Instruction-Level Parallelism and Its Dynamic Exploitation

Chapter 3 - Computer Architecture : *A Quantitative Approach* - Hennessy & Patterson

More ILP Through Multiple Issue

- More than one instructions are to be issued per clock cycle to reduce the CPI less than one (1). Such processors are called multiple-issue processors.
- There are essentially two kinds of multiple-issue processors - superscalar and VLIW (very large instruction word). We are considering EPIC (explicitly parallel instruction computer) under VLIW.

More ILP Through Multiple Issue

- A superscalar processor issues **varying number of instructions per clock cycle** e.g. 0 to 8.
- It may be **statically scheduled** using compiler technique. The execution is **in-order.**
- It may be **dynamically scheduled** using technique based on Tomasulo's algorithm. The execution is out-of-order.
- Modern superscalar processors use **speculation.**

More ILP Through Multiple Issue

- **A VLIW processor issues a fixed number of instructions put in a large word e.g. 128-bit.**
- **An EPIC processor issues a fixed size multiple instruction packets with an explicitly marked parallelism.**
- **Both these processors are statically scheduled by a compiler.**

A Few Examples

Name	Issue	Hz. Det.	Sched.	Char.	Expl.
SupS.	Dyn.	H-ware	Static	in-order	UltraSPARC II/III
SupS.	Dyn.	H-ware	Dyn.	some out-o-ord	IBM Power2

A Few Examples

Name	**Issue**	**Hz. Det.**	**Sched.**	**Char.**	**Expl.**
SupS.	Dyn.	H-ware	Dyn. Specu.	out-o-order	Pentium III/4
VLIW	Static	S-ware	Static		Trimedia, i860
EPIC	Static	S-ware	Static		Itanium

Superscalar

- Most of the old superscalars and even the current embedded processors have static scheduling.
- Most modern desktop and server superscalar processors are dynamically scheduled. In fact many of these has speculation.

Statically Scheduled Superscalar

- **Instruction issue is in-order and the pipeline hazards are checked at the time of issue.**
- **Hazards are to be checked among the instructions currently being issued and instructions already in execution.**
- **If there is a structural or data hazard for some instruction, it will be stalled along with everything following this.**

Instruction Issue

- Consider a 4-issue, static, superscalar processor.
- The **instruction fetch unit** can supply zero (0) to four (4) instructions to the pipeline in a clock cycle. This is called an issue packet.
- The **fetch unit examines the instructions in program order** for structural and data hazards with the earlier instructions in execution and also with the earlier instructions (in program order) in the issue packet.

Instruction Issue

- Naturally the instruction issue process is sufficiently complex and performing this in a single clock will restrict the minimum clock frequency.
- The issue stage is split and pipelined to solve this problem.
- Unfortunately the splitting of the issue stage is not straightforward and gives rise to other problems e.g. checking of hazards between two issue packets in the pipeline, higher branch penalties etc.

SSS 2-Issue MIPS

- **One of the two instructions is load, store, branch or ALU operation. The other one is a floating-point operation except floating-point load/store.**
- **The processor need to fetch two instructions (64-bits) and there is no order restriction.**

Fetch and Issue

- Two instruction fetch is more complex as the first instruction may be the last word of the cache block.
- Some processor in this case will fetch only one instruction, but it can be solved if there is an instruction prefetch unit.
- The issue unit will check whether zero (0), one (1) or two (2) can be issued (depending on hazards).

Hazards

- The main sources of extra hazards in this case are the floating-point load, store and move.
- If there is a load to a floating-point register along with a floating-point operation, there may be a structural hazard on the floating-point register file.
- It may give rise to a RAW hazard if for example the second instruction of the issue packet is a floating-point operation using an operand loaded by the first instruction.

Multiple or Pipelined FP Unit

- **Floating-point operations take more than one clock cycle.**
- **The issue one FP operation per clock cycle requires either multiple FP units or pipelined FP unit.**

2-Issue Superscalar Pipeline

Type	Pipeline						
Int.Inst.	IF	ID	EX	MM	WB		
FP Inst.	IF	ID	EX	EX	EX	WB	
Int.Inst.		IF	ID	EX	MM	WB	
FP Inst.		IF	ID	EX	EX	EX	WB

Imprecise Exception

- **A floating-point instruction that is earlier in program order may be completed after an integer instruction and there may be an exception in the FP instruction after the completion of the integer instruction - the exception is imprecise.**

Latency

- In the classic 1-issue 5-stage pipeline the load latency was 1-cycle i.e. the next instruction cannot use the data immediately.
- In the 2-issue version the next three (3) instructions cannot use the loaded data immediately.
- The compiler should be smarter to utilize the ILP.

Multiple Issue with Dynamic Scheduling

- **Dynamic scheduling boosts performance in the face of data hazards.**
- **Issue restriction will be relaxed.**

Two Approaches

- There are essentially two different approaches for issuing multiple instructions per clock cycle in a dynamically scheduled processor.
- The essential steps are assigning a reservation station and updating the pipeline control tables.
- Let this be an extended Tomasulo's algorithm - instructions are not issued to the reservation stations out-of-order.

Two Approaches

- Use **half of a clock cycle** for **each instruction**, or
- Build necessary logic to **handle two instructions at once.**
- Modern superscalars that issue four or more instructions uses both the techniques.

Dynamic Branch Prediction

- How can the dynamic branch prediction be integrated with dynamic scheduling?
- IBM 360/91 used static prediction. Instructions from the predicted target were fetched and issued but not executed until the branch decision is completed.

An Example

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

An Example: Assumptions

- We use an extended **Tomasulo's algorithm.**
- There is **no restriction** on the **two instructions issued.** They can be issued even if they are **dependent.**
- Integer unit is used for both **ALU operations** and **effective address** computation.
- There is a separate FP functional unit for each operation type.
- There are **two CDBs.**

An Example: Assumptions

- Issue and write takes one clock cycle each.
- There is a dynamic branch prediction hardware and a separate functional unit to evaluate branch condition. The prediction is perfect! There is no branch delay slot.
- Number of cycles of latency between the result producing instruction and the result consuming instruction are, 1-cycle for integer ALU operations, 2-cycles for load and 3-cycles for FP add.

An Example: Assumptions

- The loop is **dynamically unrolled** and two instructions are issued whenever possible.

Issue, Execution ···

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Comm**
1	L.D F0,0(R1)	1	2	3	4	
1	ADD.D F4,F0,F2	1	5		8	
1	S.D F4,0(R1)	2	3	9		
1	DADDIU R1,R1,-8	2	4		5	
1	BNE R1,R2,Loop	3	6			

Issue, Execution $\cdots$

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Comm**
2	L.D F0,0(R1)	4	7	8	9	
2	ADD.D F4,F0,F2	4	10		13	
2	S.D F4,0(R1)	5	8	14		
2	DADDIU R1,R1,-8	5	9		10	
2	BNE R1,R2,Loop	6	11			

Issue, Execution · · ·

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Comm**
3	L.D F0,0(R1)	7	12	13	14	
3	ADD.D F4,F0,F2	7	15		18	
3	S.D F4,0(R1)	8	13	19		
3	DADDIU R1,R1,-8	8	14		15	
3	BNE R1,R2,Loop	9	16			

Resource Usage Table

Clock	ALU	FPU	D-Cache	CDB
2	L.D$_1$			
3	S.D$_1$		L.D$_1$	
4	DADDIU$_1$			L.D$_1$
5		ADD.D$_1$		DADDIU$_1$
6				

Resource Usage Table

Clock	ALU	FPU	D-Cache	CDB
7	L.D$_2$			
8	S.D$_2$		L.D$_2$	ADD.D$_1$
9	DADDIU$_2$		S.D$_1$	L.D$_2$
10		ADD.D$_2$		DADDIU$_2$
11				

Resource Usage Table

Clock	ALU	FPU	D-Cache	CDB
12	L.D$_3$			
13	S.D$_3$		L.D$_3$	ADD.D$_2$
14	DADDIU$_3$		S.D$_2$	L.D$_3$
15		ADD.D$_3$		DADDIU$_3$
16				

Resource Usage Table

Clock	ALU	FPU	D-Cache	CDB
17				
18				ADD.D$_3$
19			S.D$_3$	
20				

The Example

- One iteration takes three (3) clock cycles. Instructions per clock cycle is $\frac{5}{3} = 1.67$.
- Instruction completion rate is approximately $\frac{15}{16} = 0.94$.
- The amount of overhead per iteration of the loop is high. Two (2) out of five instructions are overhead.
- There is loss of one cycle delay due to the control hazard of branch.

Hardware-Based Speculation

- The control dependence turns out to be the heavier burden as people try to get more out of ILP by issuing multiple instructions per clock cycle.
- There may be a branch on every cycle.
- The problem of control dependence can be handled by speculating the outcome of branches.
- This a subtle and important extension over the branch prediction.

Hardware-Based Speculation

- The main ideas are:
  - Dynamic branch prediction to choose the instruction to execute,
  - Speculation to allow the execution of instructions before the control dependence is resolved. It should have the ability to undo the effect of incorrect speculation.
  - Dynamic scheduling of different basic blocks.
- The program execution is driven by data-flow. Operations are performed as soon as the operands are available.

Processors

- **Large number of modern processors use speculation.**
- **PowerPC 603/604/G3/G4, MIPS R10000/R12000, Pentium III/4, Alpha 21264, AMD K5/K6/Athlon.**

Modification of Tomasulo

- **Tomasulo's algorithm** can be **modified** to support **speculation**.
- The **actual completion of instruction (commit)** is separated from **bypassing of results** from one to another.
- **Execution** and **bypassing of result** may **continue on speculation** as long as **no update (commit)** takes place.
- **Registers are updated** once the **instruction is no more speculative**. This stage is called **instruction commit**.

Key Ideas

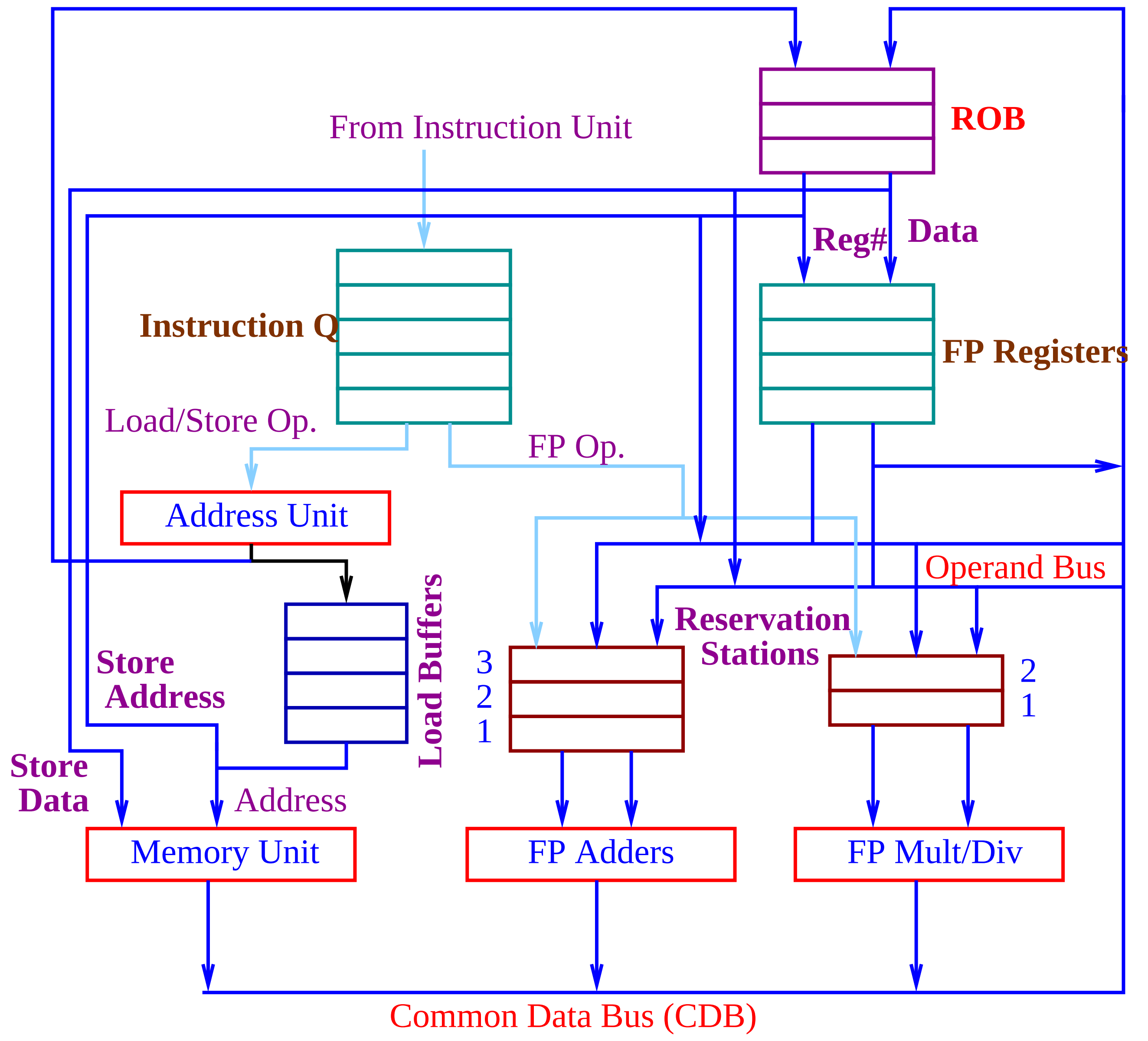
- **Instruction execution** is **out-of-order**, but the **instruction commit** is **in-order**.
- All **irreversible actions** e.g. **state change**, and **exception** are **prevented until commit**.
- The instruction execution may be over well before it is **committed**; and it is necessary to **hold the results** and **pass them** to other instructions.
- A buffer called the **reorder buffer (ROB)** is used for this purpose.

Reorder Buffer (ROB)

- **Reorder buffer (ROB) provides additional registers like the Tomasulo's reservation stations.**
- **A ROB holds the result of an instruction from its completion to commit. The register file is updated only at commit.**
- **Similar to the reservation stations the ROB is also the source of operands for other instructions.**
- **The ROB also replaces the store buffers of Tomasulo Algorithm.**

Different Fields of ROB

- **Instruction Type**: type of instruction e.g. memory, ALU, branch etc.
- **Destination Field**: destination register number, memory address.
- **Value Field**: value of the instruction result.
- **Ready Field**: execution is complete and the value is ready.


ROB
From Instruction Unit
Reg#
Data
Instruction Q
FP Registers
Load/Store Op.
FP Op.
Address Unit
Operand Bus
Load Buffers
Reservation Stations
Store Address
3
2
1
2
1
Store Data
Address
Memory Unit
FP Adders
FP Mult/Div
Common Data Bus (CDB)

Reservation Station, Store Buffer & ROB

- ROB completely **replaces the store buffers.**
- The **register renaming is done by the ROB** and **not by the reservation stations.** But still the **reservation stations** are necessary to hold the **operation** and **operands** for the period of **instruction issue** to **execution.**
- **Results before commit are stored in ROB.** The **tag of the result** sent on CDB is the name of a ROB.

Four Stages

- **Instruction issue**
- **Instruction execute**
- **Write result**
- **Commit**

Issue

- Get the **next instruction from the queue.** Issue it if there is an **free reservation station** and a **free ROB slot**.
- **Send the operands to the reservation station** if they are **available in the ROB** or in the **register file**.

Issue

- Send the **ROB number** to the **reservation station**. This number will be used as the **tag** when the **result is placed on the CDB**.
- The instruction issue is **stalled** if there is **no free reservation station** or **no ROB slot**.
- In a **dynamically scheduled processor**, this stage is often called **dispatch**.

Execute

- If **both operands are available** in the **reservation station**, the operation is performed. It may take **multiple clock cycles.**
- **Load** instruction require **two steps** - **effective address** calculation and **actual load.**
- For **store** only the **effective address** calculation is done at this stage.

Write Result

- When the **result is available** and a **CDB** is **free**, it is put on the CDB along with the **ROB tag**.
- The **data from the CDB** are written in the appropriate **ROB slot** and in the **reservation stations** that need this result as operand.
- If the **value to store** is available, it is **written** in the **ROD slot for store**.

Commit

- **Normal commit**: instruction reaches at the **head of the ROB**, **register is updated** and the entry is removed from the ROB.
- **Commit Store**: **memory is written** when the store is at the head of the buffer.
- When a branch with **incorrect prediction** reaches the **head of the ROB**, indicating a **wrong speculation**, the **ROB is flushed** and execution starts from the correct successor of the branch.

The Old Example

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

We consider the state when MUL.D is ready to write.

Tomasulo: Reservation Stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
L_1	no						
L_2	no						
A_1	no						
A_2	no						
A_3	no						
M_1	yes	mul	M[45 + R3]	F4			
M_2	yes	div		M[34+R2]	M_1		

Tomasulo: Register Status (Qi)

F0	F2	F4	F6	F8	F10	F12	$\cdots$	F30
M_1					M_2			

Speculation: Reorder Buffer

No.	Busy	Inst.	State	Dest	Val.
1	no	L	Comm.	F6	M[34+R2]
2	no	L	Comm.	F2	M[34+R2]
3	yes	MUL	Wr.	F0	#2 $\times$ F4
4	yes	SUB	Wr.	F8	#1 $-$ #2
5	yes	DIV	Exe.	F10	#1 $-$ #2
6	yes	ADD	Wr.	F6	#4 + #2

MUL.D is at the head of the ROB.

Speculation: Reservation Stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
L_1	no							
L_2	no							
A_1	no							
A_2	no							
A_3	no							
M_1	yes	mul	M[45 + R3]	F4			3	
M_2	yes	div		M[34+R2]	M_1		5	

Speculation: Register Status

Fld	F0	F2	F4	F6	F8	F10	F12	$\cdots$	F30
ROB#	3			6	4	5			
Busy	y	n	n	y	y	y			

Dynamic Scheduling vs Speculation

- In speculation no instruction after the first incomplete one has committed, though some of them are complete.
- In dynamic scheduling they already have changed the states.
- It is easy to maintain precise exception in speculation. If there is an exception in MUL.D, the following instructions can be flushed without any side effect.
- In-order commit and precise exception.

Branch Mis-prediction

- Branch misprediction recovery starts as early as possible.
- All entries following the mis-predicted branch are removed.
- Correct branch successor is fetched.
- Mis-prediction has higher impact.

Exception Handling

- Exception is recorded but not raised until the instruction is ready to commit.
- Exception in the trace of mis-predicted branch is also flushed out.

Store

- In **Tomasulo's dynamic scheduling**, when the **effective address** and the **data to store** is available (and there is no hazard), it is written in the memory.
- But in **speculation with ROB**, memory is written only when the store is at the **head of ROB** - the instruction is no more speculative.

Hazards through Memory

- **The store takes place when it is at the head of the ROB and there cannot be any pending load or store, so there cannot be any WAR or WAW hazard through memory.**

Hazards through Memory: RAW

- The second phase of load is not started if there is an earlier store with the same address in the ROB.
- The effective address computation of load with respect earlier stores should be in program order.

Multiple Issue with Speculation

- **A speculative processor can be multiple issue.**
- **The major challenges are -**
  - **Instruction issue and monitoring the CDB for completed instruction.**
  - **Multiple instruction commit per clock cycle is necessary to maintain greater than one throughput.**

An Example

```
Loop:   LD     R2,0(R1)
        DADDIU R2,R2,1
        SD     R2,0(R1)
        DADDIU R1,R1,4
        BNE    R2,R3,LOOP
```

Separate integer unit for effective address calculation, ALU operation and branch condition evaluation.

2-Issue, without Speculation

		Clock			
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**
1	LD R2,0(R1)	1	2	3	4
1	DADDIU R2,R2,1	1	5		6
1	SD R2, 0(R1)	2	3	7	
1	DADDIU R1,R1,4	2	3		4
1	BNE R2,R3, LOOP	3	7		

2-Issue, without Speculation

		Clock			
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**
2	LD R2,0(R1)	4	8	9	10
2	DADDIU R2,R2,1	4	11		12
2	SD R2, 0(R1)	5	9	13	
2	DADDIU R1,R1,4	5	8		9
2	BNE R2,R3, LOOP	6	13		

2-Issue, without Speculation

		Clock			
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**
3	LD R2,0(R1)	7	14	15	16
3	DADDIU R2,R2,1	7	17		18
3	SD R2, 0(R1)	8	15	19	
3	DADDIU R1,R1,4	8	14		15
3	BNE R2,R3, LOOP	9	19		

2-Issue, with Speculation

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Commit**
1	LD R2,0(R1)	1	2	3	4	5
1	DADDIU R2,R2,1	1	5		6	7
1	SD R2, 0(R1)	2	3			7
1	DADDIU R1,R1,4	2	3		4	8
1	BNE R2,R3, LOOP	3	7			8

2-Issue, with Speculation

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Commit**
2	LD R2,0(R1)	4	5	6	7	9
2	DADDIU R2,R2,1	4	8		9	10
2	SD R2, 0(R1)	5	6			10
2	DADDIU R1,R1,4	5	6		7	11
2	BNE R2,R3, LOOP	6	10			11

2-Issue, with Speculation

		Clock				
Iter	**Inst**	**Iss**	**Exe**	**Mem**	**CDB**	**Commit**
3	LD R2,0(R1)	7	8	9	10	12
3	DADDIU R2,R2,1	7	11		12	13
3	SD R2, 0(R1)	8	9			13
3	DADDIU R1,R1,4	8	9		10	10
3	BNE R2,R3, LOOP	9	13			14