CS 223 Computer Organization & Architecture

Lecture 32 [30.04.2020]

Dynamic Scheduling with Speculative Execution

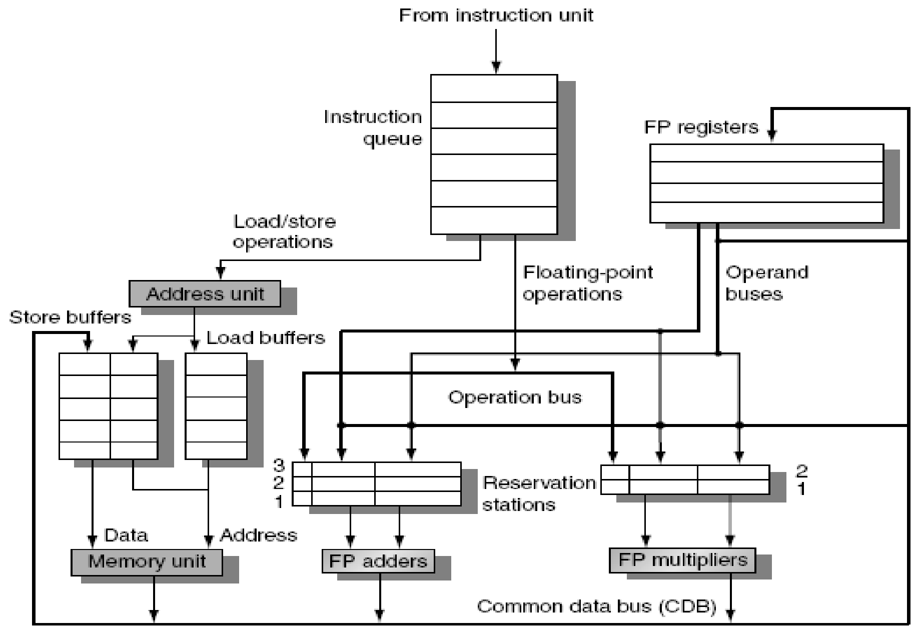
John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

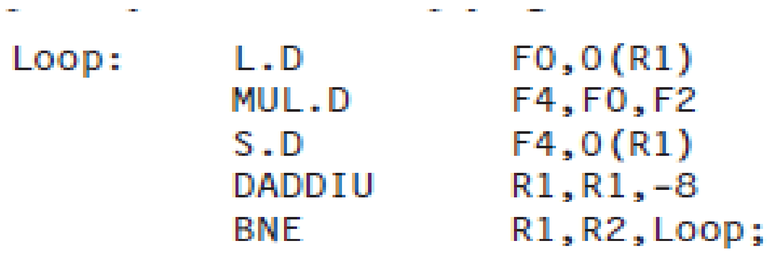
Tomasulo’s Algorithm



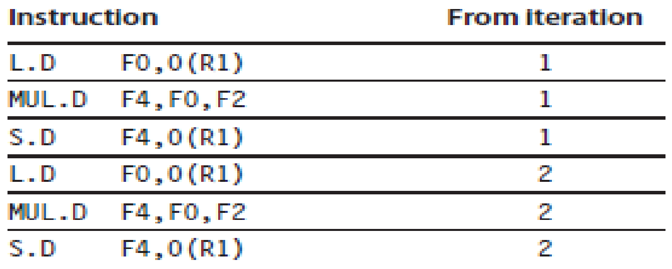
❖ No instruction will initiate execution until all branches that precede it in program order have completed

Tomasulo’s Algorithm for Loops

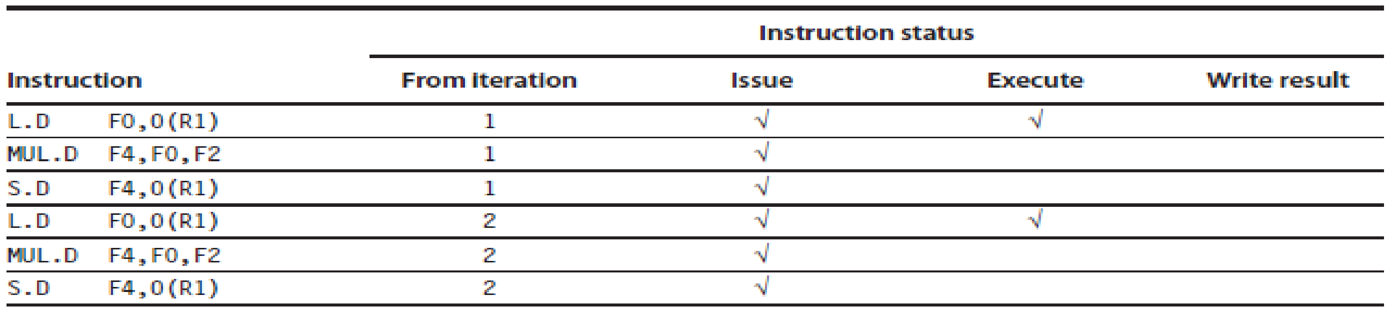
- ❖ How to run loops if no instruction will initiate execution until all branches that precede it in program order have completed?
- ❖ Predict the branches are taken and issue instructions across multiple iterations to reservation stations.



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



Instruction		From iteration
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



			Instruction status		
Instruction		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	√		

Tomasulo’s Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	No						
Load-2	No						
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	No						
Mult-2	No						
Store-1	No						
Store-2	No						

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

F0	F2	F4	F6	F8	F10
0	0	0	0	0	0

Tomasulo’s Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	No						
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	No						
Mult-2	No						
Store-1	No						
Store-2	No						

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

F0	F2	F4	F6	F8	F10
Load-1	0	0	0	0	0

Tomasulo's Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	No						
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	**Yes**	**MUL**		**Reg[F2]**	**Load-1**		
Mult-2	No						
Store-1	No						
Store-2	No						

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

F0	F2	F4	F6	F8	F10
Load-1	0	**Mult-1**	0	0	0

Tomasulo's Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	No						
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	Yes	MUL		Reg[F2]	Load-1		
Mult-2	No						
Store-1	**Yes**	**Store**	**Reg[R1]**			**Mult-1**	
Store-2	No						

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

F0	F2	F4	F6	F8	F10
Load-1	0	Mult-1	0	0	0

Tomasulo's Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	**Yes**	**Load**					**Reg[R1]-8**
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	Yes	MUL		Reg[F2]	Load-1		
Mult-2	No						
Store-1	Yes	Store	Reg[R1]			Mult-1	
Store-2	No						

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

F0	F2	F4	F6	F8	F10
Load-2	0	Mult-1	0	0	0

Tomasulo’s Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	Yes	Load					Reg[R1]-8
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	Yes	MUL		Reg[F2]	Load-1		
Mult-2	**Yes**	**MUL**		**Reg[F2]**	**Load-2**		
Store-1	Yes	Store	Reg[R1]			Mult-1	
Store-2	No						

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

F0	F2	F4	F6	F8	F10
Load-2	0	**Mult-2**	0	0	0

Tomasulo's Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					Reg[R1]+0
Load-2	Yes	Load					Reg[R1]-8
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	Yes	MUL		Reg[F2]	Load-1		
Mult-2	Yes	MUL		Reg[F2]	**Load-2**		
Store-1	Yes	Store	Reg[R1]			Mult-1	
Store-2	**Yes**	**Store**	**Reg[R1]**			**Mult-2**	

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

F0	F2	F4	F6	F8	F10
Load-2	0	Mult-2	0	0	0

Tomasulo’s Algorithm-Loop Example

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	No	Load					Reg[R1]+0
Load-2	No	Load					Reg[R1]-8
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	Yes	MUL		Reg[F2]	Load-1		
Mult-2	Yes	MUL		Reg[F2]	Load-2		
Store-1	Yes	Store	Reg[R1]			Mult-1	
Store-2	Yes	Store	Reg[R1]			Mult-2	

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

F0	F2	F4	F6	F8	F10
Load-2	0	Mult-2	0	0	0

Load-Store Order Conflict

- A load and a store can safely be done out of order, provided they access different addresses. If a load and a store access the same address, then either
- The load is before the store in program order and interchanging them results in a WAR hazard
- The store is before the load in program order and interchanging them results in a RAW hazard.
- Interchanging two stores to the same address results in a WAW hazard.

Load-Store Order Conflict

- To allow a load and a store to interchange their order in a execution perform special address check.
- To determine if a load can be executed at a given time, the processor can check whether any uncompleted store that precedes the load in program order shares the same data memory address as the load.
- A store must wait until there are no unexecuted loads or stores that are earlier in program order that share the same data memory address.

Hardware-Based Speculation

- How to overcome control hazard in Tomasulo's approach?
- Speculate – **Fetch, Issue and Execute** as if branch predictions are always correct; **commit** the results only if prediction was correct
- **Instruction commit:** allowing an instruction to update the register file/memory ONLY when instruction is no longer speculative
- Need an additional hardware to prevent any irrevocable action until an instruction commits
- Reorder buffer (ROB) – holds the result of instruction between completion and commit
- Modify reservation stations- Operand source is now reorder buffer entry instead of functional unit

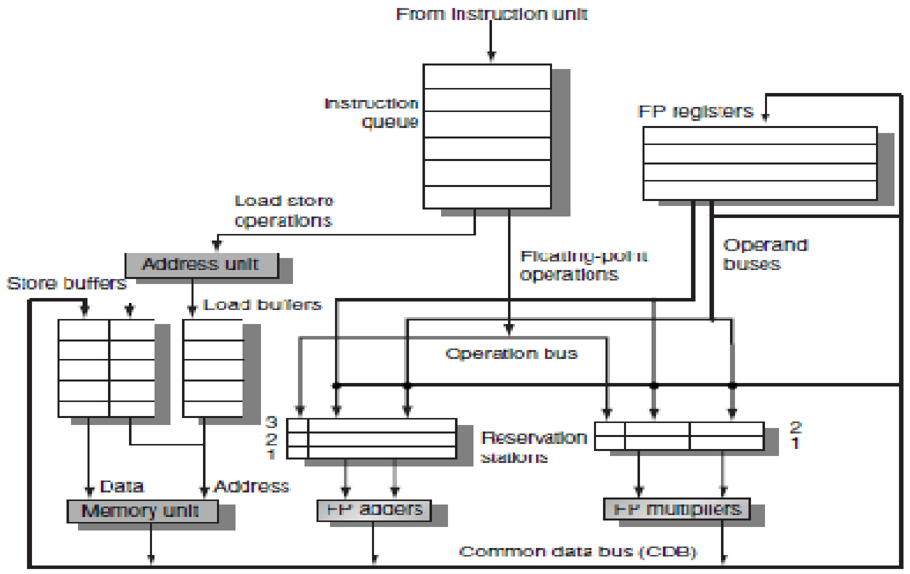
Hardware-Based Speculation

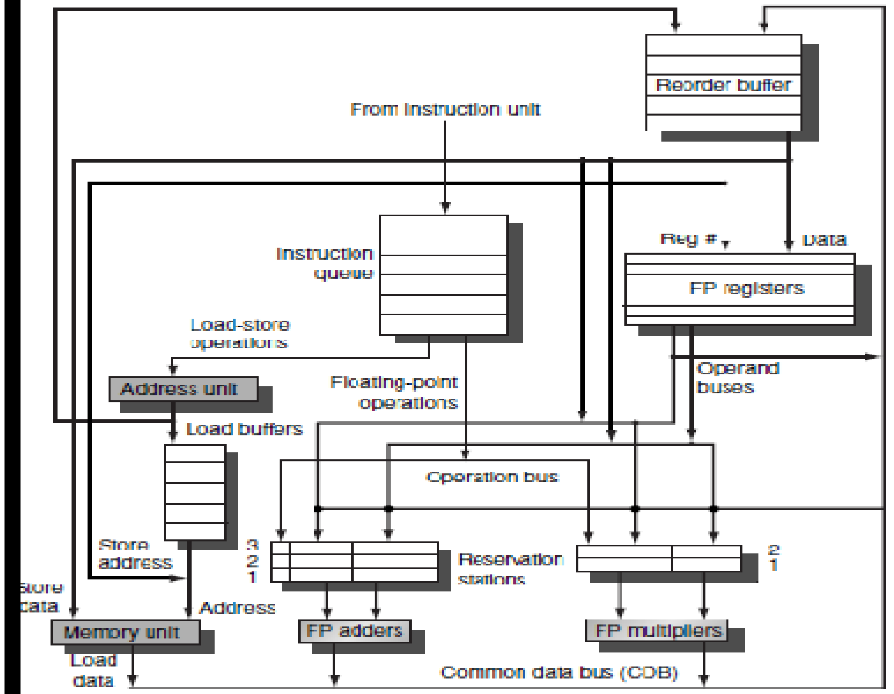
- Hardware based speculation needs 3 techniques
  - **Dynamic branch prediction** to choose which instructions to execute,
  - **Speculation** to allow the execution of instructions before the control dependences are resolved (with the ability to undo),
  - **Dynamic scheduling** to deal with the scheduling of different combinations of basic blocks.
- Allow an instruction to execute and to bypass its results to other instructions
- Prevent an instruction to perform any updates on writing in Reg/Mem, until the instruction is no longer speculative.

Reorder Buffer (ROB)

- Four fields:
  - Instruction type: branch/store/register
  - Destination field: register number / memory address
  - Value field: output value
  - Ready field: completed execution OR only issued
- Register values and memory values are not written until an instruction commits
- On misprediction:
  - Speculated entries in ROB are cleared

A comparison





Tomasulo's Algorithm **Tomasulo's Algorithm with ROB**

Operations with ROB

- Issue
- Execute
- Write Result (Complete)
- Commit

Operations with ROB - Issue

- ❖ Get an instruction from the instruction queue.
- ❖ Issue the instruction if there is an empty reservation station and an empty slot in the ROB.
- ❖ Get a slot number (tag) from ROB for this instruction.
- ❖ Send the operands to the RS if they are available in either RF or in the ROB. (speculative register read)
- ❖ The number of the ROB entry is also sent to the RS to tag the result when it is placed on the CDB.
- ❖ If either all reservations are full or the ROB is full, then instruction issue is stalled until both have available entries.

Operations with ROB-Execute

- ❖ If one or more of the operands is not yet available, monitor the CDB waiting for the value.
- ❖ When both operands are available at a reservation station, execute the operation.
- ❖ Instructions may take multiple clock cycles in this stage, and loads still require two steps in this stage.
- ❖ Stores need to have the base register value available at this step for effective address calculation.

Operations with ROB– Write Result

- Write result—When the result is available, write it on the CDB (with the ROB tag).
- Data in CDB will go into the ROB and to RS waiting for the result.
- If the value to be stored is available, it is written into the Value field of the ROB entry for the store.
- If address for store is available then store that in ROB.
- If the value to be stored is not available yet, the CDB must be monitored until that value is broadcast, at which time the Value field of the ROB entry of the store is updated.

Operations with ROB - Commit

- Commit—This is the final stage of an instruction, after which only its result remains.
- Different sequences of actions at commit if instruction is
  - a branch with an incorrect prediction,
  - a store
  - other instruction (normal commit).
- The normal commit case occurs when an instruction reaches the head of the ROB and its result is present in the buffer- the processor updates the RF with the result and removes it from ROB.
- Committing a store is similar except that memory is updated rather than a result register.

Operations with ROB - Commit

- Commit—This is the final stage of an instruction, after which only its result remains.
- When a branch with incorrect prediction reaches the head of the ROB, it indicates that the speculation was wrong.
- The ROB is flushed and execution is restarted at the correct successor of the branch.
- If the branch was correctly predicted, the branch is finished.

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/