ECSE 425 Lecture 17: Multiple-Issue Processors

H&P Chapter 2

Last Time

- Limitations of Dynamic Scheduling
- Dynamic Scheduling with Hardware Speculation
- New pipeline stage, new hardware structure

Today

- Multiple-issue processors
  - Issue more than one instruction per cycle
  - Ideal CPI < 1

Issuing Multiple Instructions/Cycle

- Basic idea: parallel pipelines
  - Fetch, issue, and complete multiple instructions per cycle
- *Superscalar*
  - Schedule instructions so when possible, more than one can issue at the same time
  - Static (compiler) or dynamic (hardware) scheduling is possible
- *Very Long Instruction Words* (VLIW)
  - Instructions are grouped into fixed-width sets
  - Sets are (generally) scheduled by the compiler
  - Architecture determines the width of the set

Multiple-Issue Processors

Common Name	Issue Structure	Hazard detection	Scheduling	Distinguishing Characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	MIPS, ARM (mainly embedded)
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution (no speculation)	None presently
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel P4, MIPS R12K, IBM Power5
VLIW	Static	Mostly Software	Static	All hazards determined and indicated by compiler (often implicitly)	C6X
EPIC	Mostly static	Mostly software	Mostly static	Explicit dependences marked by compiler	Itanium

Static Superscalar (SS) Processor

- Superscalar MIPS: 2 instructions, 1 FP & 1 integer
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

Type	*Pipe*	*Stages*				
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB
Int. instruction		IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 cycle load delay affects 3 instructions in SS
  - Second instruction can't use result, nor those in next slot

Remember the Unrolled Loop …

Add a scalar to a vector

```
1  L0:  L.D      F0,0(R1)
2       L.D      F6,-8(R1)
3       L.D      F10,-16(R1)
4       L.D      F14,-24(R1)
5       ADD.D    F4,F0,F2
6       ADD.D    F8,F6,F2
7       ADD.D    F12,F10,F2
8       ADD.D    F16,F14,F2
9       S.D      F4,0(R1)
10      S.D      F8,-8(R1)
11      DADDUI   R1,R1,#-32
12      S.D      F12,16(R1); 16-32 = -16
13      BNE      R1,R2,L0
14      S.D      F16,8(R1)  ; 8-32 = -24
```

14 clock cycles, or 3.5 per iteration

Now with Static Superscalar Issue

- Unroll the loop five times instead of four
- When we can issue one INT and one FP operation, performance improves 46%

	Integer instruction		FP instruction	
Loop	L.D	F0,0(R1)		
	L.D	F6,-8(R1)		
	L.D	F10,-16(R1)	ADD.D	F4,F0,F2
	L.D	F14,-24(R1)	ADD.D	F8,F6,F2
	L.D	F18,-32(R1)	ADD.D	F12,F10,F2
	S.D	F4,0(R1)	ADD.D	F16,F14,F2
	S.D	F8,-8(R1)	ADD.D	F20,F18,F2
	S.D	F12,-16(R1)		
	DADDUI	R1,R1,#-40		
	S.D	F16,16(R1)		
	BNE	R1,R2,Loop		
	S.D	F20,8(R1)		

2.4 cycles per iteration

5 FP registers

Hazards in *N*-Wide Static Superscalar

- Issue packet: a group of instructions from IF that could potentially issue simultaneously
- If an instruction causes a hazard (structural or data) due to
  - An earlier instruction in execution or
  - An earlier instruction in the issue packet,
- Then instruction is not issued
- As a result, 0 to *N* instructions issued per clock cycle, for *N*-wide issue

Performance in Static Superscalar

- Issue checks could limit clock cycle time
  - Many comparisons required
  - Issue stage usually split and pipelined
    - 1st stage examines hazards within the packet
    - 2nd stage examines hazards between the packet and executing instructions
  - Longer pipeline => higher branch penalties => greater accuracy prediction needed
- Integer/FP split uses simple HW, hard to 0.5 CPI
  - Programs need exactly 50% FP ops AND no hazards

VLIW

- Static superscalar processors
  - Decide how many instructions to issue on-the-fly
  - HW checks for dependencies
  - Mostly limit to 2-issue
- VLIW checks dependencies in the compiler
  - Construct an instruction packet with either no dependencies or indicate if they are present
  - Simpler hardware
  - Re-compilation often required when HW changes

The VLIW Idea

- Multiple, independent functional units
- Compiler finds independent operations, packages them together in a very long instruction word
  - Eliminates expensive issue hardware in a superscalar
- VLIW machines tend to have issue widths > 4
  - Static superscalar processors are especially expensive for wide issue widths

VLIW Example

- 5-wide VLIW
  - 1 integer operation (incl. branch) unit
  - 2 FP op. units
  - 2 memory ref. units
- Code must have enough parallelism to fill the operation slots and keep the FUs busy
  - Loop unrolling and code scheduling
- Local scheduling: straight-line codes
- Global scheduling: across branches, substantially more complex (see *trace scheduling*)

Unrolled loop example for VLIW

Mem Ref 1	Mem Ref 2	FP op1	FP op2	Int. op/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,-24(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-32(R1)		ADD.D F12,F10,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 F28,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

- Unroll the loop seven rather than five times
  - 23 operations in 9 clock cycles, or 2.5 operations / cycle
  - 1.3 cycles per iteration, 83% faster than 2-wide SS!
- Low efficiency
  - Instructions in 60% of available slots
  - Large number of registers used, too

Issues with VLIW

- Increased code size
  - Aggressively unroll loops, waste bits when packets are not full
  - Can use clever encoding or compression to reduce code size
- Lock-step operation
  - No hazard detection in hardware
  - A stall in one FU stalls the whole processor
  - Can't predict cache misses: slows down all other instructions
  - Recent processors relax this with extra HW
- Binary code compatibility
  - Different pipelines, different code (e.g., more or different FUs)
  - Object code translation (Crusoe: rapidly developing)
  - Another solution: relax this approach (IA-64)

Dynamic Superscalar with Speculation

- Now put everything together
  - Multiple issue
  - Dynamic scheduling
  - Speculation

Unrolled Loop Example Yet Again

Multiple Issue with Speculation

No speculation:

Iter.	Instruction		Issue	Exec	Mem	Write CDB	Comment
1	LD	R2,0(R1)	1	2	3	4	
1	DADDIU	R2,R2,#1	1	5		6	Wait for LD
1	SD	R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU	R1,R1,-8	2	3		4	Executes directly
1	BNE	R1,R2,L	3	7			Wait for DADDIU
2	LD	R2,0(R1)	4	8	9		Wait for BNE
2	DADDIU	R2,R2,#1	4	11		13	Etc.
2	SD	R2,0(R1)	5	9	13		
2	DADDIU	R1,R1,-8	5	8		9	
2	BNE	R1,R2,L	6	13			

Speculation:

Iter.	Instruction		Issue	Exec	Read Acces	Write CDB	Commits	Comment
1	LD	R2,0(R1)	1	2	3	4	5	
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LD
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	R1,R2,L	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No delay
2	DADDIU	R2,R2,#1	4	8		9	10	Etc.
2	SD	R2,0(R1)	5	6			10	
2	DADDIU	R1,R1,-8	5	6		7	11	
2	BNE	R1,R2,L	6	10			11	

Summary

- Superscalar execution
  - Issue and complete multiple instructions per cycle
- Static Superscalar
  - Instructions issued in-order once hazards clear
- VLIW
  - Generally wider than superscalar processors
  - Multiple instructions packed by compiler for simultaneous issue; no HW hazard detection
- Dynamic Superscalar
  - Multiple issue, dynamic scheduling, speculation

Next Time

- Advanced Techniques
  - Chapter 2.9
  - Instruction delivery
  - Speculation