EITF20: Computer Architecture

Part3.2.1: Pipeline - 3

Liang Liu
liang.liu@eit.lth.se

Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- ILP limitations
- What we have done so far (pipeline)

Instruction level parallelism (ILP)

- **ILP: Overlap execution of unrelated instructions: Pipelining**
- **Pipeline CPI = Ideal CPI + Structural stalls + Data hazard stalls + Control stalls**
- **Two main approaches:**
  - DYNAMIC ⇒ hardware detects parallelism
  - STATIC ⇒ software detects parallelism
  - Often a mix between both

Dependencies

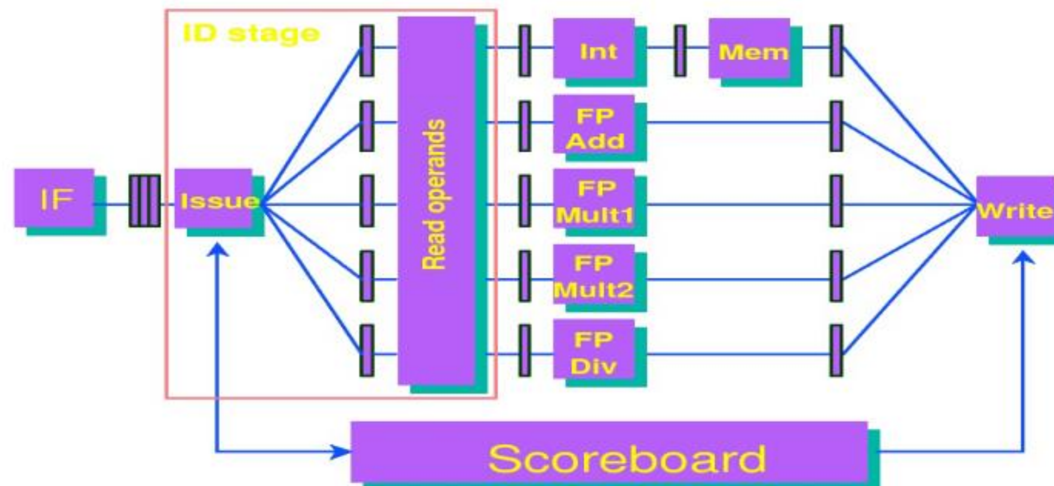
- **Two instructions must be independent in order to execute in parallel**
- **There are three general types of dependencies that limit parallelism:**
  - Data dependencies (RAW)
  - Name dependencies (WAR, WAW)
  - Control dependencies
- **Dependencies are properties of the program**
- **Whether a dependency leads to a hazard or not is a property of the pipeline implementation**

Dynamic branch prediction

- **Branches limit performance because:**
  - Branch penalties
  - Limit to available Instruction Level Parallelism
- **Dynamic branch prediction to predict the outcome of conditional branches**
  - Branch history table
  - Branch target buffer
- **Benefits:**
  - Reduce the time to when the branch condition is known
  - Reduce the time to calculate the branch target address

Scoreboard pipeline

- **Scoreboarding** is to execute an instruction as early as possible
- **Instructions execute out-of-order** when there are sufficient resources and no data dependencies
- **A scoreboard is a hardware unit that keeps track of**
  - the **instructions** that are in the process of being executed
  - the **functional units** that are doing the executing
  - and the **registers** that will hold the results of those units
- **A scoreboard centrally performs** all hazard detection and instruction control



Scoreboard functionality

- **Issue:** An instruction is issued if:
  - The needed functional unit is free (there is no **structural hazard**)
  - No functional unit has a destination operand equal to the destination of the instruction (resolves **WAW hazards**)
- **Read:** Wait until no data hazards, then read operands
  - Performed in parallel for all functional units
  - Resolves **RAW hazards** dynamically
- **EX:** Normal execution
  - Notify the scoreboard when ready
- **Write:** The instruction can update destination if:
  - All earlier instructions have read their operands (resolves **WAR hazards**)

Factors that limits performance

The scoreboard technique is limited by:

- **The amount of parallelism available in code**
- **The number of scoreboard entries (window size)**
  - A large window can look ahead across more instructions
- **The number and types of functional units**
  - Contention for functional units leads to structural hazards
- **The presence of anti-dependencies and output dependencies**
  - These lead to WAR and WAW hazards that are handled by stalling the instruction in the Scoreboard
- **Number of data-paths to registers**

Tomasulo's algorithm addresses the last two limitations.

Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- ILP limitations
- What we have done so far

Tomasulo algorithm

Another dynamic instruction scheduling algorithm

- For IBM 360/91, a few years after the CDC 6600 (Scoreboard)
- Goal: High performance without compiler support

R. M. Tomasulo

An Efficient Algorithm for Exploiting Multiple Arithmetic Units

Abstract: This paper describes the methods employed in the floating-point area of the System/360 Model 91 to exploit the existence of multiple execution units. Basic to these techniques is a simple common data busing and register tagging scheme which permits simultaneous execution of independent instructions while preserving the essential precedences inherent in the instruction stream. The common data bus improves performance by efficiently utilizing the execution units without requiring specially optimized code. Instead, the hardware, by 'looking ahead' about eight instructions, automatically optimizes the program execution on a local basis.

The application of these techniques is not limited to floating-point arithmetic or System/360 architecture. It may be used in almost any computer having multiple execution units and one or more 'accumulators.' Both of the execution units, as well as the associated storage buffers, multiple accumulators and input/output buses, are extensively checked.

Registr renaming

Example:

```
LD      F6, 34(R2)
...     ...
DIVD    F10,F0,F6
ADDD    F6,F8,F2
```

```
LD      FT, 34(R2)
...
DIVD    F10, F0, FT
ADDD    F6, F8, F2
```

```
DIVD    F10, F0, F6
ADDD    FT, F8, F2
...
SD      FT, 35(R3)
```

- **Potential WAR harzard on F6**
  - If ADDD finishes before DIVD starts
- **Register renaming**
  - Another temperoray register (FT) instead of F6
  - Any subsequent uses of F6 should be replaced by FT (until the next wirte to F6)
- **Register renaming can be done:**
  - statically by the compiler
  - dynamically by the hardware

Tomasulo orgnizations

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

Reservation stations

- Op:Operation to perform (e.g., + or –)
- Vj, Vk: Value (instead of reg specifier) of Source operands
- Qj, Qk: Reservation stations (instead of FU) producing source registers (value to be written)
  - Note: Qj,Qk=0 => ready
  - V and Q filed are mutual exclusive
- Busy: Indicates reservation station or FU is busy
- Register result status—Indicates which RS will write each register
  - Blank when no pending instructions that will write that register

Functional unit status				src 1	src 2	RS for j	RS for k
Time	Name	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Three stages of Tomasulo algorithm

- **Issue –** get instruction from instruction Queue
  - If matching reservation station free (no **structural hazard**)
  - Instruction is issued together with its operands values or RS point (register rename, handle **WAR, WAW**)
- **Execution –** operate on operands (EX)
  - When both operands are ready, then execute (handle **RAW**)
  - If not ready, watch **Common Data Bus (CDB)** for operands (snooping)
- **Write result –** finish execution (WB)
  - Write on CDB to all awaiting RS, regs (**forwarding**)
  - Mark reservation station available
  - Data Bus
    - Normal Bus: data + destination
    - Common Data Bus: data + source (snooping)

Tomasulo example, cycle 0

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Load buffers

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU								

Clock: 0

Tomasulo example, cycle 1

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Load buffers

Time		Busy	Address
	Load1	Yes	R2+34
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Load1				

Clock: 1

Tomasulo example, cycle 3

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Load buffers

Time		Busy	Address
0	Load1	Yes	R2+32
1	Load2	Yes	R3+45
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Mult		F4	Load2	
	Mult2	No					

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Load2		Load1				

Clock: 3

Note:

1. Can have multiple loads
2. Registers names are removed ("renamed") in Reservation Stations

Tomasulo example, cycle 4

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Load buffers

Time		Busy	Address
	Load1	No	
0	Load2	Yes	R3+45
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	Yes	Sub	M(R2+34)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	Mult		F4	Load2	
	Mult2	No					

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Load2		-	Add1			

Clock: 4

Tomasulo example, cycle 5

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
2	Add1	Yes	Sub	M(R2+34)	M(R3+45)		-
	Add2	No					
	Add3	No					
10	Mult1	Yes	Mult	M(R3+45)	F4	-	
	Mult2	Yes	Div		F6	Mult1	

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	-			Add1	Mult2		

Clock: 5

Tomasulo example, cycle 7

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
0	Add1	Yes	Sub	M(R2+34)	M(R3+45)		
	Add2	Yes	Add		F2	Add1	
	Add3	No					
8	Mult1	Yes	Mult	M(R3+45)	F4		
	Mult2	Yes	Div		F6	Mult1	

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add2	Add1	Mult2		

Clock: 7

Tomasulo example, cycle 10

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
0	Add2	Yes	Add	F6-F2	F2		
	Add3	No					
5	Mult1	Yes	Mult	M(R3+45)	F4		
	Mult2	Yes	Div		F6	Mult1	

Can we write the result of ADDD?

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add2		Mult2		

Clock: 10

Elimation of WAR hazard

```
Example:  LD      F6, 34(R2)
          ...     ...
          DIVD    F10,F0,F6
          ADDD    F6,F8,F2
```

- **ADDD can safely finish before DIVD has read register F6 because:**
  - DIVD has renamed register F6 to the reservation station
  - LD broadcasts its result on the Common Data Bus

Tomasulo example, cycle 11

Instruction status

Instruction	j	k		Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Mult	M(R3+45)	F4		
	Mult2	Yes	Div		F6	Mult1	

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			-		Mult2		

Clock: 11

Tomasulo example, cycle 16

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	Div	F0	F6	-	

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	-					Mult2		

Clock: 16

Tomasulo example, cycle 57

Instruction status

Instruction		j	k	Issue	Exec. compl.	Write result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

Load buffers

Time		Busy	Address
	Load1	No	
	Load2	No	
	Load3	No	

Functional unit status

Time	Name	Busy	Op	src 1 Vj	src 2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU						-		

Clock: 57

Comparing: Scoreboard example, CP62

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU						-		

Clock: 62

Tomasulo vs Scoreboard

Differences between Tomasulo Algorithm and Scoreboard

- **Control and buffers distributed with Function Units versus centralized in scoreboard**
- **Registers in instructions replaced by pointers to reservation stations**
  - Register renaming, helps avoid WAR and WAW hazards
  - More reservation stations than registers; so allow optzns compilers can't do
  - Operands stays in register in Scoreboard (stall for WAR and WAW)
- **Common Data Bus broadcasts results to all FUs (forwarding!)**

Tomasulo summary

- **Instructions:** move from decoder to reservation stations
  - In program order
  - Dependences can be correctly recorded
  - Distributed harzard detection
- **Significant increase in HW cost**
- **Benefits**
  - Register renaming, remove WAW, WAR hazard
  - Out-of-order execution, completion
  - Tolerates unpredictable delays (especially for cache)
  - Compile for one pipeline - run effectively on another

Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- **Superscalar, VLIW**
- Speculation
- ILP limitations
- What we have done so far

Getting CPI <1 !?

Issuing multiple (independent) instructions per clock cycle

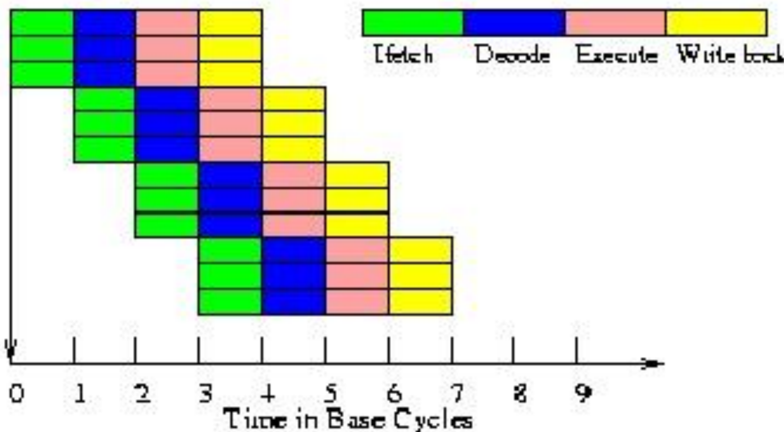
- **Superscalar: varying number of instructions/cycle (1-8) scheduled by compiler or HW**
  - IBM Power5, Pentium 4, Sun SuperSparc, DEC Alpha
  - Simple hardware, complicated compiler (static) or...
  - Very complex hardware but simple for compiler (dynamic)
- **Very Long Instruction Word (VLIW): fixed number of instructions (3-5) scheduled by the compiler**
  - HP/Intel IA-64, Itanium
  - Simple hardware, difficult for compiler
  - High performance through extensive compiler optimization

Approches for multiple issuing

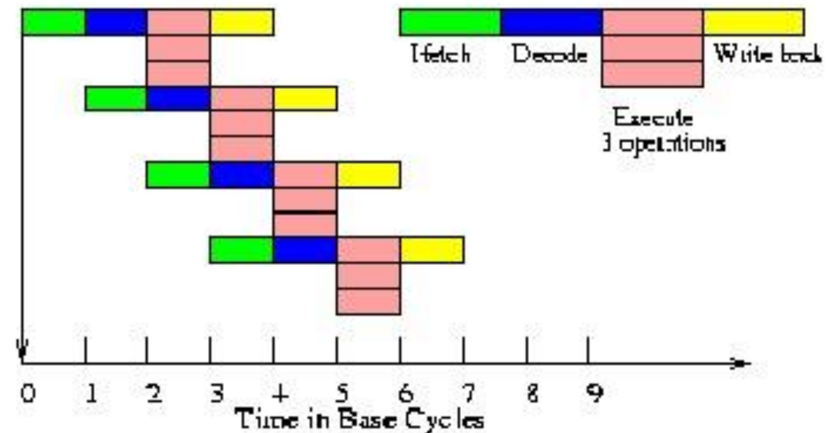
	Issue	Hazard detection	Scheduling	Characteristics /examples
Superscalar	dynamic	HW	static	in-order execution ARM
Superscalar	dynamic	HW	dynamic	out-of-order execution
Superscalar	dynamic	HW	dynamic	speculation Pentium 4 IBM power5
VLIW	static	compiler	static	TI C6x
EPIC	static	compiler	mostly static	Itanium

VLIW

- A number of functional units that independently execute instructions in parallel.
- The compiler decides which instructions can execute in parallel
- No hazard detection needed

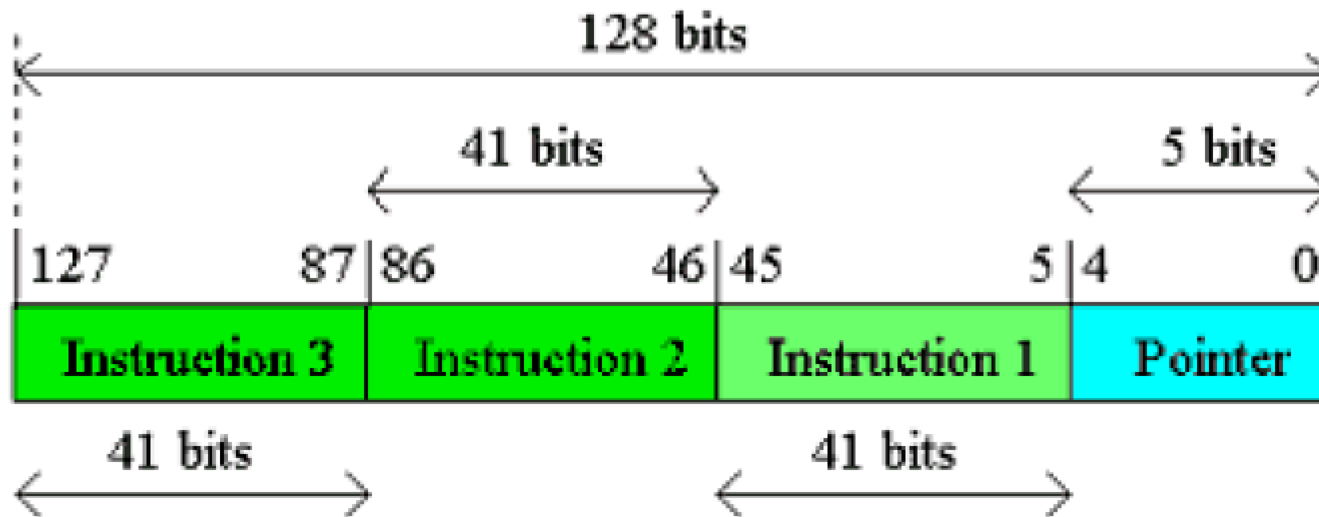


A Superscalar Processor of Degree 3



VLIW Execution with Degree 3

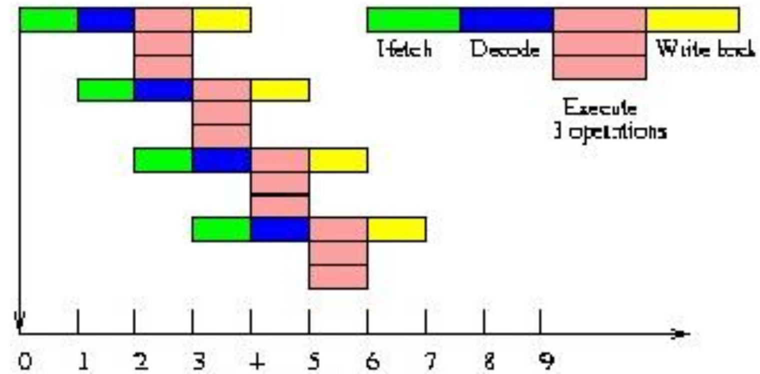
VLIW instuction format - Itanium



VLIW architecture- Itanium

L1 Instruction Cache 16 KByte
(4-way, 32 Byte / Cache Line)
PC
Fetch Engine
Branch Predictors
256
256
256
L2 Cache
(96 Kbyte
6-way, 64 Byte /
Cache Line)
IA-32
Decode,
Control
8 Bundles
6 Instructions
9 Issue Ports
B B B
M M I I
F F
Branch Units 0 1 2
Integer & MM Units 0 1 2 3
Floating Point Units 0 1
SIMD FMAC
Data Frontside Bus
(2 x 133 MHz)
64
Addr.
L3 Cache- and Systembus Control
1
64
82
256
64 Branch & Predicate Registers
128 Integer Registers
128 Floating Point Registers
Register Stack Engine and Re-Mapping
64 64 Addr. Addr. 128 128
128
Backside Bus
from / to L3 Cache
L1 Data Cache 16 KByte
(Dual Ported, 4-way, 32 Byte / Cache Line)
ALAT

VLIW limitations



Limited Instruction Level Parallelism

- With *n* functional units and *k* pipeline stages we need *n* * *k* independent instructions to utilize the hardware

Memory and register bandwidth

- With increasing number of functional units, the number of ports needed at the memory or register file must increase to prevent structural hazards

Code size

- Compiler scheduled pipeline “bubbles” take up space in the instruction
- Need more aggressive loop unrolling to work well which also increases code size

No binary code compatibility

- Different pipeline stages and # function units requires different code

Synchronization

- Stall in one FU cause the entire processor to stall

Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- **Speculation**
- ILP limitations
- What we have done so far

Hardware-base speculation

- **Trying to exploit more ILP (e.g., multiple issue) while maintaining control dependencies becomes a burden**
- **Overcome control dependencies**
  - By speculating on the outcome of branches and **executing** the program as if our guesses were correct
  - Need to handle incorrect guesses
- **Methodologies:**
  - **Dynamic branch prediction**: allows instruction scheduling across branches (no exe)
  - **Dynamic scheduling**: take advantage of ILP (wait for clear branch)
  - **Speculation**: **execute** instructions before all control dependencies are resolved (basically data flow)

Hardware vs software speculation

- **Advantages:**
  - Dynamic branch prediction is often better than static which limits the performance of SW speculation
  - HW speculation can maintain a **precise exception** model
  - Can achieve higher performance on older code (**without recompilation**)
- **Main disadvantage:**
  - Extremely complex implementation and extensive need for hardware resources

Implementing speculation

Key idea

- Allow instructions to execute **out of order**
- Force instructions to **commit in order**
- Prevent any **irrevocable action** (such as updating state or taking an exception) until an instruction commits

Strategies:

- Must separate **bypassing of results** among instructions from **actual completion** (write-back) of instructions
- Instruction commit updates register or memory when instruction **no longer speculative**

Need to add re-order buffer

- Hold the results of inst. that have finished exe but have not commiteed

Tomasulo extended to support speculation

Reorder buffer

From instruction unit

Reg #

Data

FP registers

Instruction queue

Load-store operations

Address unit

Floating-point operations

Operand buses

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)

ROB (reorder buffer)

entry	instruction type	destination	value	ready
1				
2				
...				
n				

- **Contains 4 fields:**
  - **Instruction type** indicates whether branch, store, or register op
  - **Destination field** memory or register
  - **Value field** hold the inst. result until commit
  - **Ready flag** indicates instruction has completed operation
- **Every instruction has a ROB entry until it commits**
  - Therefore tag results using ROB entry number
  - The renaming function of the reservation stations is partially replaced by the ROB

ROB (reorder buffer)

- When MUL.D is ready to commit

```
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
```

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	No	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	Yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	Yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	Yes	DIV.D	F10,F0,F6	Execute	F10	
6	Yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	Yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

Four-step speculation

- **Issue:**
  - Get instruction from instruction queue and issue if reservation station and **ROB slots** available – sometimes called dispatch
  - Send operands or **ROB entry #**
- **Execution – operate on operands (EX)**
  - If both operands ready: execute; if not, watch CDB for result;
  - When both operands are in reservation station: execute
- **Write result – complete execution**
  - Write on CDB to all awaiting **FUs (RSs) & ROB** (tagged by ROB entry #)
  - Mark reservation station available
- **Commit – update register with reorder result**
  - When instr. **is at head of ROB & result is present & no longer speculative**; update register with result (or store to memory) and remove instr. from ROB;
  - handle mis-speculations and precise exceptions

Four-step speculation

- **Commit – branch prediction wrong**
  - When branch instr. is at head of ROB & incorrect prediction (or exception): remove all instr. from reorder buffer (flush); restart execution at correct instruction
  - Expensive ⇒ try to recover as early as possible (delay in ROB)
  - Performance sensitive to branch prediction/speculation (waste computation power & time if wrong)

Sandy bridge microarchitecture

Instruction Fetch Unit

Branch Predictors

144 Entry L1 ITLB

32KB L1 I-Cache (8 way)

16 Bytes

16B Predecode, Fetch Buffer

6 instructions

18+ Entry Instruction Queue

µcode Engine

Complex Decode

Simple Decode

Simple Decode

Simple Decode

1.5K µop Cache (8 way)

4 µops

4 µops

1 µop

1 µop

1 µop

4 µops

28 µop Decoder Queue

4 µops

168 Entry Reorder Buffer (ROB)

4 µops

144 Entry FP Physical Register File

160 Entry Physical Register File

4 µops

54 Entry Unified Scheduler

Port 0

Port 1

Port 5

Port 2

Port 3

Port 4

ALU LEA Shift

SIMD MUL Shift

ALU LEA MUL

ALU Shift Branch

SIMD ALU Shuffle

64-bit AGU

64-bit AGU

Store Data

256-bit FMUL Blend

256-bit FADD

SIMD ALU Shuffle

256-bit Shuffle Blend

128 bits

64 Entry Load Buffer

36 Entry Store Buffer

128 bits

128 bits

128 bits

512 Entry L2 TLB (4 way)

100 Entry L1 DTLB

32KB L1 D-Cache (8 way)

256 bits

256KB L2 Cache (8 way)

Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- **ILP limitations**
- What we have done so far

How much performance can be get from instruction-level parallelism?

A model of an idea processor

- **Assumptions for ideal/perfect machine to start:**
  - ***Register renaming***: infinite virtual registers => all register WAW & WAR hazards are avoided
  - ***Branch prediction***: perfect; no mispredictions
  - ***Jump prediction***: all jumps perfectly predicted
  - ***2 & 3*** => machine with perfect speculation & an unbounded buffer of instructions available
  - ***Memory***: addresses are known, perfect caches;
- **There are only true data dependencies left!**

Upper limit to ILP (# of instruction per cycle)

Programs	ILP
gcc	54.8
espresso	62.6
li	17.9
fpppp	75.2
doducd	118.7
tomcatv	150.1

Window size impact

Instruction issues per cycle by benchmark and window size:

Benchmarks	Infinite	2K	512	128	32
gcc	55	36	10	10	8
espresso	63	41	15	13	8
li	18	15	12	11	9
fpppp	75	61	49	35	14
doduc	119	59	16	15	9
tomcatv	150	60	45	34	14

Window size: Infinite, 2K, 512, 128, 32

X-axis: Instruction issues per cycle (0, 20, 40, 60, 80, 100, 120, 140, 160)

Branch impact

Branch predictor: Perfect, Tournament predictor, Standard 2-bit, Profile-based, None

Benchmarks	Perfect	Tournament predictor	Standard 2-bit	Profile-based	None
gcc	35	9	6	6	2
espresso	41	12	7	6	2
li	16	10	6	7	2
fpppp	61	48	46	45	29
doduc	58	15	13	14	4
tomcatv	60	46	45	45	19

Instruction issues per cycle

Register # impact

Renaming registers — Instruction issues per cycle (Benchmarks)

Benchmark	Infinite	256 integer + 256 FP	128 integer + 128 FP	64 integer + 64 FP	32 integer + 32 FP	None
gcc	11	10	10	9	5	4
espresso	15	15	13	10	5	4
li	12	12	12	11	6	5
fpppp	59	49	35	20	5	4
doduc	29	16	15	11	5	5
tomcatv	54	45	44	28	7	5

Instruction issues per cycle: 0, 10, 20, 30, 40, 50, 60

Summary

Software (*compiler*) tricks:

- Loop unrolling
- Static instruction scheduling (with *register renaming*)
- ... and more

Hardware tricks:

- Dynamic instruction scheduling
- Dynamic branch prediction
- Multiple issue – Superscalar, VLIW
- Speculative execution
- ... and more

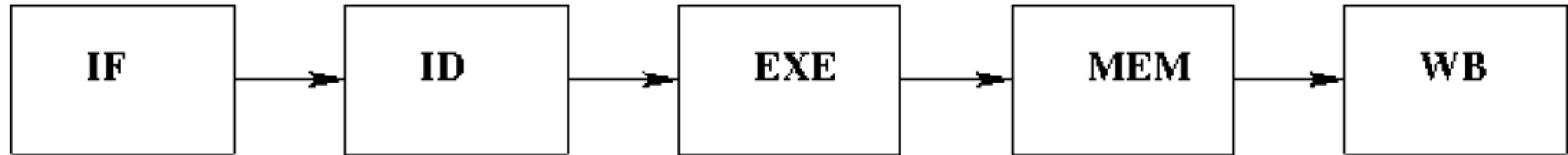
Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- ILP limitations
- **What we have done so far**

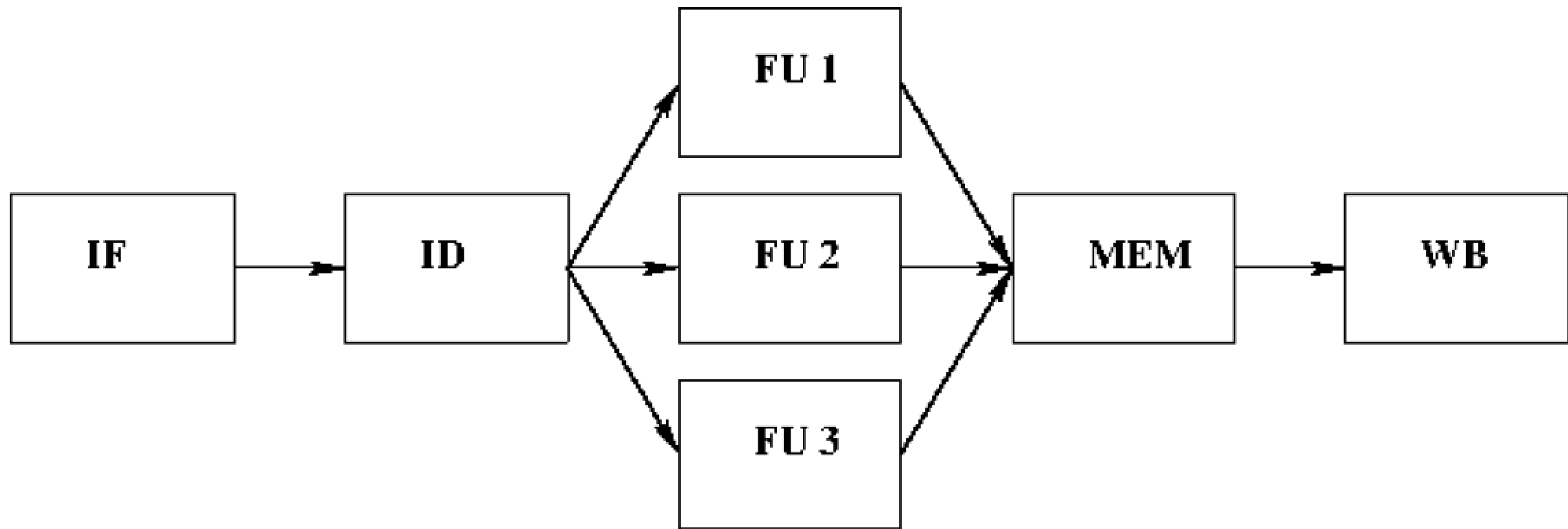
Summary pipeline - method

Dependency	Hazard	Method
Data	RAW	Forwarding, Scheduling,
Name	WAR, WAW	Register Renaming
Control	Control	Branch Prediction, Speculation, Delayed branch
Precise exceptions		in-order commit
ILP		Scheduling, Loop unrolling

Basic 5-stage pipeline



Pipeline with several FUs



Scoreboard pipeline

IF
ISSUE
READ 1
READ 2
READ 3
FU 1
FU 2
FU 3
WRITE
Scoreboard

Tomasulo pipeline

IF

ISSUE

RS 1

RS 2

RS 3

FU 1

FU 2

FU 3

WRITE

Common Data Bus

Tomasulo pipeline with speculation

ReOrder Buffer

IF

ISSUE

RS 1

RS 2

RS 3

FU 1

FU 2

FU 3

WRITE

COMMIT

Common Data Bus

Summary pipeline - implementation

Problem	Simple	Scoreboard	Tomasulo	Tomasulo + Speculation
	Static Sch	Dynamic Scheduling		
RAW				
WAR				
WAW				
Exceptions				
Issue				
Execution				
Completion				
Structural hazard				
Control hazard				