3.4 Hardware-based Speculation & Multiple Issue

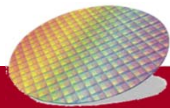
成功大學

Hardware-Based Speculation

- In Dynamic Scheduling with branch prediction
  - Instructions are fetched and issued
  - Requires a branch to be resolved by executing any instructions (the branch is determined in ID stage to reduce stalls)
  - Only partially overlap basic block => Less ILP improvement

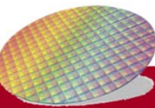
Branch Prediction

Branch inst. i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	nop	nop	nop	nop			
Branch Target			IF	ID	EX	MEM	WB		
Branch Target+1				IF	ID	EX	MEM	WB	
Branch Target+2					IF	ID	EX	MEM	WB



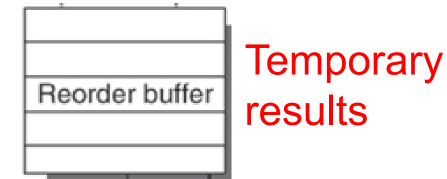
Hardware-Based Speculation

- Hardware-Based Speculation:
  - Fetch & issue & Execution instruction as if our branch prediction were always correct
  - Only commit the results if prediction was correct
  - Need to handle situation where the speculation is incorrect
- Attempt to schedule instructions in different basic blocks
  - Overcome control dependence caused by branch
- Three major concepts:
  - Dynamic branch prediction to choose which instructions to execute
  - Speculation to allow the execution of instructions before the control dependences are resolved
  - Dynamic scheduling to deal the scheduling of different combinations of basic blocks

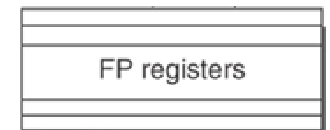


Reorder Buffer and Instruction commit Stage

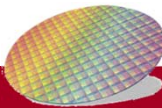
- To execute an instruction speculatively, need to separate temporary results from the actual results
- Add "Reorder buffer" hardware and one pipeline stage "instruction commit"



- Reorder buffer: to hold results of instruction that have finished execution (temp. results) but have not committed
  - Prevent any irrevocable action until an instruction commits
  - Allow an instruction to execute and bypass its results to other instructions without performing any update that cannot be undone
  - Until we know that the instruction is no longer speculative



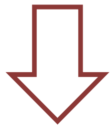
- Instruction commit: allowing an instruction to update the register file when instruction is no longer speculative
- Instruction can execute out-of-order, but commit in order



Tomasulo algorithm with speculation

- Compared to Tomasulo's algorithm without speculation in Figure 3.6, major differences is
  - Reorder buffer
  - No Store buffer (store data are from reorder buffer)

Three Steps in Tomasulo algorithm: Issue, Execute, Write Result



Four Steps in Tomasulo algorithm with speculation: Issue, Execute, Write Result, Commit

Reorder buffer
From instruction unit
Reg #
Data
Instruction queue
FP registers
Load/store operations
Operand buses
Address unit
Floating-point operations
Load buffers
Operation bus
Store address
3
2
1
Reservation stations
2
1
Store data
Address
Memory unit
FP adders
FP multipliers
Load data
Common data bus (CDB)

成功大學

Reorder Buffer

- Reorder buffer – holds the result of instruction between completion and commit
  - On misprediction: Speculated entries in ROB are cleared
- Four fields:
  - Ready field: completed execution?
  - Instruction type: branch/store/register
  - Destination field: supply the register number or the memory address where the instruction result should be written
  - Value field: hold the output value

Reorder buffer

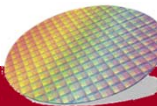
Ready (Busy)	Instruction	Destination	Value

- Modification in Reservation Stations:
  - Operand source is now reorder buffer instead of functional unit

Reservation Stations

Reservation Stations			S1	S2	RS for j	RS for k		
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No						#3	
Add3	No				#1			

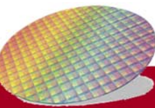
Refer to Reorder buffer entries



成功大學

Four Steps in instruction execution

- Issue: get an instruction from instruction queue. Issue the instruction when there is an empty RS and empty slot in ROB
- Execute:
  - If operands are not available, monitor the CDB while waiting for the register to be computed.
  - If operands are available, execute the instruction
- Write Result:
  - when results is available, write result on the CDB, from CDB into ROB as well as any RS waiting for the results
  - Mark the reservation station as available
- Commit: final stage of completing an instruction, three different cases:
  - Committing instruction is a branch with an incorrect prediction
  - A store
  - Normal commit



Tomasulo's Algorithm with Speculation Example

- Given the following codes, show what the table look like when the MUL.D is ready to go commit

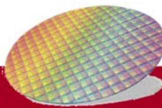
Entry	Busy	Instruction		State	Dest.	Value
1		L.D	F6, 32(R2)			
2		LD	F2, 44(R3)			
3		MULTD	F0, F2, F4			
4		SUBD	F8, F2, F6			
5		DIVD	F10, F0, F6			
6		ADDD	F6, F8, F2			

```
L.D      F6, 32(R2)
L.D      F2, 44(R3)
MUL.D    F0, F2, F4
SUB.D    F8, F2, F6
DIV.D    F10, F0, F6
ADD.D    F6, F8, F2
```

Reservation Stations				S1	S2	RS for j	RS for k		
Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
0	Load1	No							
0	Load2	No							
0	Add1	No							
0	Add2	No							
0	Add3	No							
0	Mult1	No							
0	Mult2	Yes							

add: 2 clock
multiply: 6 clock cycles
divide is 12 clock.

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Busy										
Reorder#										



L.D F6, 32(R2)
L.D F2, 44(R3)
MUL.D F0, F2, F4
SUB.D F8, F2, F6
DIV.D F10, F0, F6
ADD.D F6, F8, F2

Table status when MUL.D is ready to commit

Reorder Buffer

Entry	Busy	Instruction		State	Dest.	Value
1	No	L.D	F6, 32(R2)	Commit	F6	Mem[32+Regs[R2]]
2	No	LD	F2, 44(R3)	Commit	F2	Mem[44+Regs[R3]]
3	Yes	MULTD	F0, F2, F4	Write Result	F0	#2 * Regs[F4]
4	Yes	SUBD	F8, F2, F6	Write Result	F8	#2 - #1
5	Yes	DIVD	F10, F0, F6	Execute	F10	
6	Yes	ADDD	F6, F8, F2	Write Result	F6	#4+#2

Reservation Stations

Reservation Stations				S1	S2	RS for j	RS for k		
Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
0	Load1	No							
0	Load2	No							
0	Add1	No							
0	Add2	No							
0	Add3	No							
0	Mult1	No	MUL.D	Mem[44+Regs[R3]]	Regs[F4]			#3	
0	Mult2	Yes	DIV.D		Mem[32+Regs[R2]]	#3		#5	

Register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Busy	Yes	No	No	No	No	No	Yes	..	Yes	Yes
Reorder#	3						6		4	5

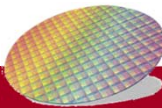


Table status when MUL.D is ready to commit

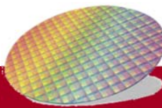
Although SUBD has completed execution, it does not commit until MUL.D commits

RS numbers are replaced with Reorder buffer entry numbers

Entry	Busy	Instruction		State	Dest.	Value
1	No	L.D	F6, 32(R2)	Commit	F6	Mem[32+Regs[R2]]
2	No	LD	F2, 44(R3)	Commit	F2	Mem[44+Regs[R3]]
3	Yes	MULTD	F0, F2, F4	Write Result	F0	#2xRegs[F4]
4	Yes	SUBD	F8, F2, F6	Write Result	F8	#2-#1
5	Yes	DIVD	F10, F0, F6	Execute	F10	
6	Yes	ADDD	F6, F8, F2	Write Result	F6	#4+#2

Reservation Stations				S1	S2	RS for j	RS for k		
Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
0	Load1	No							
0	Load2	No							
0	Add1	No							
0	Add2	No							
0	Add3	No							
0	Mult1	No	MUL.D	Mem[44+Regs[R3]]	Regs[F4]			#3	
0	Mult2	Yes	DIV.D		Mem[32+Regs[R2]]	#3		#5	

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Busy	Yes	No	No	No	No	No	Yes	..	Yes	Yes
Reorder#	3						6		4	5



Example for Tomasulo's Algorithm with Speculation

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

Assume all instructions are issued twice. Assume L.D and MUL.D from the first iteration have committed and other instructions have completed execution.

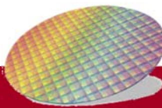
Given the above codes, show what the table look like when the MUL.D is ready to go commit

Suppose that the branch BNE is not taken, the instructions prior to the branch will commit when each inst. reaches the head of ROB

Entry	Busy	Instruction		State	Dest.	Value
1	No	L.D	F0, 0(R1)	Commit	F0	Mem[0+Regs[R1]]
2	No	MUL.D	F4, F0, F2	Commit	F4	#1xRegs[F2]
3	Yes	S.D	F4, 0(R1)	Write Result	0+Regs[R1]	#2
4	Yes	DADDIU	R1, R1, #-8	Write Result	R1	Regs[R1]-8
5	Yes	BNE	R1,R2, Loop	Write Result		
6	Yes	L.D	F0, 0(R1)	Write Result	F0	Mem[#4]
7	Yes	MUL.D	F4, F0, F2	Write Result	F4	#6xRegs[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		

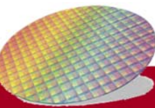
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder#	6				7					
Busy	Yes	No	No	No	Yes	No	No	No	No	No

Processor can easily undo its speculative actions: when branch reaches the head of ROB, the buffer is simply cleared and the processor begins fetching instructions from the other path



Summary of Hardware-based Speculation

- Reorder Buffer to store temporary results
  - Reorder Buffer hold results of instructions that have finished execution, but have not committed
  - The Register file is not updated until the instruction commit,
  - Reorder Buffer supplies operand in the interval between completion of instruction executing and instruction commit
- Four Pipeline steps
  - Issue
  - Execute
  - Write Result
  - Commit
- Speculation to allow the execution of instruction before the control dependence are resolved (with the ability to undo the effects of incorrectly speculated sequence)
- Support Precise Exception

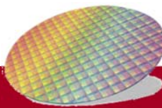


Multiple-issue Processor

- Single-issue processor has CPI >=1 (or IPC <=1)
- Multiple-issue processor: issue multiple instructions per clock
  - Can achieve CPI < 1 (or IPC > 1)
- Five different types
  - Statically scheduled superscalar processors (SSSP)
  - Dynamically scheduled superscalar processors without speculation
  - Dynamically scheduled superscalar processors with speculation
  - VLIW (very long instruction word) processors: We will introduce it briefly
  - EPIC: improved version of VLIW with aggressive software speculation. Will not be explained (See Appendix H)

SSSP, VLIW are both based on compilers and are similar
=> only VLIW is introduced

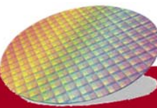
DSSP w/o speculation and w/ speculation are hardware-based techniques



Multiple issue static scheduling (VLIW) Processors

- VLIW(Very Long Instruction word): Package multiple operations into one instruction
- Example: An VLIW processor with five operations
  - One integer instruction (or branch)
  - Two independent floating-point operations
  - Two independent memory references
- Need a field for each instruction

- Instructions must have enough parallelism to fill the available slots
  - One technique is loop unrolling
  - Using instructions across branches



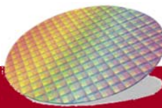
Recall: Loop Unrolling/Pipeline Scheduling

- Pipeline schedule the unrolled loop:

```
Loop:   L.D F0,0(R1)
        L.D F6,-8(R1)
        L.D F10,-16(R1)
        L.D F14,-24(R1)
        ADD.D F4,F0,F2
        ADD.D F8,F6,F2
        ADD.D F12,F10,F2
        ADD.D F16,F14,F2
        S.D F4,0(R1)
        S.D F8,-8(R1)
        S.D F12,16(R1)
        S.D F16,8(R1)
        DADDUI R1,R1,#-32
        stall
        BNE R1,R2,Loop
```

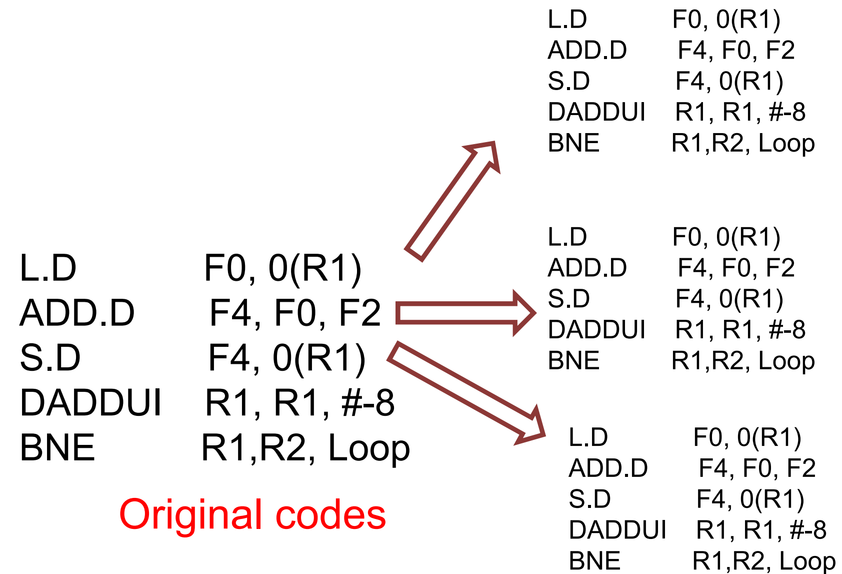
14 Cycles to generate 4 results

14/4 = 3.5 cycles for each results

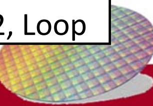


VLIW Example

- Consider an VLIW instruction contains five operations
  - One integer instruction (or branch)
  - Two independent floating-point operations
  - Two independent memory references
- Show an unrolled version of the loop x[i] = x[i] +s
- Results when loop has been unrolled 7 times

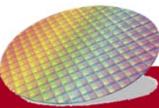


Memory reference 1	Memory Reference 2	FP Operations 1	FP Operations 2	Integer operation/branch
L.D F0, 0(R1)	L.D F6, -8(R1)			
L.D F10, -16(R1)	L.D F14, -24(R1)			
L.D F18, -32(R1)	L.D F22, -40(R1)	ADD.D F4, F0, F2	ADD.D F8, F6, F2	
L.D F26, -48(R1)		ADD.D F12, F0, F2	ADD.D F16, F14, F2	
		ADD.D F20, F18, F2	ADD.D F24, F22, F2	
S.D F4, 0(R1)	S.D F8, -8(R1)	ADD.D F428, F26, F2		
S.D F12, -16(R1)	S.D F16, -24(R1)			DADDUI R1, R1, #-56
S.D F20, 24(R1)	S.D F24, 16(R1)			
S.D F28, 8(R1)				BNE R1,R2, Loop



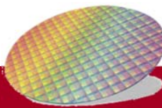
Remarks on VLIW Example

- Results when loop has been unrolled seven times
- All stalls are eliminated (no completely empty issue cycles)
- Only need 9 cycles to generate 7 results
- Cycle per results = 9 / 7 = 1.26 (Compared to Section 3.2)
- VLIW Issues 1: Increase in code size because unrolling codes and waste bits
  - Solution1: Using clever encoding
  - Solution2: Compress in main memory and decompress whenever they are used
- VLIW Issues 2: Binary code compatibility
  - Code sequences make use of detailed pipeline structure including functional units and latency.
  - May need recompilation
- VLIW Issues 3: Statically finding parallelism
  - Be conservative

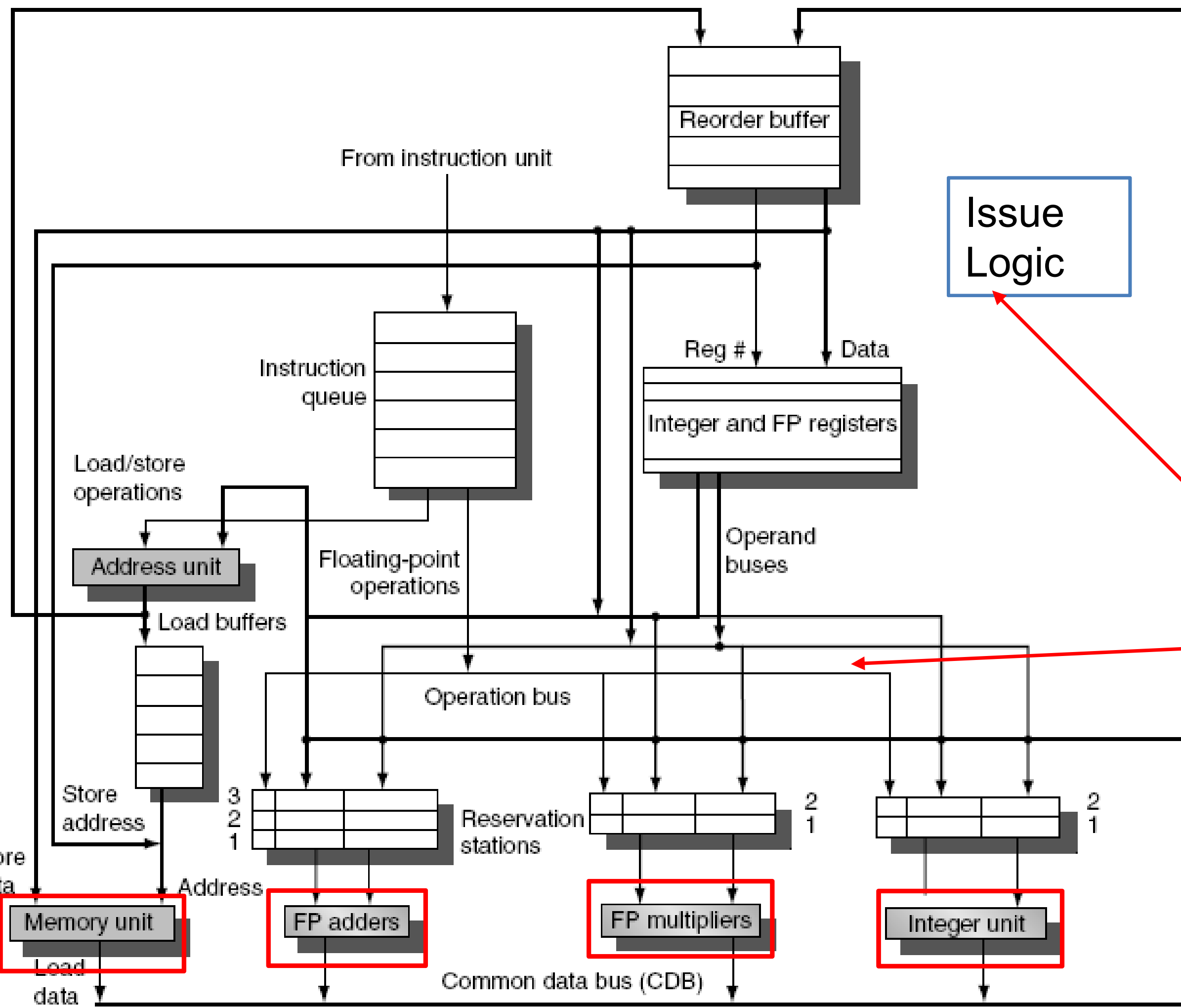


Dynamic Scheduling, Multiple Issue, and Speculation

- Modern microarchitectures:
  - Dynamic scheduling + multiple issue + speculation
- To simplify: only consider 2-issue
- Extend Tomasulo's algorithm to support multiple-issue superscalar pipeline (See next page)
  - Separate integer, load/store, and FP units (each can initiate an operation on every cycle)
  - Allow pipeline to issue any combination of two instructions in a clock
- Issue logic can become bottleneck
  - Issue multiple instructions per clock in a dynamically scheduled processor is very complexed
    - Multiple instruction may depend on one another
  - Table must be updated in parallel
    - otherwise, tables will be incorrect or dependence may be lost

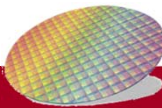


Multiple Issue Dynamic Scheduling Processor with Speculation



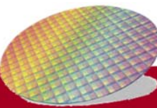
- FP multiply, FP add, integer and load/store instructions can be issued simultaneously
- Issue logic is not shown. But it is much complicated that single-issue processor.
- The datapath of CDB, operand buses, instruction issue logic must be widen to support multiple issues.

Wider datapath



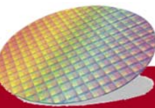
Multiple Issue

- Two approaches to issue multiple instructions per cycle
  - Key issue is assigning reservation stations and update pipeline control table
- Method1: Assign reservation stations in first half cycle and update pipeline control in the second half cycles
  - Limitation: Only supports 2 instructions/clock
- Method 2: Design logic to handle two or more instructions at once
  - Support more instructions/clock
  - Much more complicated
- Modern computer uses hybrid approaches:
  - pipeline and widen the issue logic
- Difficult to go beyond issuing 4 instructions/clock



Steps for multiple Issue

- Choose instructions of a given class that can be issued together in a bundle and reserve reservation stations and reorder buffers
  - i.e. 4 instruction (one FP, one integer, one load, one store)
  - The number of instructions has limit (larger number results in more complex circuits)
- Examine all the dependencies among the instructions in a bundle
- If an instruction depends on an earlier instruction (dependencies exist in two bundles), encode them in reservation stations
- Also need multiple completion/commit



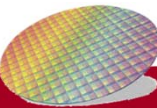
Example

Consider the execution of the following loop, which increment each element of an integer array, on a two-issue processor

1. without speculation
2. with speculation

Create a table for the first three iterations of this loop for both processors. Assume that up to 2 instructions of any type can commit per clock

```
Loop:   LD R2,0(R1)          ;R2=array element
        DADDIU R2,R2,#1      ;increment R2
        SD R2,0(R1)          ;store result
        DADDIU R1,R1,#8      ;increment pointer
        BNE R2,R3,LOOP       ;branch if not last element
```



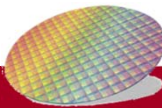
Example (Without Speculation)

Issue multiple instructions in a cycle

LD.D following BNE cannot start execution earlier because it must wait until the branch outcome is determined

Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD R2,0(R1)	1	2	3	4	First issue
1	DADDIU R2,R2,#1	1	5		6	Wait for LW
1	SD R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	Execute directly
1	BNE R2,R3,LOOP	3	7			Wait for DADDIU
2	LD R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU R2,R2,#1	4	11		12	Wait for LW
2	SD R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU R1,R1,#8	5	8		9	Wait for BNE
2	BNE R2,R3,LOOP	6	13			Wait for DADDIU
3	LD R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU R2,R2,#1	7	17		18	Wait for LW
3	SD R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU R1,R1,#8	8	14		15	Wait for BNE
3	BNE R2,R3,LOOP	9	19			Wait for DADDIU

Need 19 cycles

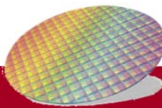


Example (with speculation)

Iteration number	Instructions		Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD	R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LW
1	SD	R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	8	Commit in order
1	BNE	R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU	R2,R2,#1	4	8		9	10	Wait for LW
2	SD	R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU	R1,R1,#8	5	6		7	11	Commit in order
2	BNE	R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD	R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU	R2,R2,#1	7	11		12	13	Wait for LW
3	SD	R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU	R1,R1,#8	8	9		10	14	Executes earlier
3	BNE	R2,R3,LOOP	9	13			14	Wait for DADDIU

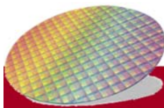
LD.D following BNE can start execution earlier because it is speculative

Need 13 cycles



Multiple Issue Processor Comparison

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic various number of instruction per cycle	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static fixed number of inst. per cycle	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium



Backup slides