ADD.D` and `MUL.D` - results in WAW hazard
- Out-of-order execution should preserve the exception behavior of the program i.e., exceptions that arise if the program is executed in strict program order actually do arise

Dynamic Scheduling

- dynamically scheduled processors could generate imprecise exceptions
- An exception is ***imprecise*** if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order

- Two possibilities:
  1. The pipeline may have already completed instructions that are later in program order than the instruction causing the exception.
  2. The pipeline may have not yet completed some instructions that are earlier in program order than the instruction causing the exception.

Dynamic Scheduling

- To allow out-of-order execution, split the ID pipe stage of simple five-stage pipeline into two stages:
  1. *Issue*—Decode instructions, check for structural hazards.
  2. *Read operands*—Wait until no data hazards, then read operands.

IF	**ID**		EX	MEM	WB
IF	***Issue***	***Read operands***	EX	MEM	WB

- This pipeline allows multiple instructions in execution at the same time which requires the presence of multiple functional units.
- In a dynamically scheduled pipeline, all instructions pass through the issue stage in order (in-order issue); they can be stalled or bypass each other in the second stage (read operands) and thus enter execution out of order.

Tomasulo's Approach

- Tracks when operands for instructions are available to minimize RAW hazards
- Introduces register renaming in hardware to minimize WAW and WAR hazards

```
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
```

Antidependence
- **ADD.D** and **SUB.D**
- **S.D** and **MUL.D**

True data dependence
- **ADD.D** and **DIV.D**
- **S.D** and **ADD.D**
- **MUL.D** and **SUB.D**

Output dependence
- **ADD.D** and **MUL.D**

Tomasulo’s Approach

- register renaming – Assume the existence of two temporary registers, S and T

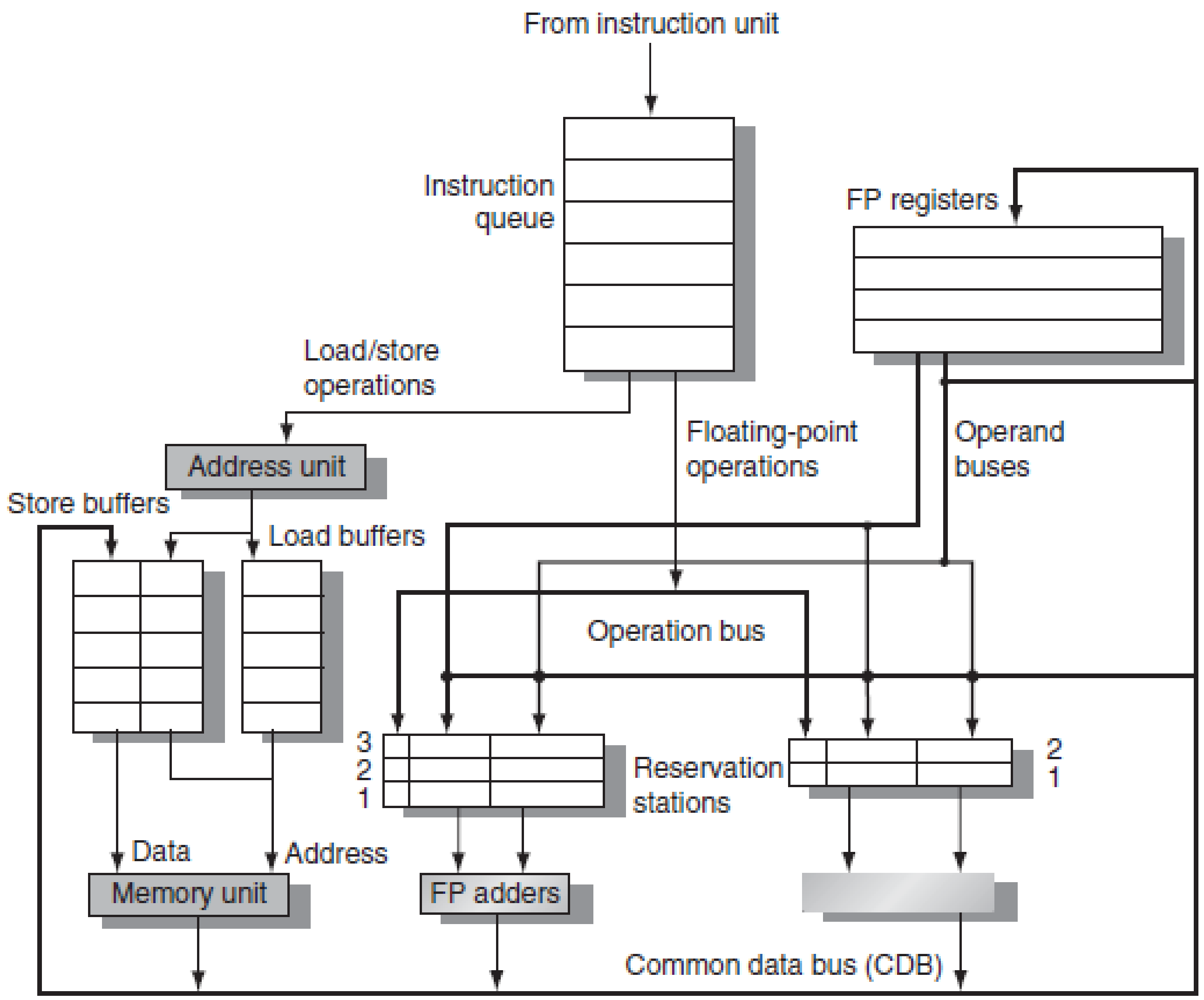
```
DIV.D  F0,F2,F4
ADD.D  S,F0,F8
S.D    S,0(R1)
SUB.D  T,F10,F14
MUL.D  F6,F10,T
```

- In Tomasulo’s scheme, register renaming is provided by reservation stations - buffers the operands of instructions waiting to issue

Tomasulo’s Approach

- Presence of reservation stations can eliminate the hazards arising from name dependences that could not be eliminated by a compiler - there can be more reservation stations than real registers
- Hazard detection and execution control are distributed - the information held in the reservation stations at each functional unit determines when an instruction can begin execution at that unit
- Results are passed directly to functional units from the reservation stations where they are buffered, rather than going through the registers
- This bypassing is done with a common result bus that allows all units waiting for an operand to be loaded simultaneously called the *common data bus (CDB)*

Tomasulo's Approach



Steps – an instruction goes through

- *Issue*
  - Get the next instruction from the head of instruction queue, in FIFO order to ensure the maintenance of correct data flow
  - If a matching reservation station is empty, issue the instruction to the station with the operand values, if they are currently in the registers.
  - If there is no empty reservation station, then there is a structural hazard and the instruction stalls until a station or buffer is freed.
  - If the operands are not in the registers, keep track of the functional units that will produce the operands. This step renames registers, eliminating WAR and WAW hazards.

Steps – an instruction goes through

- *Execute*
  - When an operand becomes available, it is placed into any reservation station awaiting it.
  - When all the operands are available, the operation can be executed at the corresponding functional unit.
  - By delaying instruction execution until the operands are available, RAW hazards are avoided.

Steps – an instruction goes through

- *Execute*
  - Loads and Stores require two-step execution process
  - The first step computes the effective address when the base register is available, and the effective address is then placed in the load or store buffer.
  - Loads in the load buffer execute as soon as the memory unit is available.
  - Stores in the store buffer wait for the value to be stored before being sent to the memory unit.
  - Loads and stores are maintained in program order through the effective address calculation, which will help to prevent hazards through memory.
  - To preserve exception behavior, no instruction is allowed to initiate execution until all branches that precede the instruction in program order has been completed execution.

Steps – an instruction goes through

- *Write Result*
  - When the result is available, write it on the CDB and from there into the registers and into any reservation stations including store buffers, waiting for this result.
  - Stores are buffered in the store buffer until both the value to be stored and the store address are available, then the result is written as soon as the memory unit is free.

Steps – an instruction goes through

- The data structures that detect and eliminate hazards are attached to the reservation stations, to the register file, and to the load and store buffers with slightly different information attached to different objects.
- The tag field describes which reservation station contains the instruction that will produce a result needed as a source operand.
- Because there are more reservation stations than actual register numbers, WAW and WAR hazards are eliminated by renaming results using reservation station numbers.
- The combination of the common result bus and the retrieval of results from the bus by the reservation stations implements the forwarding and bypassing mechanisms used in a statically scheduled pipeline.
- The tags in the Tomasulo scheme refer to the buffer or unit that will produce a result

Steps – an instruction goes through

- Each reservation station has seven fields:
  - Op—The operation to perform on source operands S1 and S2.
  - Vj, Vk—The value of the source operands.
  - Qj, Qk—The reservation stations that will produce the corresponding source operand; a value of zero indicates that the source operand is already available in Vj or Vk, or is unnecessary.
  - A—Used to hold information for the memory address calculation for a load or store. Initially, the immediate field of the instruction is stored here; after the address calculation, the effective address is stored here.
  - Busy—Indicates that this reservation station and its accompanying functional unit are occupied.
- The register file has a field, Qi:
  - Qi—The number of the reservation station that contains the operation whose result should be stored into this register.

Example

- Show what the information tables look like for the following code sequence when only the first load has completed and written its result:

```
L.D    F6,32(R2)
L.D    F2,44(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F2,F6
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Example

Instruction		Instruction status		
		Issue	Execute	Write result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

Register status

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Advantages of Tomasulo’s Scheme

- **Distribution of hazard detection logic**
  - Due to distributed reservation stations and use of the CDB.
  - If multiple instructions are waiting on a single result, and each instruction already has its other operand, then the instructions can be released simultaneously by broadcast of result on the CDB.
  - If a centralized register file were used, units have to read their results from registers when register buses are available.

- **Elimination of stalls for WAW and WAR hazards**
  - Done by renaming registers using the reservation stations and by the process of storing operands into the reservation station as soon as they are available

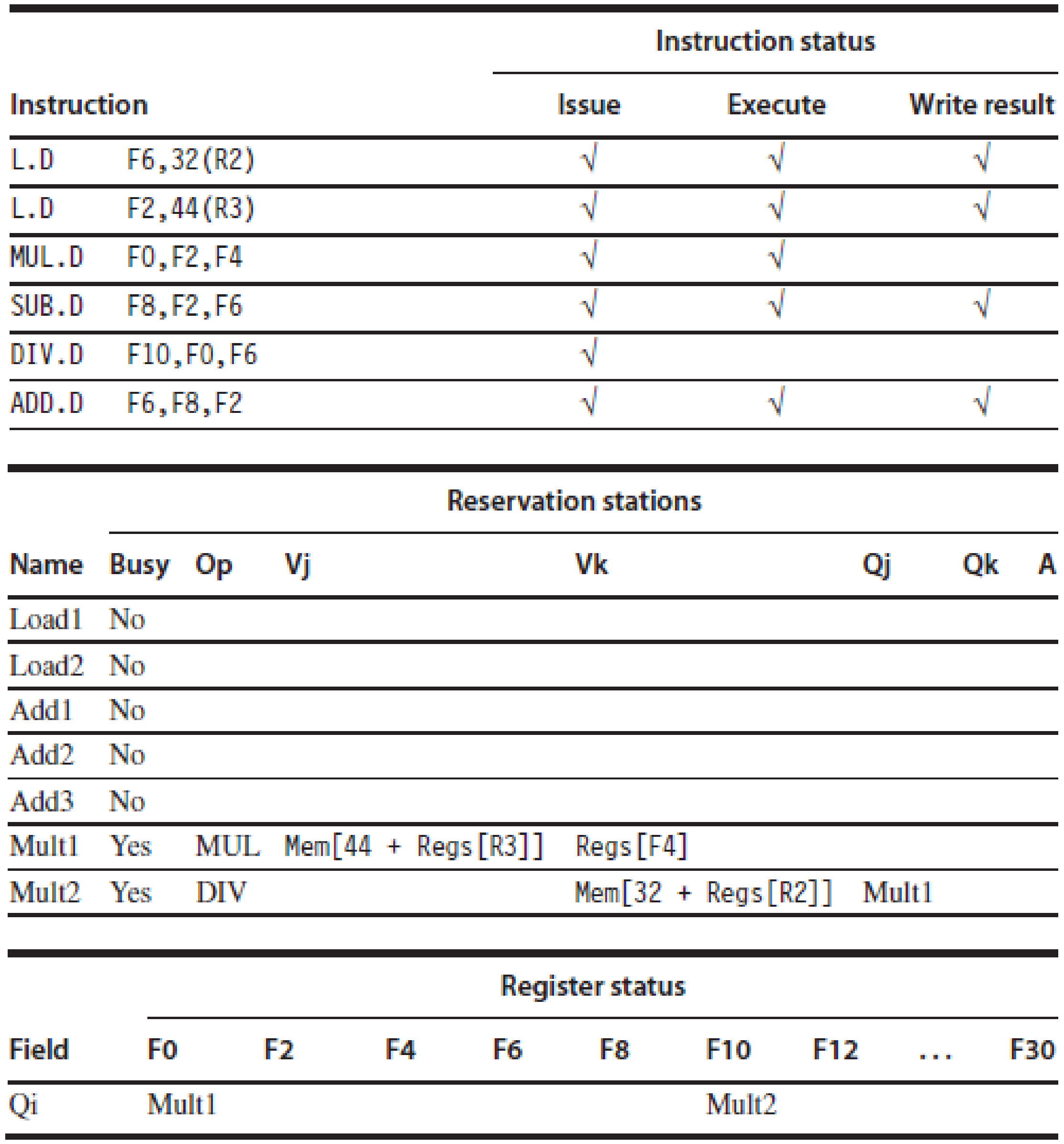
Example

- Assume the following latencies:
  - load is 1 clock cycle
  - add is 2 clock cycles
  - multiply is 6 clock cycles
  - divide is 12 clock cycles.

- Show what the status table look like when the MUL.D is ready to write its result.

```
L.D    F6,32(R2)
L.D    F2,44(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F2,F6
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Example



		Instruction status		
Instruction		Issue	Execute	Write result
`L.D`	`F6,32(R2)`	√	√	√
`L.D`	`F2,44(R3)`	√	√	√
`MUL.D`	`F0,F2,F4`	√	√	
`SUB.D`	`F8,F2,F6`	√	√	√
`DIV.D`	`F10,F0,F6`	√		
`ADD.D`	`F6,F8,F2`	√	√	√

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL	`Mem[44 + Regs[R3]]`	`Regs[F4]`			
Mult2	Yes	DIV		`Mem[32 + Regs[R2]]`	Mult1		

	Register status							
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Loop-Based Example

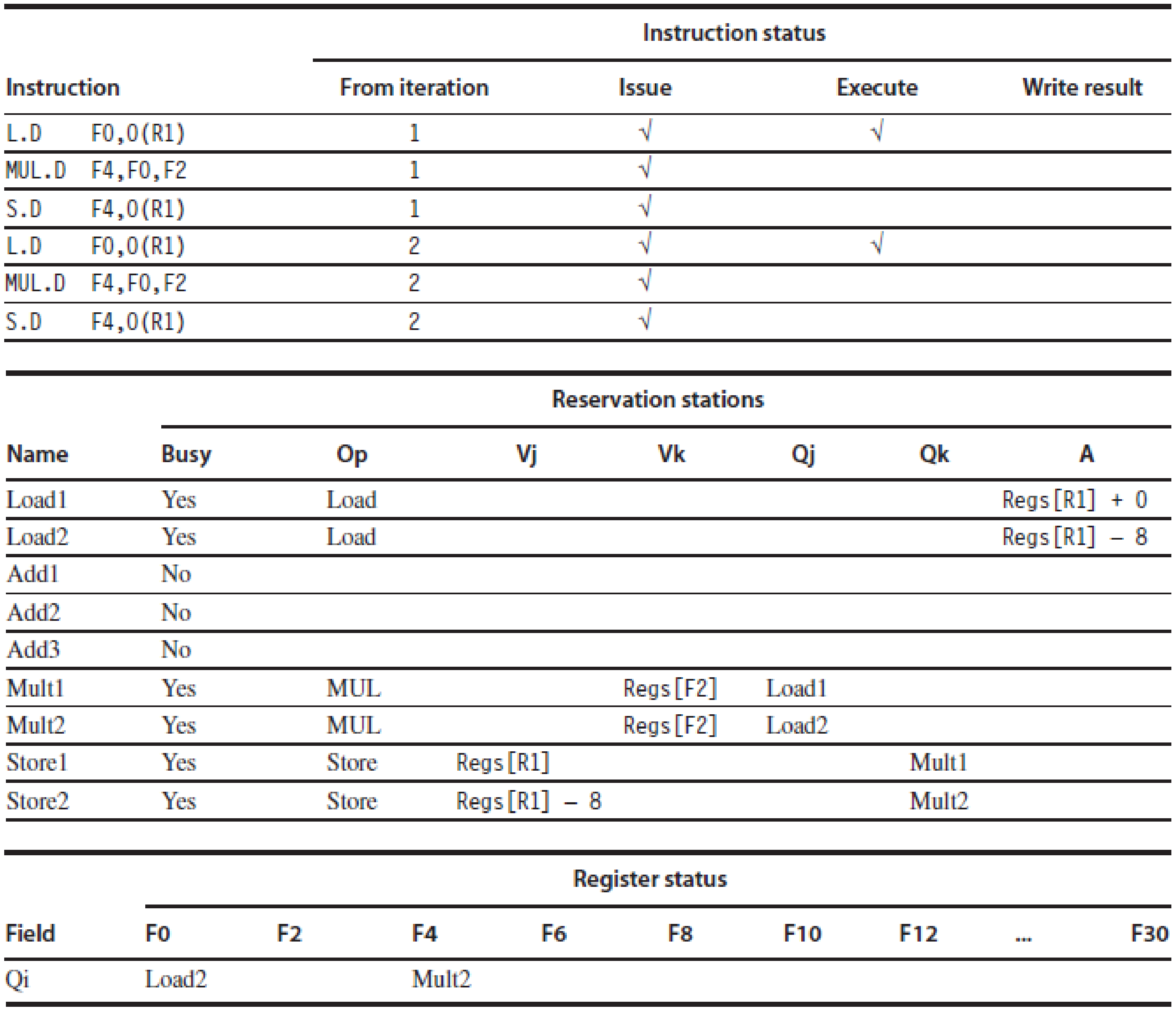
- Eliminating WAW and WAR hazards through dynamic renaming of registers

- Show what the status table look like when the loop is iterated twice with no instruction completed

```
Loop: L.D    F0,0(R1)
      MUL.D  F4,F0,F2
      S.D    F4,0(R1)
      DADDIU R1,R1,-8
      BNE    R1,R2,Loop;  branches if R1¦R2
```

- Assume the branch was predicted as taken.
- Loop unrolling is done dynamically using reservation stations.
- Consider all instructions in two successive iterations of loop but none of the loads have finished writing the result.

Loop-Based Example



Instruction	Instruction status			
	From iteration	Issue	Execute	Write result
`L.D F0,0(R1)`	1	√	√	
`MUL.D F4,F0,F2`	1	√		
`S.D F4,0(R1)`	1	√		
`L.D F0,0(R1)`	2	√	√	
`MUL.D F4,F0,F2`	2	√		
`S.D F4,0(R1)`	2	√		

Name	Reservation stations						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	Yes	Load					Regs[R1] + 0
Load2	Yes	Load					Regs[R1] – 8
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL		Regs[F2]	Load1		
Mult2	Yes	MUL		Regs[F2]	Load2		
Store1	Yes	Store	Regs[R1]			Mult1	
Store2	Yes	Store	Regs[R1] – 8			Mult2	

Field	Register status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

Hardware Speculation