Tomasulo's algorithm

- "Weaknesses" in scoreboard:
  - Centralized control
  - No forwarding (more RAW than needed)
- Tomasulo's algorithm as implemented first in IBM 360/91
  - Control decentralized at each functional unit
  - Forwarding
  - Concept and implementation of *renaming registers* that eliminates WAR and WAW hazards

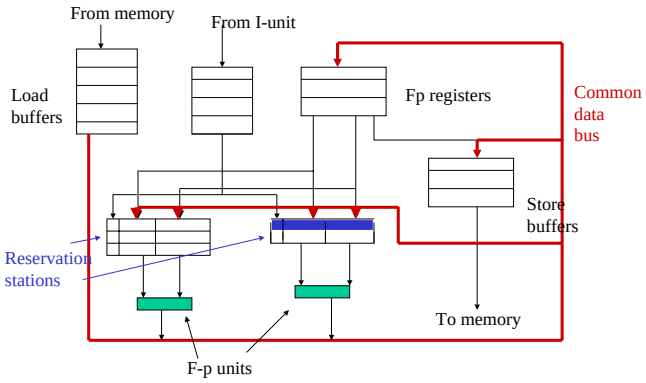
Improving on Dispatch with Reservation Stations

- With each functional unit, have a set of buffers or *reservation stations*
  - Keep operands and function to perform
  - Operands can be values or names of units that will produce the value (register renaming) with appropriate flags
- Not both operands have to be "ready" at the same time
- When both operands have values, functional unit can execute on that pair of operands
- When a functional unit computes a result, it broadcasts its name and the value.

Tomasulo's solution to resolve hazards

- Structural hazards
  - No free reservation station (stall at issue time). No further issue
- RAW dependency (detected in each functional unit -- decentralized)
  - The instruction with the dependency is issued (put in a reservation station) but not dispatched (stalled). Subsequent instructions can be issued, dispatched, executed and completed.
- No WAR and WAW hazards
  - Because of register renaming through reservation stations
- Forwarding
  - Done at end of execution by use of a common (broadcast) data bus

Example machine (cf. Figure 3.2)



An instruction goes through 3 steps

- Assume the instruction has been fetched
- 1. Issue, dispatch, and read operands
  - Check for structural hazard (no free reservation station or no free load-store buffer for a memory operation). If there is structural hazard, stall until it is not present any longer
  - Reserve the next reservation station
  - Read source operands
    - If they have values, put the values in the reservation station
    - If they have names, store their names in the reservation station
  - Rename result register with the name of the reservation station in the functional unit that will compute it

An instruction goes through 3 steps (c'ed)

- 2. Execute
  - If any of the source operands is not ready (i.e., the reservation station holds at least one name), monitor the bus for broadcast
  - When both operands have values, execute
- 3. Write result
  - Broadcast name of the unit and value computed. Any reservation station/result register with that name grabs the value
- Note two more sources of structural hazard:
  - Two reservation stations in the same functional unit are ready to execute in the same cycle: choose the "first" one
  - Two functional units want to broadcast at the same time. Priority is encoded in the hardware

Implementation

- All registers (except load buffers) contain a pair {value,tag}
- The tag (or name) can be:
  - Zero (or a special pattern) meaning that the value is indeed a value
  - The name of a load buffer
  - The name of a reservation station within a functional unit
- A reservation station consists of :
  - The operation to be performed
  - 2 pairs (value,tag) *(Vj,Qj) (Vk,Qk)*
  - A flag indicating whether the accompanying f-u is busy or not

Instruction	Issue	Execute	Write result
Load F6, 34(r2)	yes	yes	yes
Load F2, 45(r3)	yes	yes	
Mul F0, F2, F4	yes		
Sub F8, F6, F2	yes		
Div F10, F0, F6	yes		
Add F6,F8,F2	yes		

Initial: waiting for F2 to be loaded from memory

Reservation Stations

Name	Busy	Fm	Vj	Vk	Qj	Qk
Add 1	yes	Sub	(*Load1*)		0	Load2
Add2	yes	Add			Add1	Load2
Add3	no					
Mul1	yes	Mul		(*F4*)	Load2	0
Mul2	yes	Div		(*Load1*)	Mul1	0

(x) Means a value: contents of x

Qj = 0 means Vj has a value

Register status

F0	F2	F4	F6	F8	F10	F12...
(Mul1)	(Load2)	( )	(Add2)	(Add1)	(Mul2)	

Instruction	Issue	Execute	Write result
Load F6, 34(r2)	yes	yes	yes
Load F2, 45(r3)	yes	yes	yes
Mul F0, F2, F4	yes	yes	
Sub F8, F6, F2	yes	yes	
Div F10, F0, F6	yes		
Add F6,F8,F2	yes		

Cycle after 2nd load has written its result

Reservation Stations

Name	Busy	Fm	Vj	Vk	Qj	Qk
Add 1	yes	Sub	(Load1)	(Load2)	0	0
Add2	yes	Add		(Load2)	Add1	0
Add3	no					
Mul1	yes	Mul	(Load2)	(F4)	0	0
Mul2	yes	Div		(Load1)	Mul1	0

Register status

F0	F2	F4	F6	F8	F10	F12...
(Mul1)	()	( )	(Add2)	(Add1)	(Mul2)	

Instruction	Issue	Execute	Write result
Load F6, 34(r2)	yes	yes	yes
Load F2, 45(r3)	yes	yes	yes
Mul F0, F2, F4	yes	yes	
Sub F8, F6, F2	yes	yes	yes
Div F10, F0, F6	yes		
Add F6,F8,F2	yes	yes	

Cycle after sub has written its result

Reservation Stations

Name	Busy	Fm	Vj	Vk	Qj	Qk
Add 1	no					
Add2	yes	Add	(Add1)	(Load2)	0	0
Add3	no					
Mul1	yes	Mul	(Load2)	(F4)	0	0
Mul2	yes	Div		(Load1)	Mul1	0

Register status

F0	F2	F4	F6	F8	F10	F12...
(Mul1)	()	( )	(Add2)	()	(Mul2)	

Instruction	Issue	Execute	Write result
Load F6, 34(r2)	yes	yes	yes
Load F2, 45(r3)	yes	yes	yes
Mul F0, F2, F4	yes	yes	yes
Sub F8, F6, F2	yes	yes	yes
Div F10, F0, F6	yes	yes	
Add F6,F8,F2	yes	yes	yes

Cycle after add has written its result

Reservation Stations

Name	Busy	Fm	Vj	Vk	Qj	Qk
Add 1	no					
Add2	no					
Add3	no					
Mul1	yes	Mul	(Load2)	(F4)	0	0
Mul2	yes	Div		(Load1)	Mul1	0

Register status

F0	F2	F4	F6	F8	F10	F12...
(Mul1)	()	( )	()	()	(Mul2)	

Instruction	Issue	Execute	Write result
Load F6, 34(r2)	yes	yes	yes
Load F2, 45(r3)	yes	yes	yes
Mul F0, F2, F4	yes	yes	yes
Sub F8, F6, F2	yes	yes	yes
Div F10, F0, F6	yes	yes	
Add F6,F8,F2	yes	yes	yes

Cycle after mul has written its result

Reservation Stations

Name	Busy	Fm	Vj	Vk	Qj	Qk
Add 1	no					
Add2	no					
Add3	no					
Mul1	no					
Mul2	yes	Div	(Mul1)	(Load1)	0	0

Register status

F0	F2	F4	F6	F8	F10	F12...
()	()	( )	()	()	(Mul2)	

Other checks/possibilities

- In the example in the book there is no load/store dependencies but since they can happen
  - Load/store buffers must keep the addresses of the operands
  - On load, check if there is a corresponding address in store buffers. If so, get the value/tag from there (load/store buffers have tags)
  - Better yet, have load/store functional units (still needs checking)
- The Tomasulo engine was intended only for f-p operations. We need to generalize to include
  - Handling branches, exceptions etc
  - In-order completion
  - More general register renaming mechanisms
  - Multiple instruction issues