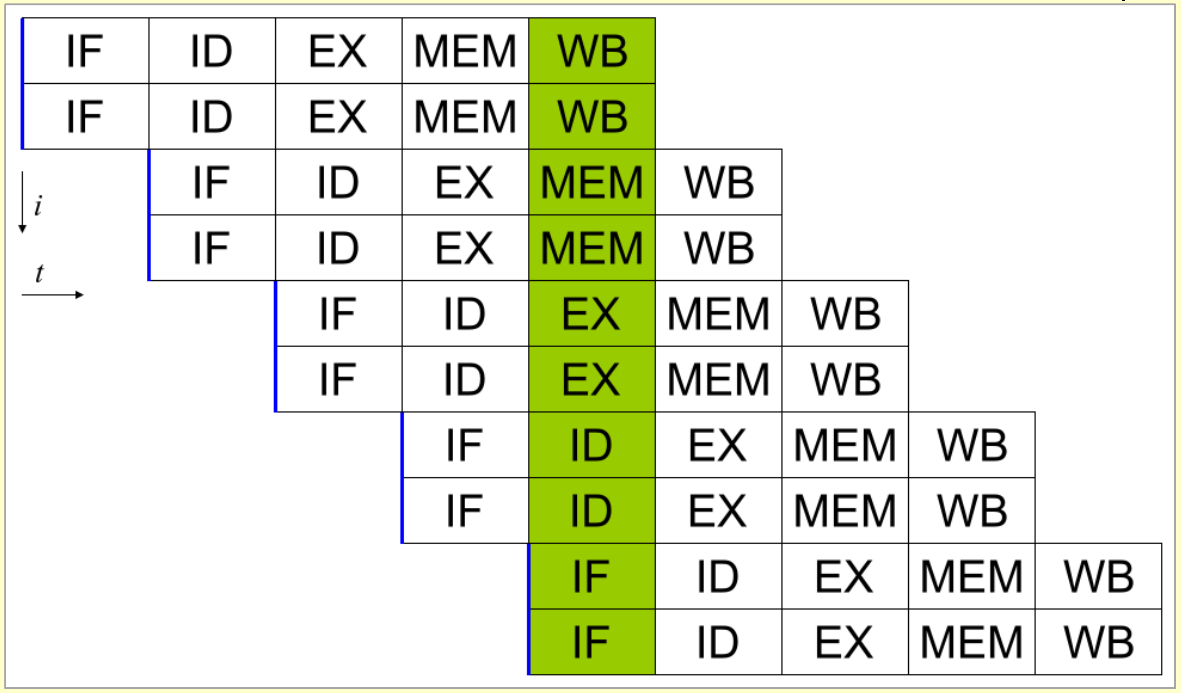
Instruction-level Parallelism “Superscalar” Processors

Professor Hugh C. Lauer
CS-4515, System Programming Concepts

Definition — "Superscalar"

- **The ability to execute more than one instruction per cycle**
  - In a single processor
  - From a *single* instruction stream

Naïve example



"Superscalarpipeline" by Amit6, Wikipedia

Requires

- Ability to fetch more than one instruction at a time from instruction stream
- Multiple execution units
- Ability to deal with multiple control and data hazards on each cycle
- Ability to write and/or forward results
- All to maintain instruction rate > 1 per cycle
  - CPI < 1.0 !

Reading assignment

- **Re-read §C.7**
  - Especially “Dynamically Scheduled Pipelines”
- **Chapter 3**
  - Esp §3.4, “Overcoming Data Hazards with Dynamic Scheduling

Basic idea

- **Replace ID, EX, WB steps of pipeline with:–**
- ***Issue***
  - I.e., dispatch an instruction to a functional unit
- ***Read Operands***
  - Get the operands from wherever they come from
- ***Execute***
  - "Do" the instruction
- ***Write Result***

Scoreboard

Great big combinatorial algorithm!

- **Status of functional units**
- **Status of instructions**
- **Hazards**
  - Including RAW, WAR, WAW
- **Status of registers**
  - More than just the "architectural" registers

Definition: Architectural Registers:–
Registers that are named in assembly language instructions

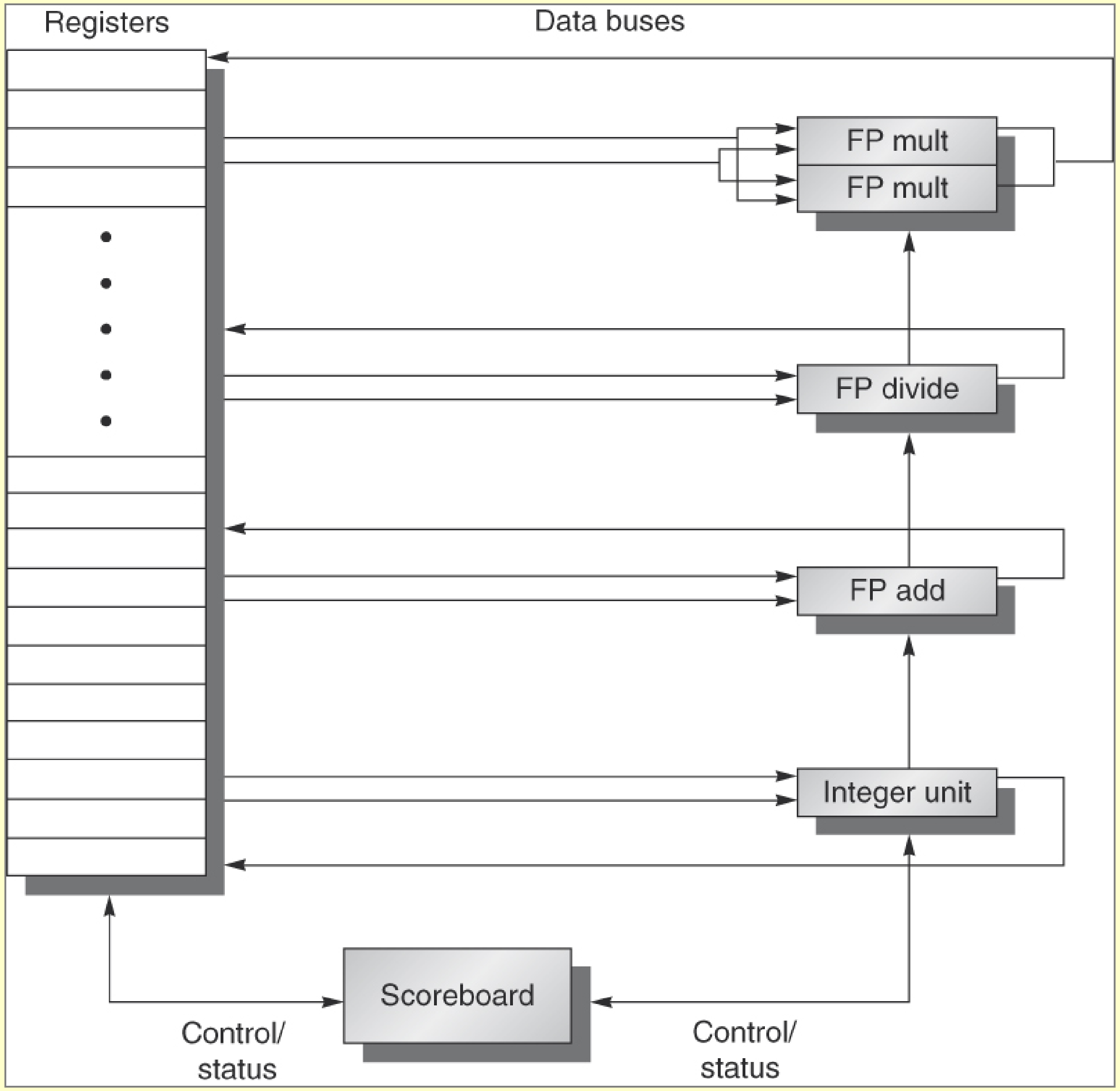

Registers
Data buses
FP mult
FP mult
FP divide
FP add
Integer unit
Scoreboard
Control/ status
Control/ status

Fig C.49

Step 1 — Issue

- **Scoreboard checks functional unit needed by instruction**
  - If free and
  - If does not share destination register with any other active instruction
  - ⇒ *Issue* instruction to functional unit!
- **Guarantees no WAW hazards!**
- **If hazard,**
  - Instruction issue stalls
  - No more instructions will issue until hazards clear!
- **Replaces part of the ID step**

Step 2 — Read Operands

- **Scoreboard monitors source operand availability**
  - Operand available if no earlier issued active instruction will write it
- **When available, functional unit may read operand(s) from register(s)**
  - Begin execution
- **Resolves RAW hazards!**
- **Instructions may be sent into execution out of order.**
- **Replaces the rest of the ID step**

Step 3 - Execution

- Functional unit begins execution upon receiving operands
- When the result is ready, functional unit notifies scoreboard of completed execution
- This step replaces the EX step

Step 4 – Write result

- **Scoreboard checks for WAR hazards**
  - After completion of execution

- **Stalls the completing instruction if necessary**

Step 5 — Retirement

- Flush the instruction from scoreboard

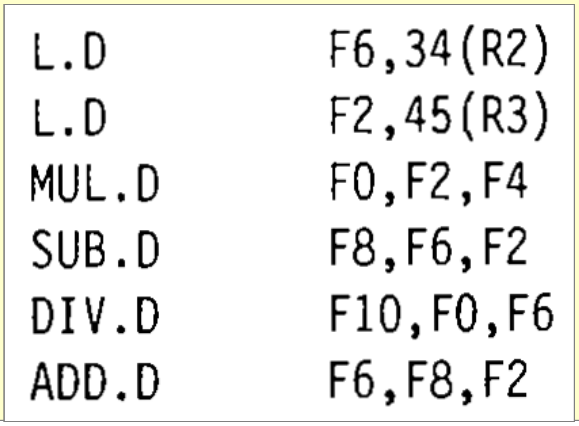
Not in Hennessy & Patterson

In Bryant & O’Hallaron, §5.7

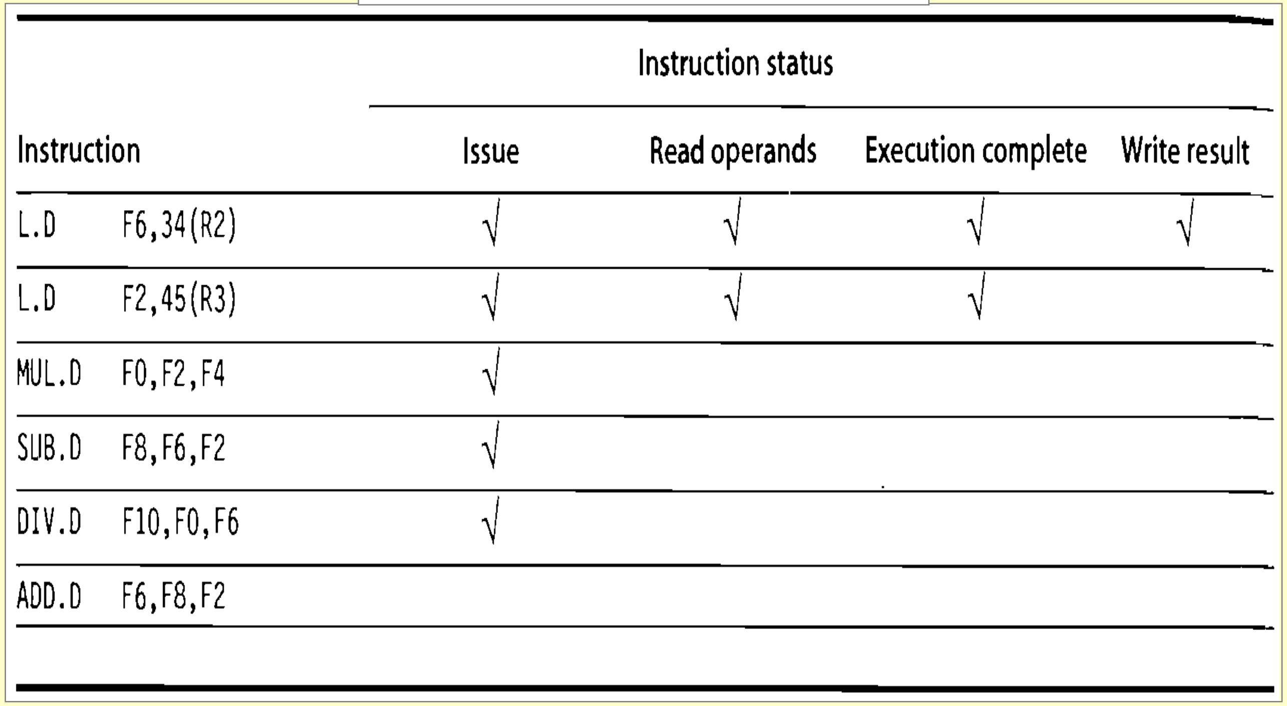
Parts of the Scoreboard

- **Instruction status – Indicates which of the four steps the instruction is in**
- **Functional unit status – Indicates the state of the function unit.**
  - Busy – indicates whether unit is busy
  - Op – operation to perform in the unit
  - 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
- **Register result status – indicates which functional unit will write each register**

Instruction Status

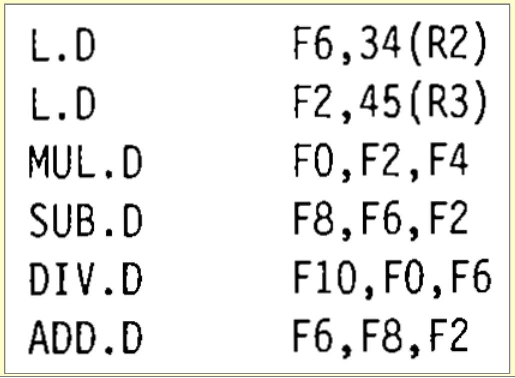


```
L.D       F6,34(R2)
L.D       F2,45(R3)
MUL.D     F0,F2,F4
SUB.D     F8,F6,F2
DIV.D     F10,F0,F6
ADD.D     F6,F8,F2
```

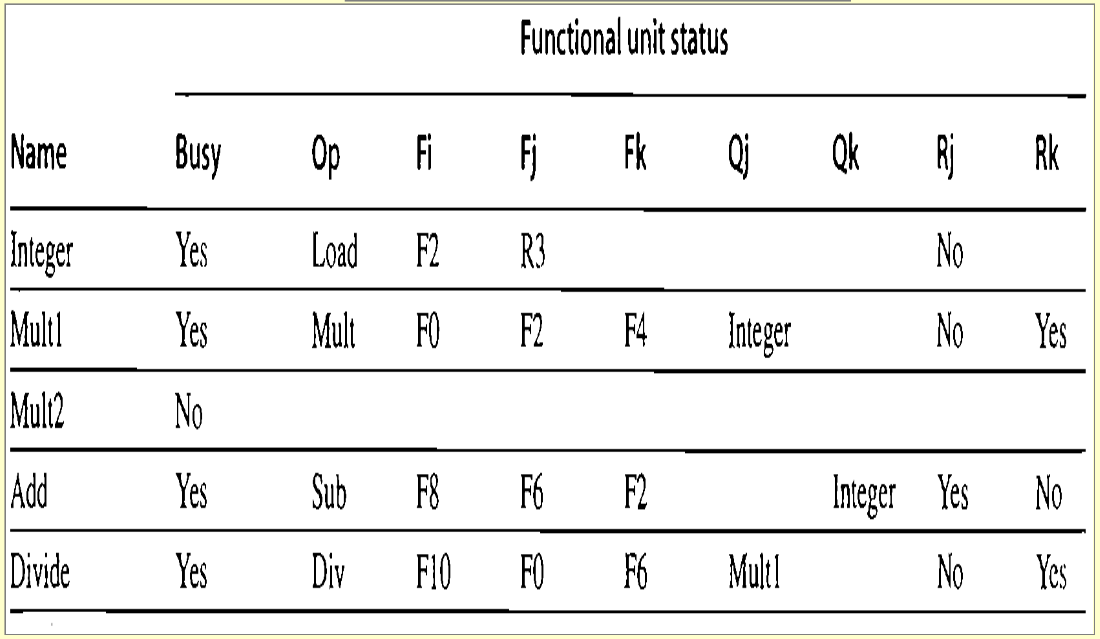


	Instruction status			
Instruction	Issue	Read operands	Execution complete	Write result
L.D F6,34(R2)	√	√	√	√
L.D F2,45(R3)	√	√	√	
MUL.D F0,F2,F4	√			
SUB.D F8,F6,F2	√			
DIV.D F10,F0,F6	√			
ADD.D F6,F8,F2				

Functional Unit Status

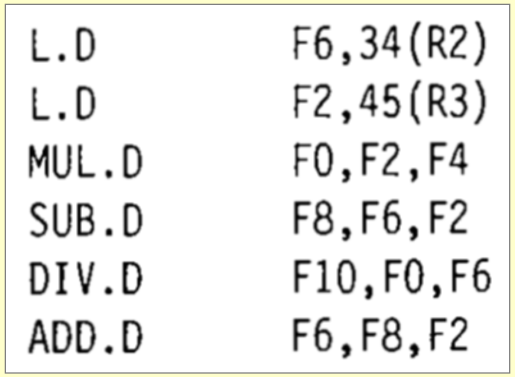


```
L.D        F6,34(R2)
L.D        F2,45(R3)
MUL.D      F0,F2,F4
SUB.D      F8,F6,F2
DIV.D      F10,F0,F6
ADD.D      F6,F8,F2
```

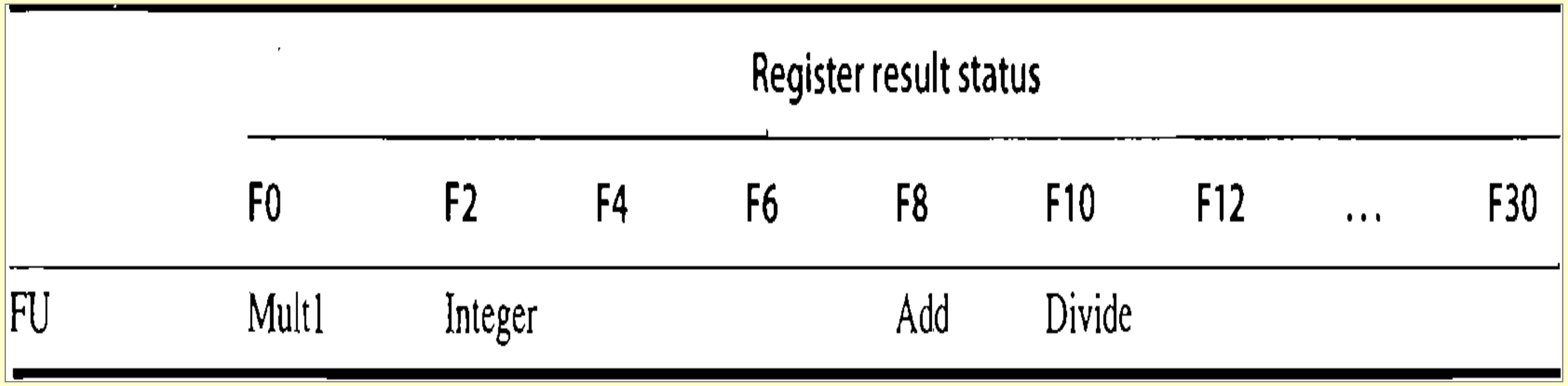


		Functional unit status							
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

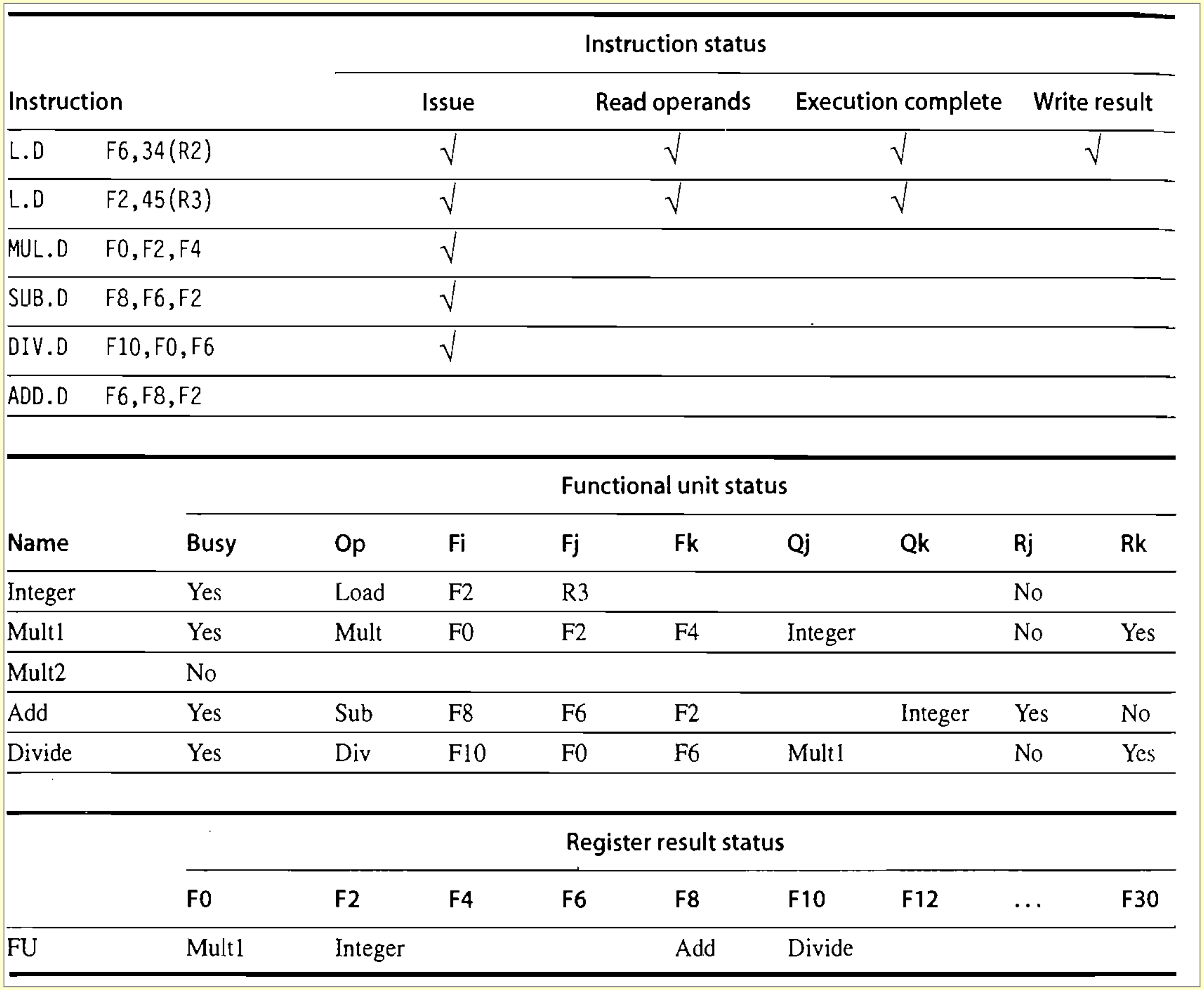
Register Result Status



```
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
```



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



Instruction status

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

Functional unit status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	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			

Dynamic Scheduling

- **Hardware rearranges the execution order of instructions**
  - Helps to avoid stalls
  - Simplifies the compiler
  - Tolerates unpredictable delays (ex., cache misses)
- **Does not change the data flow of the program**
  - Necessary for correct execution

Questions?