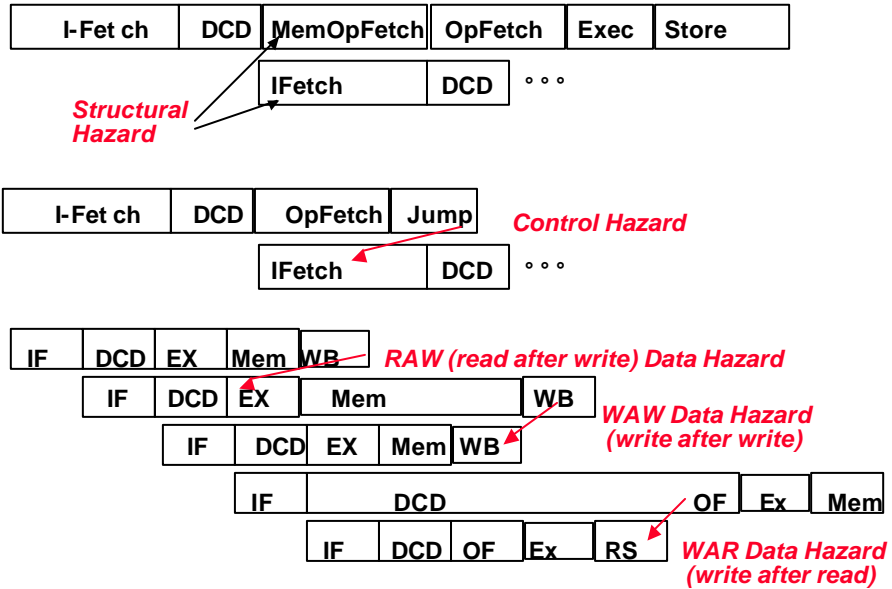

Tik-61.123 Computer Architecture Lecture 9: Introduction to Pipelining 3

Review: Summary of Pipelining Basics

- Pipelines pass control information down the pipe just as data moves down pipe
- Forwarding/Stalls handled by local control
- Hazards limit performance
 - Structural: need more HW resources
 - Data: need forwarding, compiler scheduling
 - Control: early evaluation & PC, delayed branch, prediction
- Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency
- Interrupts, Instruction Set, FP makes pipelining harder
- Compilers reduce cost of data and control hazards
 - Load delay slots
 - Branch delay slots
 - Branch prediction

Recap: Pipeline Hazards

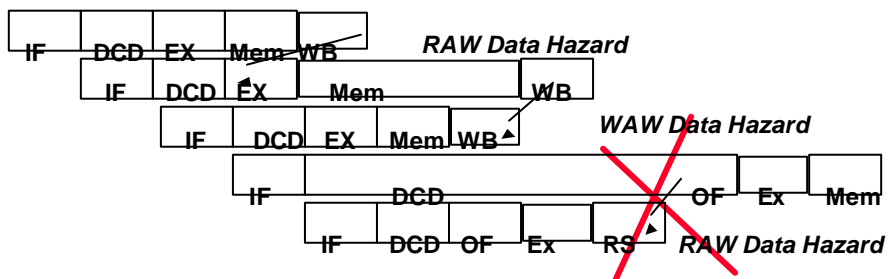


cs 152 L1 5 3

DAP Fa97, © UCB

Recap: Data Hazards

- Avoid some “by design”
 - eliminate WAR by always fetching operands early (DCD) in pipe
 - eliminate WAW by doing all WBs in order (last stage, static)
- Detect and resolve remaining ones
 - stall or forward (if possible)



cs 152 L1 5 4

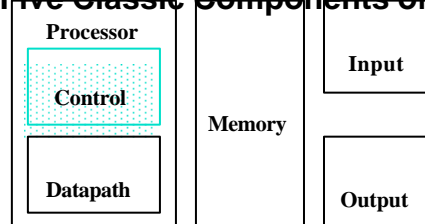
DAP Fa97, © UCB

Recap: Exception Problem

- **Exceptions/Interrupts:** 5 instructions executing in 5 stage pipeline
 - How to stop the pipeline?
 - Restart?
 - Who caused the interrupt?
- Stage** *Problem interrupts occurring*
- IF** Page fault on instruction fetch; misaligned memory access; memory-protection violation
- ID** Undefined or illegal opcode
- EX** Arithmetic exception
- MEM** Page fault on data fetch; misaligned memory access; memory-protection violation; memory error
- Load with data page fault, Add with instruction page fault?
- **Solution 1:** interrupt vector/instruction 2: interrupt ASAP, restart everything incomplete

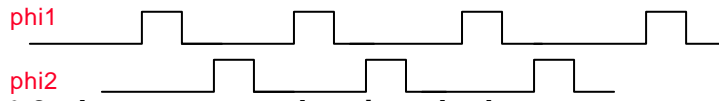
The Big Picture: Where are We Now?

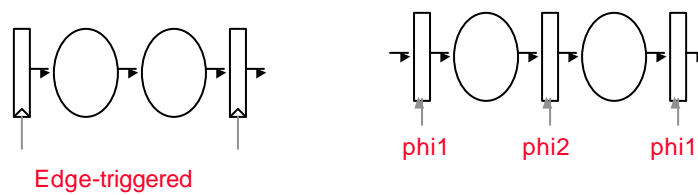
- **The Five Classic Components of a Computer**



- **Today's Topics:**
 - Recap last lecture
 - Review MIPS R3000 pipeline
 - Administrivia
 - Advanced Pipelining
 - SuperScalar, VLIW/EPIC

FYI: MIPS R3000 clocking discipline

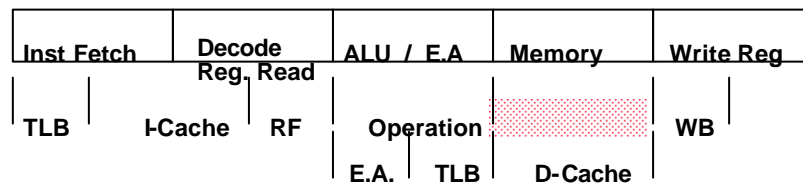
- phi1
- 
- phi2
- 2-phase non-overlapping clocks
 - Pipeline stage is two (level sensitive) latches



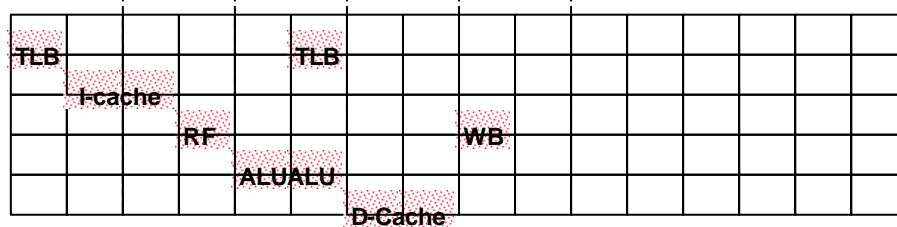
cs 152 L1 5 7

DAP Fa97, © UCB

MIPS R3000 Instruction Pipeline



Resource Usage

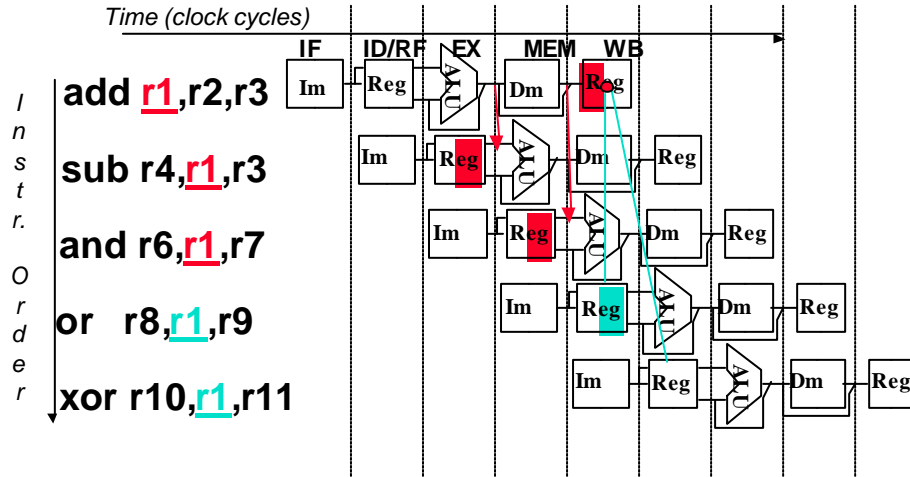


Write in phase 1, read in phase 2 => eliminates bypass from WB

cs 152 L1 5 8

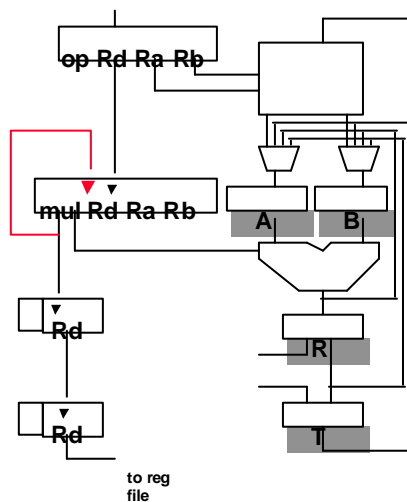
DAP Fa97, © UCB

Recall: Data Hazard on r1



With MIPS R3000 pipeline, no need to forward from WB stage

MIPS R3000 Multicycle Operations



Ex: Multiply, Divide, Cache Miss

Stall all stages above multicycle operation in the pipeline

Drain (bubble) stages below it

Use control word of local stage state to step through multicycle operation

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- **Two main variations: Superscalar and VLIW**
- **Superscalar: varying no. instructions/cycle (1 to 6)**
 - Parallelism and dependencies determined/resolved by HW
 - IBM PowerPC 604, Sun UltraSparc, DEC Alpha 21164, HP 7100
- **Very Long Instruction Words (VLIW): fixed number of instructions (16) parallelism determined by compiler**
 - Pipeline is exposed; compiler must schedule delays to get right result
- **Explicit Parallel Instruction Computer (EPIC)/ Intel**
 - 128 bit packets containing 3 instructions (can execute sequentially)
 - Can link 128 bit packets together to allow more parallelism
 - Compiler determines parallelism, HW checks dependencies and forwards/stalls

cs 152 L1 5.11

DAP Fa97, © UCB

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- **Superscalar DLX: 2 instructions, 1 FP & 1 anything else**
 - 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

<i>Type</i>	<i>PipeStages</i>						
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 expands to 3 instructions in SS**
 - instruction in right half can't use it, nor instructions in next slot

cs 152 L1 5.12

DAP Fa97, © UCB

Unrolled Loop that Minimizes Stalls for Scalar

1	Loop: LD	F0,0(R1)	
2	LD	F6,-8(R1)	LD to ADDD: 1 Cycle
3	LD	F10,-16(R1)	ADDD to SD: 2 Cycles
4	LD	F14,-24(R1)	
5	ADDD	F4,F0,F2	
6	ADDD	F8,F6,F2	
7	ADDD	F12,F10,F2	
8	ADDD	F16,F14,F2	
9	SD	0(R1),F4	
10	SD	-8(R1),F8	
11	SD	-16(R1),F12	
12	SUBI	R1,R1,#32	
13	BNEZ	R1,LOOP	
14	SD	8(R1),F16 ; 8-32 = -24	

14 clock cycles, or 3.5 per iteration

Loop Unrolling in Superscalar

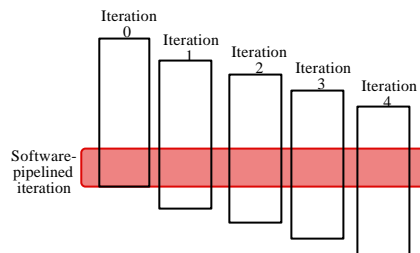
	<i>Integer instruction</i>	<i>FP instruction</i>	<i>Clock cycle</i>
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16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

◦ **Unrolled 5 times to avoid delays (+1 due to SS)**

◦ **12 clocks, or 2.4 clocks per iteration**

Software Pipelining

- **Observation:** if iterations from loops are independent, then can get ILP by taking instructions from different iterations
- **Software pipelining:** reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop (- Tomasulo in SW)



cs 152 L1 5.15

DAP Fa97, © UCB

Software Pipelining Example

Before: Unrolled 3 times

```

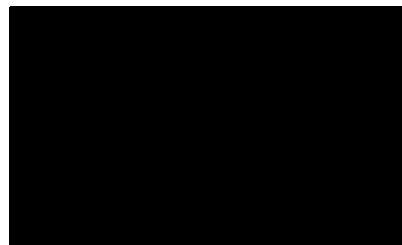
1 LD F0,0(R1)
2 ADDD F4,F0,F2
3 SD 0(R1),F4
4 LD F6,-8(R1)
5 ADDD F8,F6,F2
6 SD -8(R1),F8
7 LD F10,-16(R1)
8 ADDD F12,F10,F2
9 SD -16(R1),F12
10 SUBI R1,R1,#24
11 BNEZ R1,LOOP
    
```

After: Software Pipelined

```

1 SD 0(R1),F4 ; Stores M[i]
2 ADDD F4,F0,F2 ; Adds to M[i-1]
3 LD F0,-16(R1); Loads M[i-2]
4 SUBI R1,R1,#8
5 BNEZ R1,LOOP
    
```

- **Symbolic Loop Unrolling**
 - Less code space
 - Fill & drain pipe only once vs. each iteration in loop unrolling



cs 152 L1 5.16

DAP Fa97, © UCB

Limits of Superscalar

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
 - Exactly 50% FP operations
 - No hazards
- If more instructions issue at same time, greater difficulty of decode and issue
 - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue
- VLIW: tradeoff instruction space for simple decoding
 - The long instruction word has room for many operations
 - By definition, all the operations the compiler puts in the long instruction word can execute in parallel
 - E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch
 - 16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide
 - Need compiling technique that schedules across several branches

cs 152 L1 5.17

DAP Fa97, © UCB

Loop Unrolling in VLIW

Memory reference 1	Memory reference 2	FP operation 1	FP op. 2	Int. op/branch	Clock
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16				7
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#48	8
SD -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays

7 results in 9 clocks, or 1.3 clocks per iteration

Need more registers in VLIW(EPIC => 128int + 128FP)

cs 152 L1 5.18

DAP Fa97, © UCB

Trace Scheduling

- **Parallelism across IF branches vs. LOOP branches**
- **Two steps:**
 - **Trace Selection**
 - Find likely sequence of basic blocks (*trace*) of (statically predicted) long sequence of straight-line code
 - **Trace Compaction**
 - Squeeze trace into few VLIW instructions
 - Need bookkeeping code in case prediction is wrong

HW Schemes: Instruction Parallelism

- **Why in HW at run time?**
 - Works when can't know real dependence at compile time
 - Compiler simpler
 - Code for one machine runs well on another
- **Key idea: Allow instructions behind stall to proceed**
 - DIVD F0, F2, F4
 - ADDD F10, F0, F8
 - SUBD F12, F8, F14
 - Enables out-of-order execution => out-of-order completion
 - ID stage checked both for structural

HW Schemes: Instruction Parallelism

- **Out-of-order execution divides ID stage:**
 1. **Issue**—decode instructions, check for structural hazards
 2. **Read operands**—wait until no data hazards, then read operands
- **Scoreboards allow instruction to execute whenever 1 & 2 hold, not waiting for prior instructions**
- **CDC 6600: In order issue, out of order execution, out of order commit (also called completion)**

Scoreboard Implications

- **Out-of-order completion => WAR, WAW hazards?**
- **Solutions for WAR**
 - Queue both the operation and copies of its operands
 - Read registers only during Read Operands stage
- **For WAW, must detect hazard: stall until other completes**
- **Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units**
- **Scoreboard keeps track of dependencies, state or operations**
- **Scoreboard replaces ID, EX, WB with 4 stages**

Performance of Dynamic SS

Iteration no.	Instructions	Issues	Executes	Writes result
1	LD F0,0(R1)	1	2	4
1	ADDD F4,F0,F2	1	5	8
1	SD 0(R1),F4	2	9	
1	SUBI R1,R1,#8	3	4	5
1	BNEZ R1,LOOP	4	5	
2	LD F0,0(R1)	5	6	8
2	ADDD F4,F0,F2	5	9	12
2	SD 0(R1),F4	6	13	
2	SUBI R1,R1,#8	7	8	9
2	BNEZ R1,LOOP	8	9	

- 4 clocks per iteration

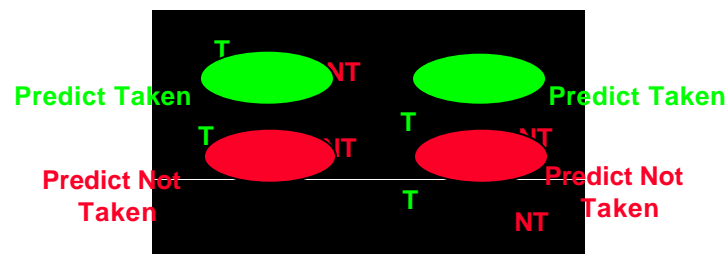
Branches, Decrements still take 1 clock cycle

cs 152 L1 5 23

DAP Fa97, © UCB

Dynamic Branch Prediction

- ° Solution: 2-bit scheme where change prediction only if get misprediction *twice*



cs 152 L1 5 24

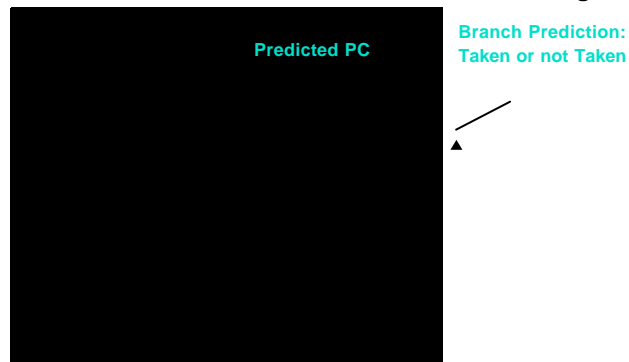
DAP Fa97, © UCB

BHT Accuracy

- **Mispredict because either:**
 - Wrong guess for that branch
 - Got branch history of wrong branch when index the table
- **4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%**
- **4096 about as good as infinite table, but 4096 is a lot of HW**

Need Address @ Same Time as Prediction

- **Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)**
 - Note: must check for branch match now, since can't use wrong branch address



- **Return instruction addresses predicted with stack**

Dynamic Branch Prediction Summary

- **Branch History Table: 2 bits for loop accuracy**
- **Branch Target Buffer: include branch address & prediction**

HW support for More ILP

- **Avoid branch prediction by turning branches into conditionally executed instructions:**
 - if (x) then A = B op C else NOP**
 - If false, then neither store result nor cause exception
 - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
 - EPIC: 64 1-bit condition fields selected so conditional execution
- **Drawbacks to conditional instructions**
 - Still takes a clock even if “annulled”
 - Stall if condition evaluated late
 - Complex conditions reduce effectiveness; condition becomes known late in pipeline

HW support for More ILP

- **Speculation**: allow an instruction *without* any consequences (including exceptions) if branch is not actually taken (“HW undo”)
- Often try to combine with dynamic scheduling
- Separate **speculative** bypassing of results from real bypassing of results
 - When instruction no longer speculative, write results (*instruction commit*)
 - execute out-of-order but commit in order

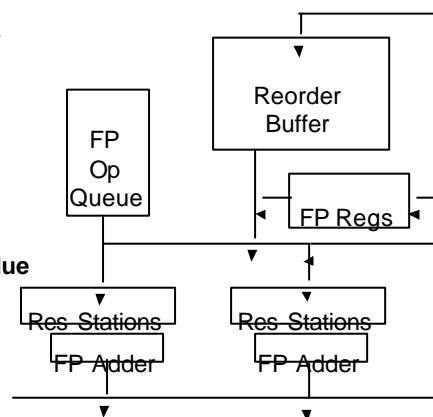
cs 152 L1 5 29

DAP Fa97, © UCB

HW support for More ILP

- **Need HW buffer for results of uncommitted instructions: reorder buffer**

- Reorder buffer can be operand source
- Once operand commits, result is found in register
- 3 fields: instr. type, destination, value
- Use reorder buffer number instead of reservation station
- Instructionsd instructions on mispredicted branches or on exceptions



cs 152 L1 5 30

DAP Fa97, © UCB

Pentium III

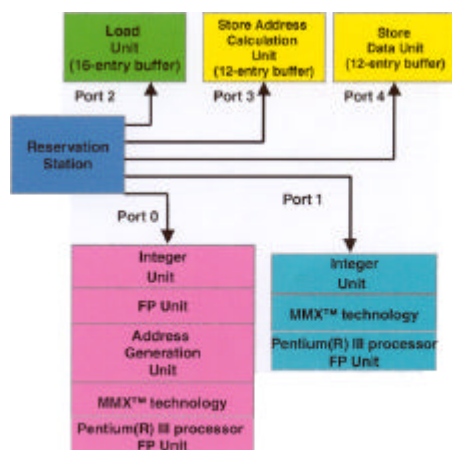
- Pentium II & III are based on Pentium Pro (P6) architecture
- PIII is available at speeds ranging from 450 MHz to 1.13 GHz
- System bus either 133 MHz or 100 MHz
- Versions with on-die full speed 256 KB L2 cache
- 32 KB (16 KB/16 KB) non-blocking, level 1 (L1) cache
- P6 Dynamic Execution microarchitecture including multiple branch prediction, data flow analysis and speculative execution
- Internet Streaming SIMD Extensions, consisting of 70 new instructions
- Intel MMX media technology
- Dual Independent Bus (DIB) architecture increases bandwidth and performance over single-bus processors
- Memory cacheability up to 4 GB and memory scalability up to 64

cs 152 L1 5 31

DAP Fa97, © UCB

Dynamic Scheduling in Pentium Pro architecture

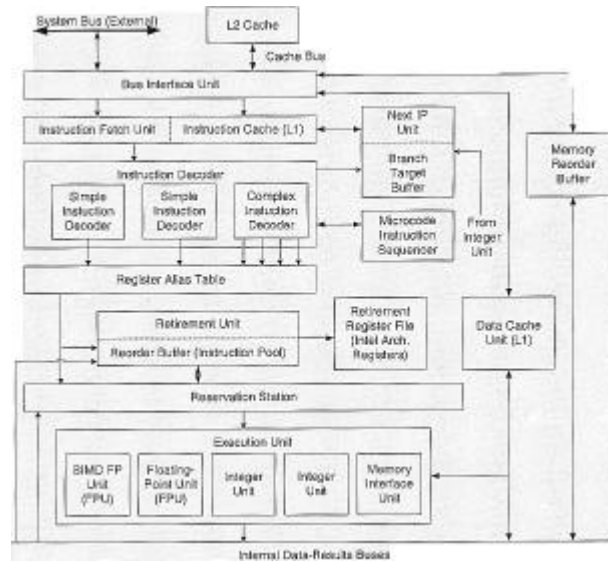
Execution Units and Ports in the Out-Of-Order Core



cs 152 L1 5 32

DAP Fa97, © UCB

Functional Block Diagram of P6 Family microarchitecture



cs 152 L1 5 33

DAP Fa97, © UCB

Dynamic Scheduling in P6 architecture

Parameter	P6
Max. instructions issued/clock	3
Max. instr. complete exec./clock	5
Max. instr. committed/clock	3
Instructions in reorder buffer	40
Number of rename buffers	40
Number of reservations stations	20
No. integer functional units (FUs)	2
No. floating point FUs	1
No. branch FUs	1
No. complex integer FUs	0
No. memory FUs	1 load +1 store

cs 152 L1 5 34

DAP Fa97, © UCB

Dynamic Scheduling in P6 (Pentium Pro) architecture

- P6 (Pentium II & III) doesn't pipeline 80x86 instructions
- P6 decode unit translates the Intel instructions into 72-bit micro-operations (- MIPS)
- Sends micro-operations to reorder buffer & reservation stations
- Takes 1 clock cycle to determine length of 80x86 instructions + 2 more to create the micro-operations
- Most instructions translate to 1 to 4 micro-operations
- Complex 80x86 instructions are executed by a conventional microprogram (8K x 72 bits) that issues long sequences of micro-operations
- Multiple branch prediction: predicts program execution through multiple branches
- Speculative execution: carries out instruction execution speculatively and based upon this optimized schedule, ensures that the processor's superscalar execution unit remain busy

Limits to Multi-Issue Machines

- **Inherent limitations of ILP**
 - 1 branch in 5: How to keep a 5-way VLIW busy?
 - Latencies of units: many operations must be scheduled
 - Need about Pipeline Depth x No. Functional Units of independent
Difficulties in building HW
 - Duplicate FUs to get parallel execution
 - Increase ports to Register File
 - VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
 - Increase ports to memory
 - Decoding SS and impact on clock rate, pipeline depth

Limits to Multi-Issue Machines

◦ **Limitations specific to either SS or VLIW implementation**

- Decode issue in SS
- VLIW code size: unroll loops + wasted fields in VLIW
- VLIW lock step => 1 hazard & all instructions stall
- VLIW & binary compatibility

3 Machines in middle of 90s

	Alpha 21164	Pentium II	HP PA-8000
Year	1995	1996	1996
Clock	600 MHz ('97)	300 MHz ('97)	236 MHz ('97)
Cache	8K/8K/96K/2M	16K/16K/0.5M	0/0/4M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (μ op)	56 instr
Rename regs	none	40	56

Summary

- **MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)**
- **More performance from deeper pipelines, parallelism**
- **Superscalar and VLIW**
 - **CPI < 1**
 - **Dynamic issue vs. Static issue**
 - **More instructions issue at same time, larger the penalty of hazards**
- **SW Pipelining**
 - **Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead**