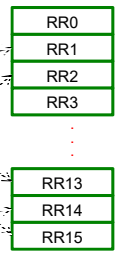
Renaming Registers

- Common variation of speculative design
- Reorder buffer keeps instruction information but not the result
- Extend register file with extra *renaming registers* to hold speculative results
- Rename register allocated at issue; result is written into renaming register when execution completes; renaming register is copied to real register on commit.
- Operands read either from register file (real or speculative) or via Common Data Bus
- Advantage: operands are always from single source (extended register file)

Name	Busy	Instruction		State	Dest.	R Reg.
ROB1	no	fld	F6, 34(R2)	Commit	F6	1
ROB2	no	fld	F2, 45(R3)	Commit	F2	13
ROB3	yes	fmul.d	F0, F2, F4	Write result	F0	15
ROB4	yes	fsub.d	F8, F2, F6	Write result	F8	2
ROB5	yes	fdiv.d	F10, F0, F6	Issued/executing	F10	
ROB6	yes	fadd.d	F6, F8, F2	Write result	F6	14

ROB status



Renaming Registers

(49)

Multiple issue processors (§3.7)

- Issue more than one instruction per clock cycle
  - VLIW processors (static scheduling by the compiler)
  - Superscalar processors
    - » Statically scheduled (in-order execution)
    - » Dynamically scheduled (out-of-order execution)
- results in CPI < 1 (Instructions per clock, IPC > 1)
- The fetch unit gets an *issue packet* which contains multiple instructions
- The issue unit issues 1-8 instructions every cycle (usually, the issue unit is itself pipelined)
  - independent instructions
  - multiple resources should be available
  - branch prediction should be accurate
- Leads to multiple instruction completion per cycle

(50)

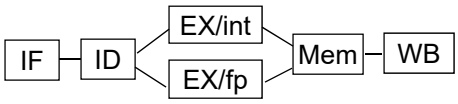
Five primary approaches

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	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	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

(51)

A statically scheduled MIPS superscalar

- Assume two execution pipes
  - one for load, store, branch and integer operations
  - one for floating point operations
  - IF fetches 2 instructions
  - ID process 2 instructions
  - increase load on register file



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

(52)

Loop Unrolling in Superscalar

Assume: 1 cycle latency from *fld* to *fadd.d*
2 cycle latency from *fadd.d* to *fsd*

	Integer instruction		*FP instruction*		*Clock cycle*
Loop:	fld	F0,0(R1)			1
	fld	F6,-8(R1)			2
	fld	F10,-16(R1)	fadd.d	F4,F0,F2	3
	fld	F14,-24(R1)	fadd.d	F8,F6,F2	4
	fld	F18,-32(R1)	fadd.d	F12,F10,F2	5
	fsd	F4, 0(R1)	fadd.d	F16,F14,F2	6
	fsd	F8, -8(R1)	fadd.d	F20,F18,F2	7
	fsd	F12, -16(R1)			8
	fsd	F16, -24(R1)			9
	fsd	F20, -32(R1)			10
	addi	R1, R1, -40			11
	bne	R1, R2, Loop			12

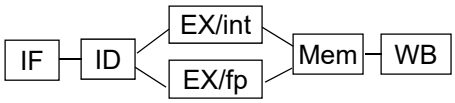
F0

F4

- **Unrolled 5 times to avoid delays.**
- **12 clocks, or 2.4 clocks per iteration.**

(53)

MIPS superscalar

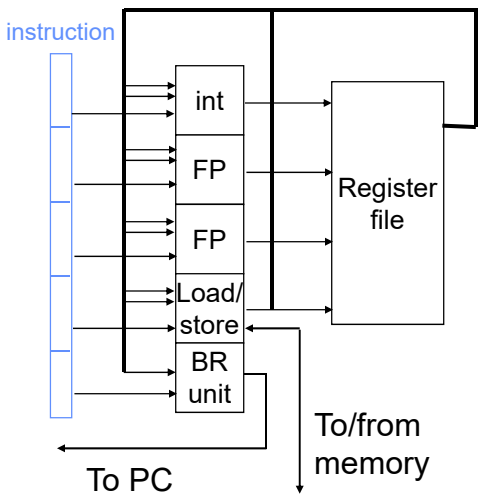


- While Integer/FP split is simple for the HW, we can get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - No hazards
- If two instructions issued every cycle, greater difficulty of decode and issue
  - Even 2 units => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue
- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can't use it, nor instructions in next slot
- Control hazards is twice as expensive

(54)

VLIW architectures

- Compiler packs multiple, independent, operations into one single instruction.
- E.g.,
  - 1 integer operations,
  - 2 FP ops,
  - 1 Memory refs,
  - 1 branch.
  - instruction = 5*24 = 120 bits.
- Lock-step issue of instructions (one per cycle)
- In a pure VLIW approach, the compiler resolves all hazards (resolved by hardware in MIPS superscalars)



(55)

Loop Unrolling in VLIW

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
fld F0,0(R1)	fld F6,-8(R1)				1
fld F10,-16(R1)	fld F14,-24(R1)				2
fld F18,-32(R1)	fld F22,-40(R1)	fadd.d F4,F0,F2	fadd.d F8,F6,F23		
fld F26,-48(R1)		fadd.d F12,F10,F2	fadd.d F16,F14,F2		4
		fadd.d F20,F18,F2	fadd.d F24,F22,F2		5
fsd F4,0(R1)	fsd F8,-8(R1)	fadd.d F28,F26,F2			6
fsd F12,-16(R1)	fsd F16,-24(R1)				7
fsd F20,-32(R1)	fsd F24,-40(R1)			addi R1,R1,-48	8
fsd F28,-0(R1)				bne R1,R2,Loop	9

- Unrolled 7 times
- 7 results in 9 clocks, or 1.3 clocks per iteration.
- Average: 2.5 ops per clock, 50% efficiency
- Note: Need more registers in VLIW (15 vs. 11 in SS)

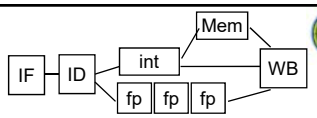
(56)

Dynamic scheduling (§3.8)

- Extends Tomasulo's algorithm to issue two (or more) instructions simultaneously to reservation stations.
- Either issue an instruction every half clock cycle, or double the logic to handle two instructions at once.
- Use the same logic. Some restrictions may be used to simplify hardware.
  - Example: issue only one FP and one int. operation every clock cycle. This reduces the load on the register files.
  - Do not issue dependent instructions in the same cycle
- To deal with control hazards without speculation (no ROBs): instructions following a branch can be issued but cannot start execution before the branch is resolved.
- Will look at the execution of

```
L1:  fld     F0, 0(R1)
     fadd.d  F4, F0, F2
     fsd     F4, 0(R1)
     addi    R1, R1, -8
     bne     R1, R2, L1
```

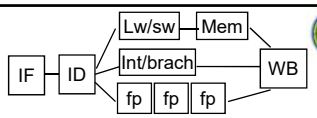

Dual issue with one int. and one FP add unit (not pipelined). Latency = 1, 2 and 3 for int. operations, loads and FP adds (one CDB).



Iteration	Instruction		Issued at	Executes	Mem access	Write CDB	comments
1	fld	F0, 0(R1)	1	2	3	4	First issue
1	fadd.d	F4, F0, F2	1	5		8	Wait for fld
1	fsd	F4, 0(R1)	2	3	9		Wait for fadd.d
1	addi	R1, R1, -8	2	4		5	Wait for ALU
1	bne	R1, R2, L1	3	6			wait for addi
2	fld	F0, 0(R1)	4	7	8	9	Wait for bne
2	fadd.d	F4, F0, F2	4	10		13	Wait for fld
2	fsd	F4, 0(R1)	5	8	14		Wait for fadd.d
2	addi	R1, R1, -8	5	9		10	Wait for ALU
2	bne	R1, R2, L1	6	11			wait for addi
3	fld	F0, 0(R1)	7	12	13	14	Wait for bne
3	fadd.d	F4, F0, F2	7	15		18	Wait for fld
3	fsd	F4, 0(R1)	8	13	19		Wait for fadd.d
3	addi	R1, R1, -8	8	14		15	Wait for ALU
3	bne	R1, R2, L1	9	16			wait for addi

- No speculation: Instruction after *bne* cannot be issued in same cycle since the branch is not yet predicted (orange arrows)
- *fld, fsd, addi* and *bne* use the same integer Execute unit (see blue circles)
- Assume as many reservation stations as needed

If we have a separate int. unit for memory address calculation and we have a second CDB.

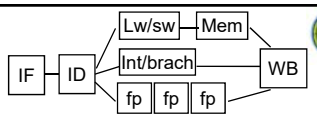


Iteration	Instruction		Issued at	Executes	Mem access	Write CDB	comments
1	fld	F0, 0(R1)	1	2	3	4	First issue
1	fadd.d	F4, F0, F2	1	5		8	Wait for fld
1	fsd	F4, 0(R1)	2	3	9		Wait for fadd.d
1	addi	R1, R1, -8	2	3		4	Exec. earlier
1	bne	R1, R2, L1	3	5			wait for addi
2	fld	F0, 0(R1)	4	6	7	8	Wait for bne
2	fadd.d	F4, F0, F2	4	9		12	Wait for fld
2	fsd	F4, 0(R1)	5	7	13		Wait for fadd.d
2	addi	R1, R1, -8	5	6		7	Exec. earlier
2	bne	R1, R2, L1	6	8			wait for addi
3	fld	F0, 0(R1)	7	9	10	11	Wait for bne
3	fadd.d	F4, F0, F2	7	12		15	Wait for fld
3	fsd	F4, 0(R1)	8	10	16		Wait for fadd.d
3	addi	R1, R1, -8	8	9		10	Exec. earlier
3	bne	R1, R2, L1	9	11			wait for addi

No speculation: no instruction after *bne* can start before *bne* completes execution (cycle 5)

(59)

Multiple issue with speculation



Consider same example (as last slide)

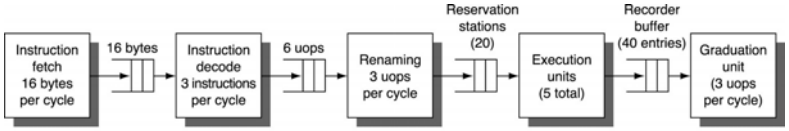
Iteration	Instruction		Issued at	Executes	Mem access	Write CDB	comments
1	fld	F0, 0(R1)	1	2	3	4	First issue
1	fadd.d	F4, F0, F2	1	5		8	Wait for fld
1	fsd	F4, 0(R1)	2	3	9		Wait for fadd.d
1	addi	R1, R1, -8	2	3		4	
1	bne	R1, R2, L1	3	5			wait for addi
2	fld	F0, 0(R1)	~~4~~ 3	~~6~~ 4	~~7~~ 5	~~8~~ 6	No wait for bne
2	fadd.d	F4, F0, F2	4	~~9~~ 7		~~12~~ 10	Wait for fld
2	fsd	F4, 0(R1)	~~5~~ 4	~~7~~ 5	~~13~~ 11		Wait for fadd.d
2	addi	R1, R1, -8	5	6		7	
2	bne	R1, R2, L1	~~6~~ 5	8			wait for addi
3	fld	F0, 0(R1)	~~7~~ 6	~~9~~ 7	~~10~~ 8	~~11~~ 9	No wait for bne
3	fadd.d	F4, F0, F2	~~7~~ 6	~~12~~ 10		~~15~~ 13	Wait for fld
3	fsd	F4, 0(R1)	~~8~~ 7	~~10~~ 8	~~16~~ 14		Wait for fadd.d
3	addi	R1, R1, -8	~~8~~ 7	~~9~~ 8		~~10~~ 9	
3	bne	R1, R2, L1	~~9~~ 8	~~11~~ 10			wait for addi

Allow speculation, and may issue a branch with another instruction

(60)

Dynamic Scheduling in the P6 microarchitecture

- Doesn't pipeline the IA-32 instructions (complex instructions)
- Decode unit translates the Intel instructions into MIPS-like micro-operations, called μops (most instructions translate to 1 – 4 μ ops).
- Complex IA-32 instructions are executed by a conventional microprogram that issues long sequences of μ ops
- Sends μops to reorder buffer & reservation stations



- A 14-stage pipeline:
  - 8 stages for in-order fetch of IA-32 instructions, generating μ ops, decoding, dynamic branch prediction and dispatch of μ ops
  - 3 stages for execution into 5 pipelined units (from 1 to 32 pipelined stages)
  - 3 stages for instruction commit.

(61)

Limits to Multi-Issue Machines

- 1 branch in 5: How to keep a 5-issue processor busy?
- Latencies of units: many operations must be scheduled
- Need about (Pipeline Depth x No. Functional Units) of independent instructions.
- Need More instruction fetch bandwidth (easy)
- Need Duplicate FUs to get parallel execution (easy)
- Need more ports in Register File (hard) and more memory bandwidth (harder)
- Impact of decoding multiple instructions on clock rate and pipeline depth.
- Decode issue in Superscalar: how wide is practical?
- More logic lead to larger power consumption ➔ lower performance per watt.

(62)

Return address prediction

- Most unconditional branches come from function returns
- The same procedure can be called from multiple sites
  - Causes the buffer to potentially forget about the return address from previous calls

Integrated instruction Fetch unit that performs

- Branch prediction
- Instruction prefetch

Energy Efficiency

- Speculation is only energy efficient when it significantly improves performance

Value Prediction

- Loads that load from a constant pool
- Instruction that produces a value from a small set of values
- Not incorporated into modern processors

(63)

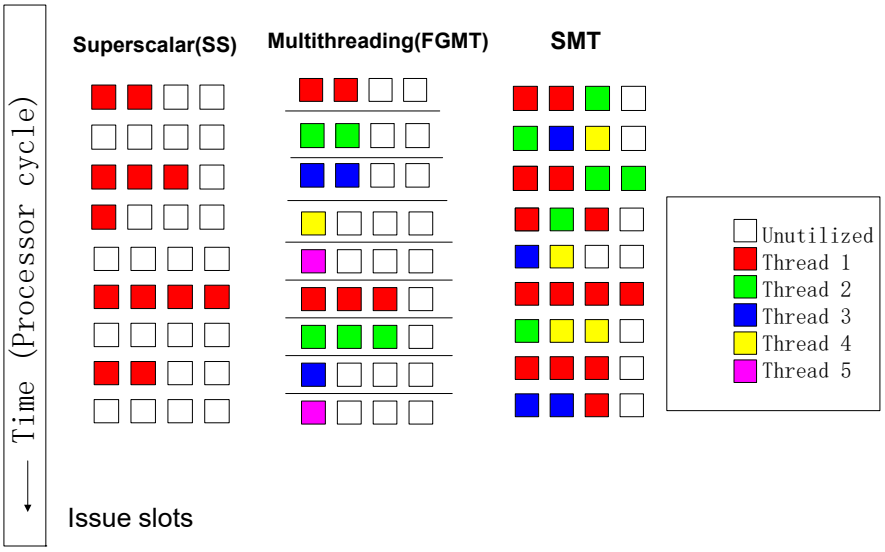
Simultaneous Multithreading (§3.10)

- Key idea

  Issue multiple instructions from multiple threads each cycle

- Features
  - Fully exploit thread-level parallelism and instruction-level parallelism.
  - Better Performance for
    - Mix of independent programs
    - Programs that are parallelizable

(64)

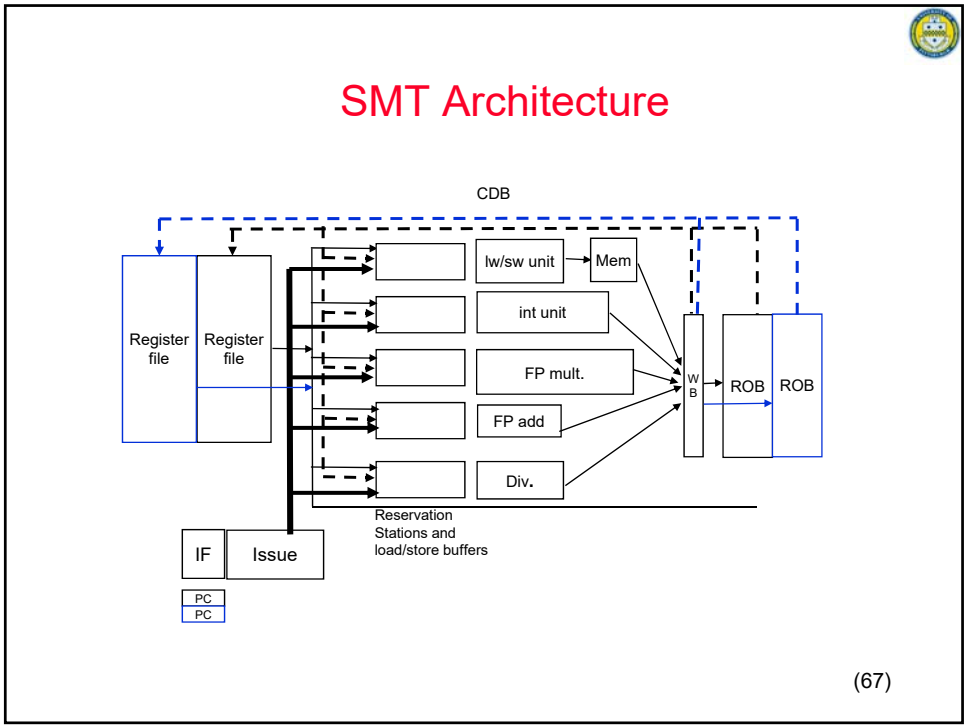


(65)

SMT ARCHITECTURE

- Base Processor: like out-of-order superscalar processor.[MIPS R10000]
- With N simultaneous running threads, need N PC and more than N*32 physical registers for register renaming in total.
- Need large register files, longer register access time → pipeline stages are added
- Share the cache hierarchy and branch prediction hardware.
- Cache and branch prediction interference, and increased memory pressure.

(66)



SMT Architecture

(67)