Super Scalar

Kalyan Basu

basu@cse.uta.edu

Super scalar Pipelines

- A pipeline that can complete more than 1 instruction per cycle is called a super scalar pipeline.
- We know how to build pipelines with multiple functional units.
- If we can issue more than 1 instruction into the pipe at a time, then perhaps we can complete more than 1 instruction per cycle.
- This implies that we need to fetch and decode 2 or more instructions per cycle.

Multiple Issue Processors

Sperscalar Processors

Variable number of instructions per clock cycle

Instruction Scheduling

Statically: Compiler technique

Instruction execution in order of sequence

dynamically: Tomasulo's Algorithm

Instructions are out of order execution

VLIW : Very Long Instruction Word

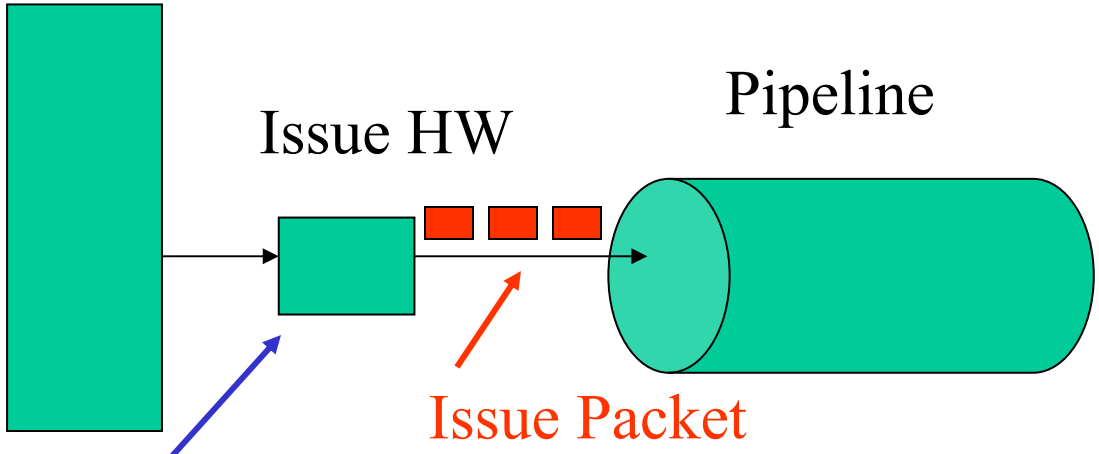
Fixed number of instructions formatted as a large instruction or a fixed instruction packet with parallelism among instructions [***EPIC***: explicitly parallel Instruction Computing]

Statically scheduled by the compiler

Multiple-Issue Processor Types

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristics	Examples
Super scalar (static)	dynamic	HW	static	in-order execution	SUN UltraSPARC
Super scalar (dynamic)	dynamic	HW	dynamic	some out of order	IBM Power 2
Super scalar (speculative)	dynamic	HW	dynamic with speculation	in-order execution with speculation	Pentium III/4, Alpha HP PA8500, IBM RS64III
VLIW/LIW	static	SW	static	no hazards between issue packets	Trimedia,i860
EPIC	mostly static	mostly SW	mostly static	explicit dependency marked by compiler	Itanium

Super scalar



Complexity of HW
This stage is pipelined in all dynamic super scalar system

0-8 instruction per cycle
Static scheduling
all pipe line hazards are checked
instructions in order
Pipeline control logic will check hazards between the instructions in execution phase and the new instruction sequences. In case of hazard, only those instructions preceding that one in the instruction sequence will be issued.
All instructions are checked at the same time by Issue HW

Hardware based Speculation

Instruction Level Parallelism challenge: ***Control dependencies (?)***

- Branch Prediction predict branches accurately
- Multiple branches in one cycle require management of control dependencies

Speculation predicts the outcome of branch instruction and execute the instruction based on this speculation

Operation	Dynamic Scheduling	Speculation
Fetch	Done	Done
Issue	Done	Done
Execute	not Done	Done

HW Speculation

Three functions

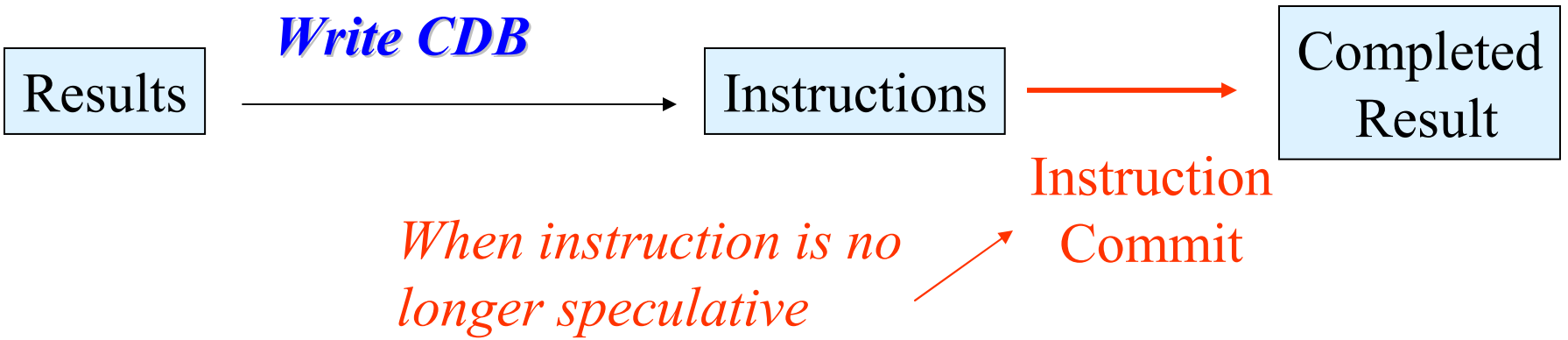
1. Dynamic branch prediction to choose which instruction to execute
2. Speculation to allow ***continuation of execution before the control dependencies are resolved*** with capability to ***undo effects of incorrectly speculated*** the sequence.
3. Dynamic scheduling to deal with the scheduling of different combination of basic blocks.

Used: Power PC 603/604/G3/G4, Intel Pentium II/III/4, Alpha 21264, AMD K5/K6/Athlon, MIPS R11000/R1 12000

HW Speculation Implementation

Tomasul's Algorithm

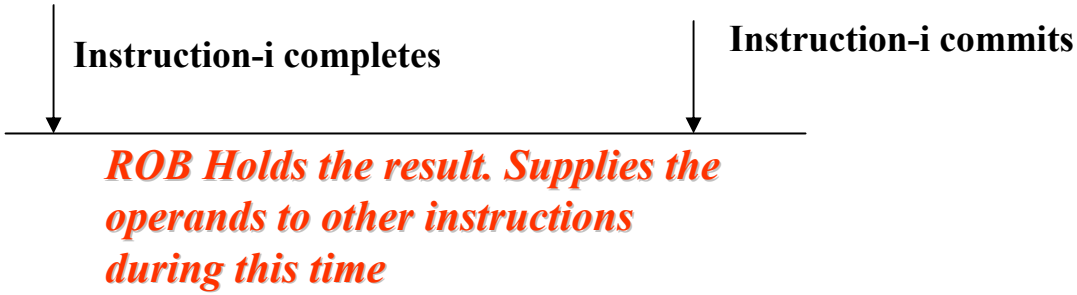
Modification

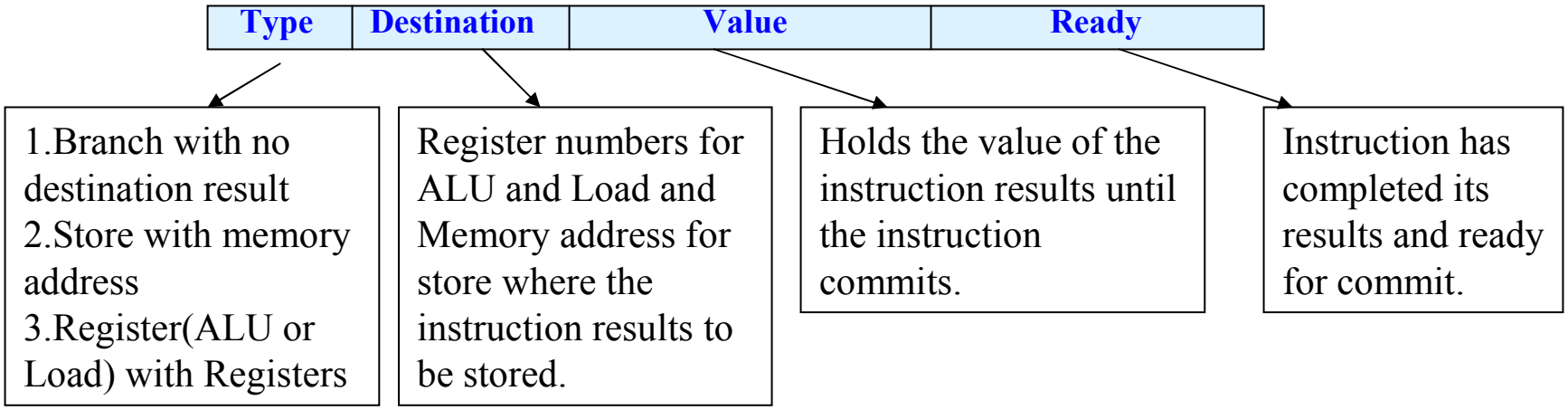


ROB

Reorder Buffer : ROB

It holds the result till Instruction Commits





4 Steps

Issue

Get instruction from Instruction Queue. Get a free reservation station and a free ROB buffer. Send the operand to reservation station, if they are available in registers or the ROB. Update the control status of reservation station and ROB buffers as busy. The number of ROB allocated for the results is given to Reservation Station to send the results when it is placed in CDB. If either reservation stations are full or ROB buffers are full, the instruction is stalled.

4 Steps

Execute

If the operands are not available, monitor CDB to get the operands of the instruction. When both operands are available at the reservation station, execute the instruction. ALU operation may need multiple clock cycle. Load will require two cycle. Store will only compute the effective address and will require one cycle.

4 Steps

Write Result

When result is available, write on CDB. From CDB to any reservation stations waiting for the result and also on ROB using the Tag available at reservation station. Free the reservation station. If the value of the store instruction available, it is written on the value field of ROB. If the value of the store instruction is not available, CDB is monitored to get the value when it is ready and then written on ROB.

Commit Step

Commit

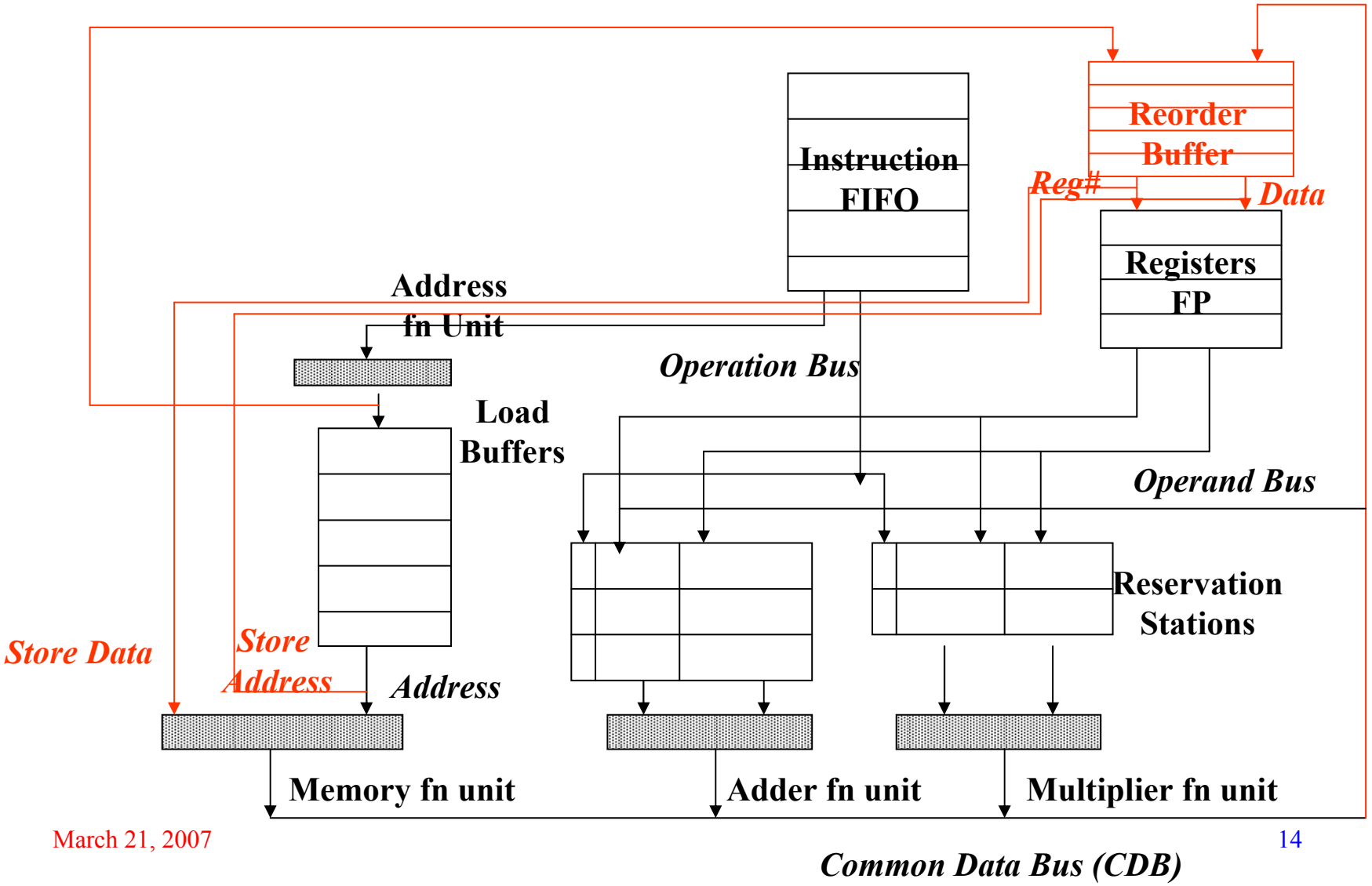
Three different sequences of actions

1. Normal Instruction: The instruction reaches head of ROB, result is present in the buffer. At this stage, commit updates the register with the results from buffer and frees the buffer.

2. Store Instruction: Similar to Normal, only result is updated in the Memory, in place of register.

3. Branch instruction with incorrect prediction: The ROB is flashed. The execution is restarted at the current successor of the branch.

Modified Tomasulo's Algorithm



Example: Modified Tomasulo

Iteration1:	L,D	F0,0(R1)	F0: array element
	MUL,D	F4,F0,F2	Add scalar in F2
	S,D	F4,0(R1)	store result
	DADDIU	R1,R1,-#8	decrement pointer 8-bytes
	BNE	R1,R2,Loop	branch R1!=R2 (R2 has the loop exit count)
Iteration2:	L,D	F0,0(R1)	
	MUL,D	F4,F0,F2	
	S,D	F4,0(R1)	
	DADDIU	R1,R1,-#8	
	BNE	R1,R2,Loop	

Assume: L.D and MUL.D in the first iteration have committed, and all other instructions have completed execution. The store would wait in the ROB for both effective address operand R1 and the value to be stored F4.

ROB Status

Entry No	Busy	Instruction	Status	Destination	Value
1	No	L.D F0,0(R1)	Commit	F0	Men[0+Reg[R1]]
2	No	MUL.D F4,F0,F2	Commit	F4	#1.Reg[F2]
3	Yes	S.D F4,0(R1)	Write Result	0+Reg[R1]	#2
4	Yes	DADDIU R1,R1,-#8	Write Result	R1	Reg[R1- 8]
5	Yes	BNE R1,R2,Loop	Write Result		
6	Yes	L.D F0,0(R1)	Write Result	F0	Men[#4]
7	Yes	MUL.D F4,F0,F2	Write Result	F4	#6.Reg[F2]
8	Yes	S.D F4,0(R1)	Write Result	0+ #4	#7
9	Yes	DADDIU R1,R1,-#8	Write Result	R1	#4 - 8
10	Yes	BNE R1,R2,Loop	Write Result		

FP Register Status				
Field	F0	F1…….	F4	F5 …….
Reorder#	6		7	
Busy	Yes	No	Yes	No

Design Consideration for Speculation

Register Renaming versus ROB

Architecturally Visible Register Set

Temporary Values in Tomasulo's algorithm: RS, ROB

Register values reside on Architecturally visible registers, RS and ROB

Extended set of Physical Registers for Register Renaming.

Renaming Process maps the physical registers to architecturally visible registers.

The ***speculation recovery*** by explicit commit function to make a register architectural register. ***Simplified Commit function***.

Reallocating Registers in renaming is complex process. In ROB, reallocation of ROB and RS are automatic.

Renaming is ***simplified issue function***, as only registers to be checked for the operand. In ROB, Registers, RS and ROBs are checked for issue.

Real architectural registers are ***continuously changing*** in Renaming

How much Speculation

Advantage of speculation

Ability to uncover events that would otherwise stall the pipeline.

Multiple branches speculation simultaneously required for:

- ***very high branch frequency***
- ***significant clustering of branches (DB management)***
- ***Long delays in functional unit.***

Disadvantages of speculation:

Processor may speculate a wrong costly exception function that will waste the valuable resources.

To minimize those penalties, processor only speculates the simple exception conditions like first order cache misses. The second level cache misses are not speculated.

Exception handling complexities.

Hardware Speculation Model

1. Register Renaming: Infinite number of virtual registers available to resolve all WAR and WAW Hazards.

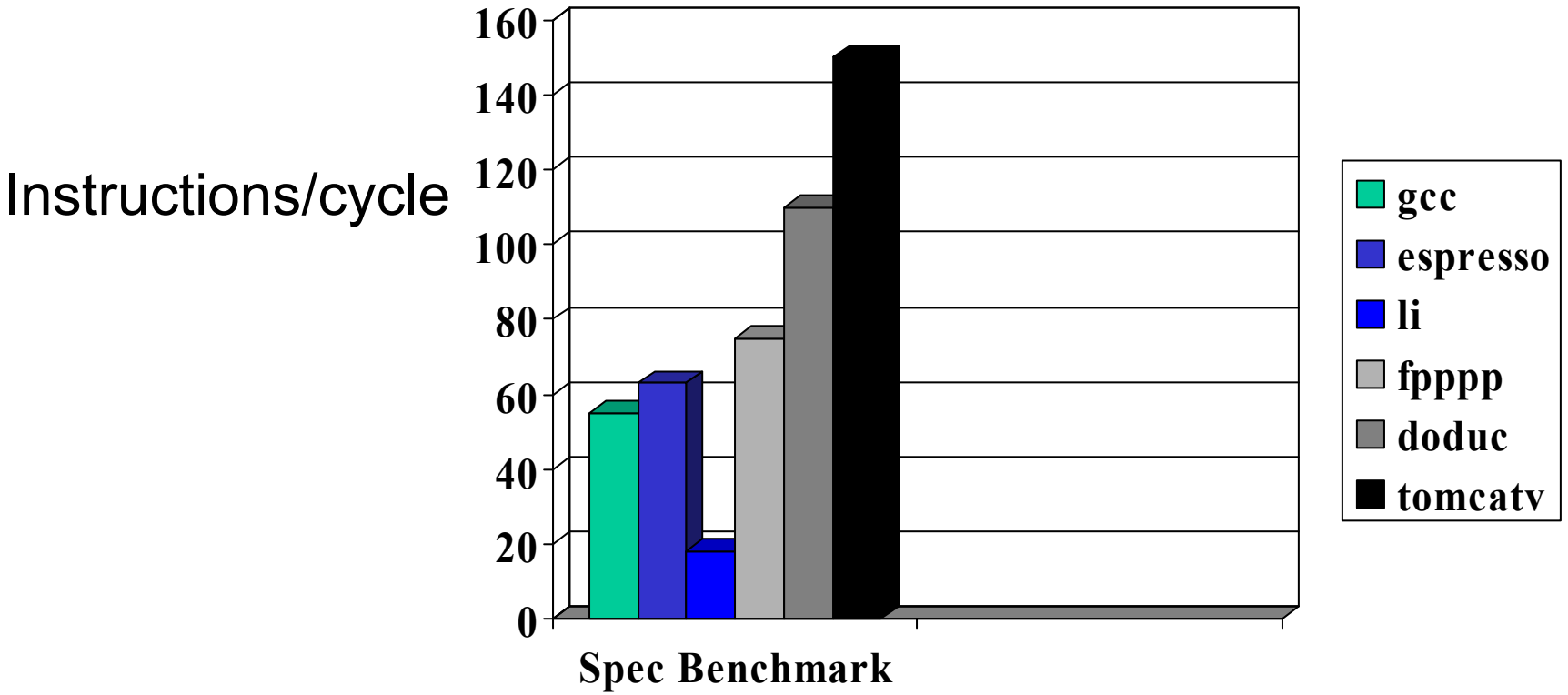
2. Branch Prediction is perfect

3. Jump prediction perfect

4. Memory address alias analysis: All memory address is known exactly and load can be moved before store provided that the addresses are not identical

5. Perfect caches: All load and store instructions are cache hit.

ILP in Perfect Processor



Window Size

Dynamic scheduled processor to reach to an ideal ILP, need to do the following:

1. Look arbitrarily far ahead to find a set of instructions to issue, predicting all branches perfectly.
2. Rename all register uses to avoid WAR and WAW
3. Identify all data dependencies in amongst the instructions and rename accordingly.
4. Identify all memory dependencies and remove them.
5. Provide sufficient functional unit to allow all ready instructions to issue.

Window Size

Window: The set of instructions that are examined simultaneously for issue is called Window

These actions will require large number of comparisons.

Assume n instructions are issued in a Window.

The total number of comparison needed

$$2\{(n-1) + (n-2) + \ldots\ldots\ldots +1\} = 2.{}^{n}C_{2} = n^{2} - n$$

Let n=50 Comparison = 2450

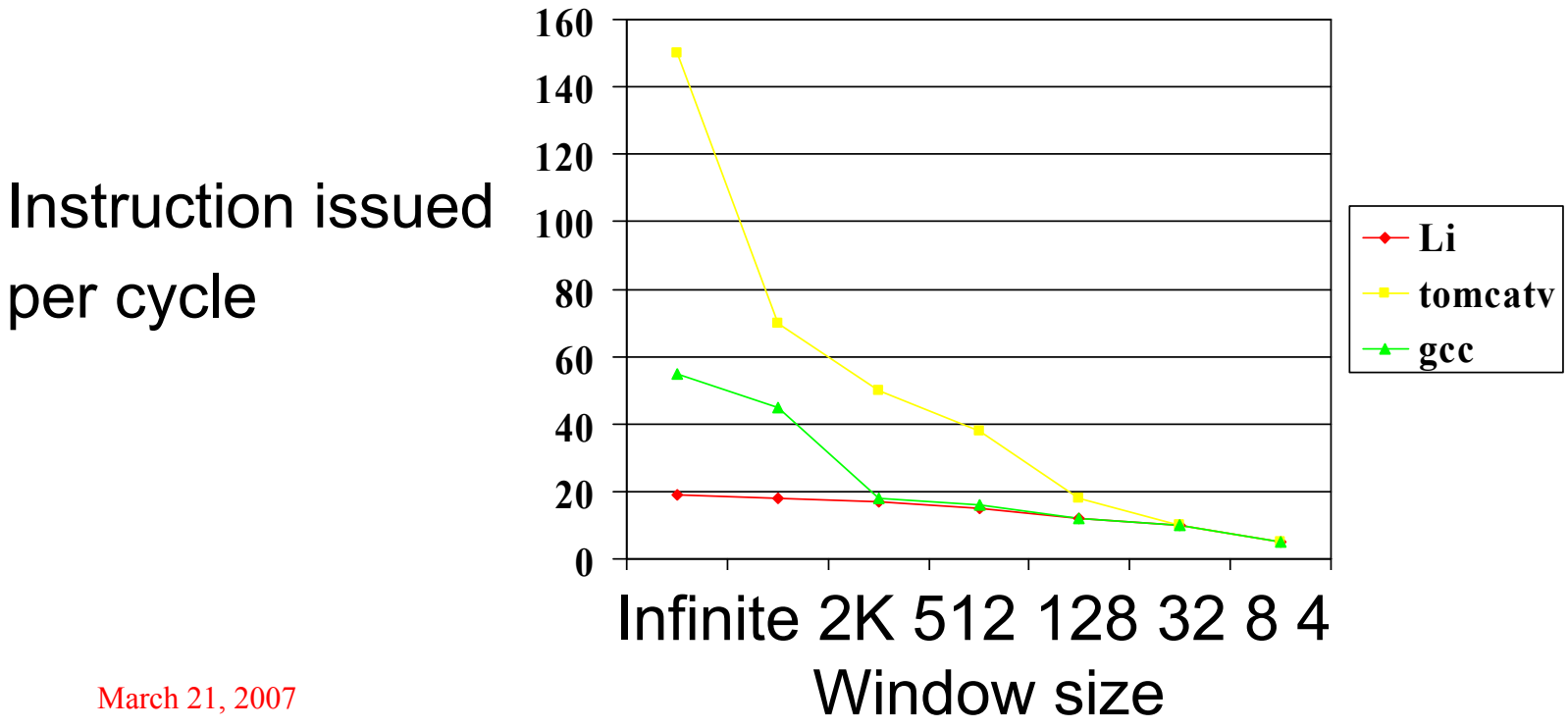
n= 2000 Comparison = 4.10^{6}

Window Size

Number of comparison required per clock cycle

= maximum completion rate X Window Size X number of operands per instruction.

Currently window size is limited to 32- 120

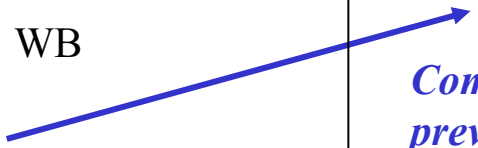


Static Super scalar in MIPS Processor

Assume

1. Two instructions per clock cycle
2. One Integer and one floating point instructions
3. Floating point Integer instructions have separate register files and functional units
4. Floating point instructions is only ADD that takes 3 cycles.
5. Issue restriction in case Integer instruction use FP registers in Load, Store or Move operation

Instruction type	Pipeline stages
Integer	IF ID EX MM WB
Floating Point	IF ID EX EX EX MM WB
Integer	IF ID EX MM WB
Floating Point	IF ID EX EX EX MM WB
Integer	IF ID EX MM WB
Floating point	IF ID EX EX EX MM WB



Completed before previous FP Instruction

Exception handling logic

Example

Loop:	L,D	F0,0(R1)	F0: array element
	ADD,D	F4,F0,F2	Add scalar in F2
	S,D	F4,0(R1)	store result
	DADDIU	R1,R1,-#8	decrement pointer 8-bytes
	BNE	R1,R2,Loop	branch R1!=R2

Both FP and Integer operation in one cycle, Single Branch Instruction per cycle
Assume MIPS pipeline extended with Tomasulo's algorithm
One Integer ALU for ALU operation and effective address calculation
Separate pipeline units for Integer and FP operations
Separate HW to evaluate branch condition, ***Two separate CDBs***
Dynamic branch prediction HW, No delayed branches, Branch prediction perfect

Cycles used for different functions

Issue	1
write results	1
Integer ALU operation	1
FP ALU operation	3

UNWINDING OF Loop

```
Iteration1:   L,D      F0,0(R1)      F0: array element
              ADD,D    F4,F0,F2      Add scalar in F2
              S,D      F4,0(R1)      store result
              DADDIU   R1,R1,-#8     decrement pointer 8-bytes
              BNE      R1,R2,Loop    branch R1!=R2 (R2 has the loop exit count)
Iteration2:   L,D      F0,0(R1)
              ADD,D    F4,F0,F2
              S,D      F4,0(R1)
              DADDIU   R1,R1,-#8
              BNE      R1,R2,Loop
Iteration3:   L,D      F0,0(R1)
              ADD,D    F4,F0,F2
              S,D      F4,0(R1)
              DADDIU   R1,R1,-#8
              BNE      R1,R2,Loop
```

Super scalar Pipelines

- Obviously, the two fetched instructions need to be independent in every way (no structural or other pipeline hazards between them).
- One of the best choices is to have one integer and one FP instruction scheduled to be fetched and issued to the pipe at the same time.

Instruction type	Pipe stages						
Integer instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	MEM	WB		
Integer instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB	
Integer instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB
Integer instruction				IF	ID	EX	MEM	WB
FP instruction				IF	ID	EX	MEM	WB

- Note that all hazard resolutions become more complicated in a super- scalar pipeline.

Time cycles

Iteration No	Instruction	Issue at	Executes at	Memory access at	Write CDB	Comments
1:L,D	F0,0(R1)	1	2	3	4	First Issue
ADD,D	F4,F0,F2	1	5		8	Wait for L,D
S,D	F4,0(R1)	2	3	9		Wait for ADD,D
DADDIU	R1,R1,-#8	2	4		5	Wait for Integer ALU
BNE	R1,R2,Loop	3	6			Wait for DADDIU
2:L,D	F0,0(R1)	4	7	8	9	Wait for BNE complete
ADD,D	F4,F0,F2	4	10		13	Wait for L,D
S,D	F4,0(R1)	5	8	14		Wait for ADD,D
DADDIU	R1,R1,-#8	5	9		10	Wait for Integer ALU
BNE	R1,R2,Loop	6	11			Wait for DADDIU
3:L,D	F0,0(R1)	7	12	13	14	Wait for BNE complete
ADD,D	F4,F0,F2	7	15		18	Wait for L,D
S,D	F4,0(R1)	8	13	19		Wait for ADD,D
DADDIU	R1,R1,-#8	8	14		15	Wait for Integer ALU
BNE	R1,R2,Loop	9	16			Wait for DADDIU

Resources Status

Clock Cycle	Integer ALU	FP ALU	Data Cache	CDB
2	1/LD			
3	1/SD		1/LD	
4	1/DADDIU			1/LD
5		1/ADDD		1/DADDIU
6				
7	2/LD			
8	2/SD		2/LD	1/ADDD
9	2/DADDIU		1/SD	2/LD
10		2/ADDD		2/DADDIU
11				
12	3/LD			
13	3/SD		3/LD	2/ADDD
14	3/DADDIU		2/SD	3/LD
15		3/ADDD		3/DADDIU
16				
17				
18				3/ADDD
19			3/SD	

Time cycles for separate Integer ALU

Iteration No	Instruction	Issue at	Executes at	Memory access at	Write CDB	Comments
1:L,D	F0,0(R1)	1	2	3	4	First Issue
ADD,D	F4,F0,F2	1	5		8	Wait for L,D
S,D	F4,0(R1)	2	3	9		Wait for ADD,D
DADDIU	R1,R1,-#8	2	3		4	Integer ALU Free
BNE	R1,R2,Loop	3	5			Wait for DADDIU
2:L,D	F0,0(R1)	4	6	7	8	***Wait for BNE complete***
ADD,D	F4,F0,F2	4	9		12	Wait for L,D
S,D	F4,0(R1)	5	7	13		Wait for ADD,D
DADDIU	R1,R1,-#8	5	6		7	Integer ALU Free
BNE	R1,R2,Loop	6	8			Wait for DADDIU
3:L,D	F0,0(R1)	7	9	10	11	***Wait for BNE complete***
ADD,D	F4,F0,F2	7	12		15	Wait for L,D
S,D	F4,0(R1)	8	10	16		Wait for ADD,D
DADDIU	R1,R1,-#8	8	9		10	Integer ALU Free
BNE	R1,R2,Loop	9	11			Wait for DADDIU

Resources Status with separate Integer ALU

Clock Cycle	Integer ALU	Address ALU	FP ALU	Data Cache	CDB#1	CDB#2
2		1/LD				
3	1/DADDIU	1/SD		1/LD		
4					1/LD	1/DADDIU
5			1/ADDD			
6	2/DADDIU	2/LD				
7		2/SD		2/LD	2/DADDIU	
8					1/ADDD	2/LD
9	3/DADDIU	3/LD	2/ADDD	1/SD		
10		3/SD		3/LD	3/DADDIU	
11					3/LD	
12			3/ADDD		2/ADDD	
13				2/SD		
14						
15						3/ADDD
16				3/SD		

Super scalar Pipelines

	Integer instruction		FP instruction	Clock cycle
Loop:	LD	F0,0(R1)		1
	LD	F6,-8(R1)		2
	LD	F10,-16(R1)	ADDD F4,F0,F2	3
	LD	F14,-24(R1)	ADDD F8,F6,F2	4
	LD	F18,-32(R1)	ADDD F12,F10,F2	5
	SD	0(R1),F4	ADDD F16,F14,F2	6
	SD	-8(R1),F8	ADDD F20,F18,F2	7
	SD	-16(R1),F12		8
	SUBI	R1,R1,#40		9
	SD	16 (R1), F16		10
	BNEZ	R1,LOOP		11
	SD	8(R1),F20		12

- 12 cycles per 5 iterations 2.4 cycles per iteration!

The End of Pipelining!

Super scalar Pipelines

- Consider our previous loop example:

```
Loop:   LD     F0, 0(R1)      ; F0=array element
        ADD    F4, F0, F2     ; add scalar in F2
        SD     0(R1), F4      ; store result
        SUBI   R1, R1, #8     ; decrement pointer
                              ; 8 bytes (per DW)
        BNEZ   R1, Loop       ; branch R1 != zero
```

- We were able to unroll this loop 4 times and schedule the instructions so that each loop iteration could complete every 3.5 clock cycles.
- Can we do better with super scaling?
  - Need to unroll 5 times
  - Pair integer and FP operations