Multiple Issue (Superscalar)

- basic pipeline: single, in-order issue
- first extension: multiple issue (superscalar)
  - still in-order
- future topics
  - out-of-order execution
  - fancy static scheduling (i.e., compiler help)

Readings

H+P

- Chapter 2
- Chapter 3 (not required, but suggested)

Research papers (not yet ready to read, but will be soon!):

- Hinton et al: “The Microarchitecture of the Pentium 4 Processor”
- Palacharla, Jouppi, and Smith: “Complexity-Effective Superscalar Processors”
- Akkary, Rajwar, and Srinivasan: “Checkpoint Processing and Recovery”

Flynn Meets Fisher

- "Flynn bottleneck"
  - single issue performance limit is CPI = IPC = 1
  - hazards + overhead ⇒ CPI >= 1 (IPC <= 1)
  - diminishing returns from superpipelining [Hrishikesh paper!]
- solution: issue multiple instructions per cycle

	1	2	3	4	5	6	7
inst0	F	D	X	M	W		
inst1	F	D	X	M	W		
inst2		F	D	X	M	W	
inst3		F	D	X	M	W	

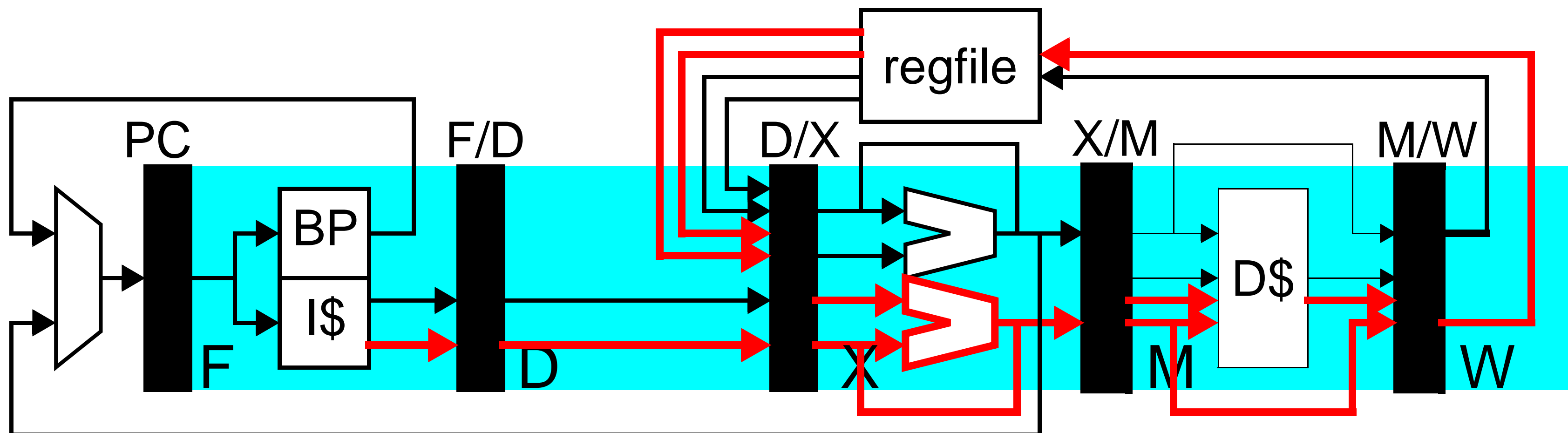
- 1st superscalar: IBM America → RS/6000 → POWER1

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

Base Implementation

- statically scheduled (in-order) superscalar
  - executes unmodified sequential programs
  - figures out on its own what can be done in parallel
  - e.g., Sun UltraSPARC, Alpha 21164
  - we'll start with this one

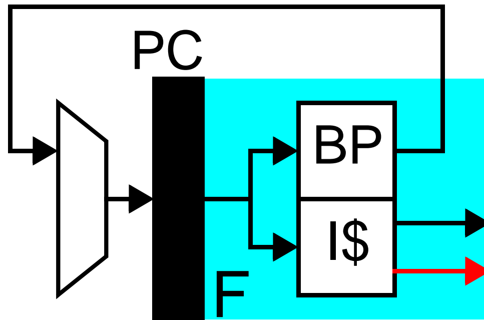
5-Stage Dual-Issue Pipeline



- what is involved in
  - fetching two instructions per cycle?
  - decoding two instructions per cycle?
  - executing two ALU operations per cycle?
  - accessing the data cache twice per cycle?
  - writing back two results per cycle?
- what about 4 or 8 instructions per cycle?

© 2009 by Sorin, Roth, Hill, Wood,
Sohi, Smith, Vijaykumar, Lipasti

Wide Fetch



what is involved in fetching multiple instructions per cycle?

- if instructions are sequential...
  - and on same cache line ⇒ nothing really
  - and on different cache lines ⇒ banked I$ + combining network
- if instructions are not sequential...
  - – more difficult
  - – two serial I$ accesses (access1⇒predict target⇒access2)? no
- note: embedded branches OK as long as predicted NT
  - serial access + prediction in parallel
  - if prediction is T, discard serial part after branch

One Solution to Wide Fetch: Trace Cache

problem: low fetch utilization on taken branches

- only fetch up to taken branch, remaining fetch slots lost

trace cache: combine branch predictor with I$

  - [Weiser+Peleg'95, Rotenberg+Bennett+Smith'96]

- stores dynamic instruction sequences
  - tag: initial PC + directions of embedded branches
  - fetch from trace, but make sure that branch directions were ok
  - typically backed by I$ (in case of trace cache miss)
- used in Pentium4
  - actually a decoded (μop) trace cache

Trace Cache Example

- instruction cache with 2 instrs per cache block

I$	
tag	data
PC0	inst0,inst1
PC2	inst2,inst3
PC4	inst4,inst5

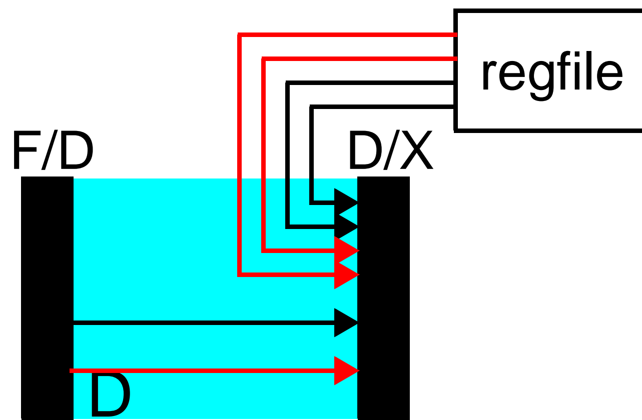
	1	2	3	4	5	6	7
inst0 (beq r1, inst4)	F	D	X	M	W		
inst4	f*	F	D	X	M	W	
inst5		F	D	X	M	W	
inst6			F	D	X	M	W

- trace-cache with 2 instrs per cache block

T$	
tag	data
PC0:T	inst0,inst4
PC2:–	inst2,inst3
PC5:–	inst5,inst6

	1	2	3	4	5	6	7
inst0 (beq r1, inst4)	F	D	X	M	W		
inst4	F	D	X	M	W		
inst5		F	D	X	M	W	
inst6		F	D	X	M	W	

Wide Decode



what is involved in decoding N instructions per cycle?

- actually decoding instructions?
  - + easy if fixed length instructions (multiple decoders)
  - – harder (but possible) if variable length
- reading input register values?
  - – 2N register read ports (register file read latency ~2N)
  - • actually less than 2N, since most values come from bypasses
- what about the stall logic to enforce RAW dependences?

N^2 Dependence Cross-Check

- remember stall logic for single issue pipeline
  - rs1(D) == rd(D/X) || rs1(D) == rd(X/M) || rs1(D) == rd(M/W)
  - same for rs2(D)
  + if full-bypassing, then just: rs1(D) == rd(D/X) && op(D/X) == LOAD

– doubling issue width (N) quadruples stall logic!
  - not only 2 instructions in D, but two instructions in every stage
  - (rs1(D_1) == rd(D/X_1) && op(D/X_1) == LOAD)
  - (rs1(D_1) == rd(D/X_2) && op(D/X_2) == LOAD)
  - repeat for rs1(D_2), rs2(D_1), rs2(D_2)
  - also check dependence of 2nd instruction on 1st: rs1(D_2) == rd(D_1)

"N^2 dependence cross-check"

– for N-wide pipeline, stall (and bypass) circuits grow as N^2

Superscalar Stalls

- invariant: stalls propagate upstream to younger instructions
- what if older instruction in issue “pair” (inst0) stalls?
  - younger instruction (inst1) stalls too, cannot pass it
- what if younger instruction (inst1) stalls?
  - can older instruction from next group (inst2) move up?

rigid pipe ⇒ no

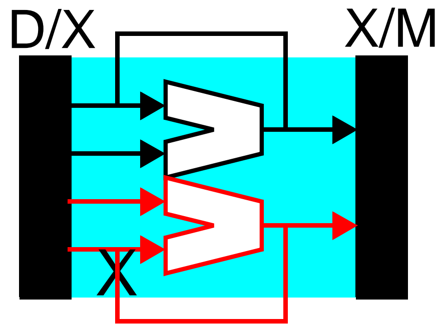
	1	2	3	4
inst0	F	D	X	X
inst1	F	D	**d***	X
inst2		F	**p***	D
inst3		F	**p***	D

fluid pipe ⇒ yes

	1	2	3	4
inst0	F	D	X	M
inst1	F	D	**d***	X
inst2		F	**D**	X
inst3		F	**p***	D

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

Wide Execute

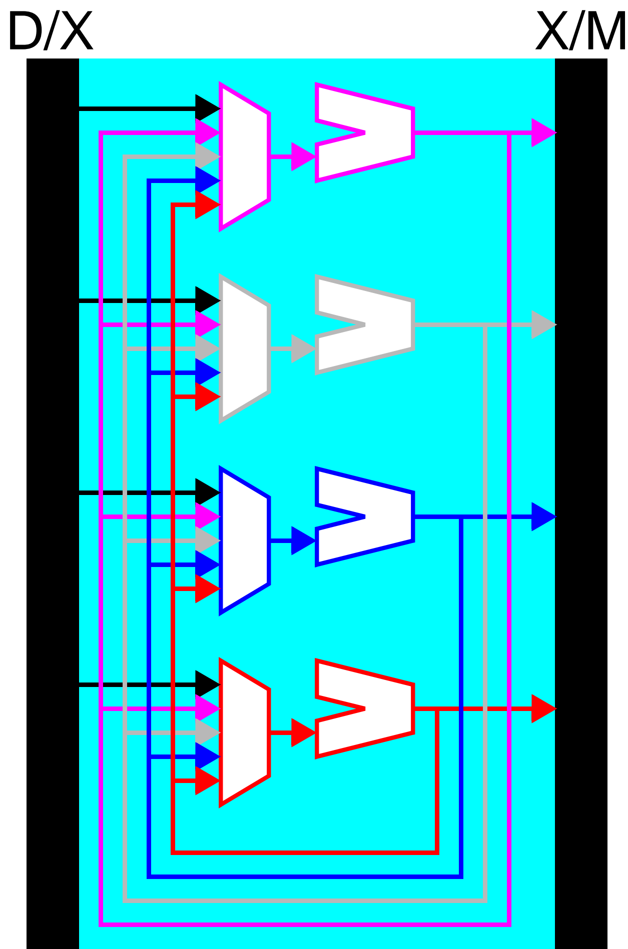


what is involved in execut]ing N instructions per cycle?

- multiple execution units...N of every kind?
  - N ALUs? OK, ALUs are small
  - N FP dividers? no, FP dividers are huge (and fdiv is uncommon)
- typically have some mix (proportional to instruction mix)
  - RS/6000: 1 ALU/memory/branch + 1 FP
  - Pentium: 1 any + 1 ALU
  - Pentium II: 1 ALU/FP + 1 ALU + 1 load + 1 store + 1 branch
  - Alpha 21164: 1 ALU/FP/branch + 2 ALU + 1 load/store

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

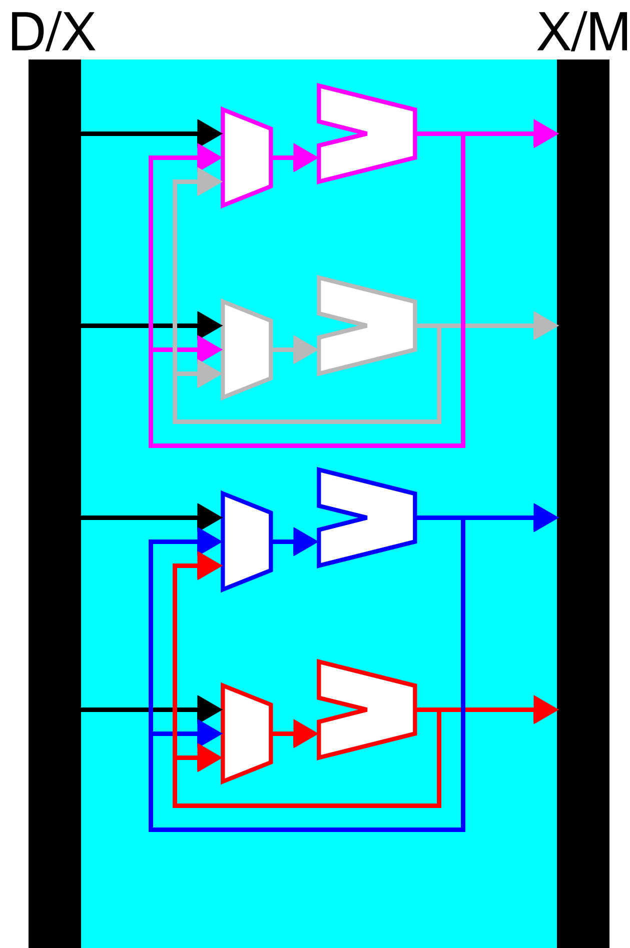
N^2 Bypass



- N^2 bypass logic... OK
  - only 5-bit quantities
  - compare to generate 1-bit outcomes
  - similar to stall logic

– N^2 bypass buses... not even close to OK
  - 32-bit or 64-bit quantities
  - broadcast, route, and multiplex (mux)
  - – difficult to lay out and route all the wires
  - – wide (SLOW) muxes
  - big design problem today

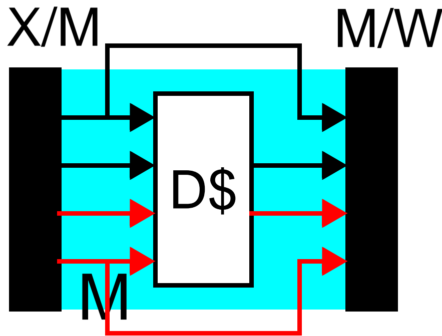
One Solution to N^2 Bypass: Clustering



- group functional units into clusters
  - full bypass within cluster
  - no bypass between clusters
  - ~(N/k) inputs at each mux
  - $\sim(N/k)^2$ routed buses in each cluster
- steer instructions to different clusters
  - dependent instructions to same cluster
  - exploit intra-cluster bypass
  - static or dynamic steering is possible
- e.g., Alpha 21264
  - 4-wide, 300MHz
  - full bypass didn't fit into 1 clock cycle
  - 2 clusters with full intra-cluster bypass

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

Wide Memory Access

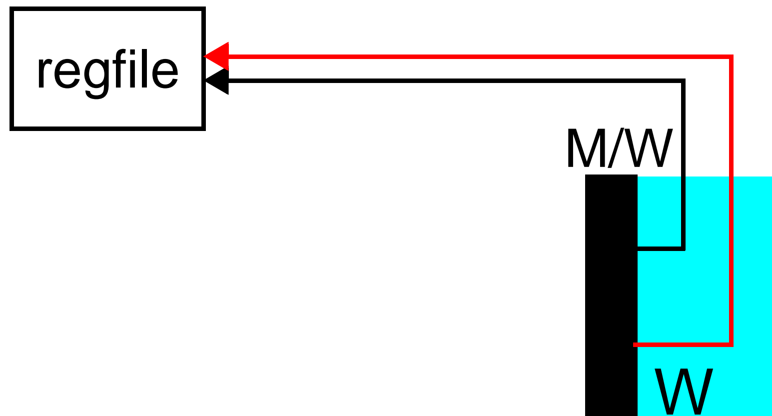


what is involved in accessing memory for multiple instructions per cycle?

- multi-banked D$
  - requires bank assignment and conflict-detection logic
- (rough) instruction mix: 20% loads, 15% stores
  - for width N, we need about 0.2*N load ports, 0.15*N store ports

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

Wide Writeback



what is involved in writing back multiple instructions per cycle?

- nothing too special, just another port on the register file
  - everything else is taken care of earlier in pipeline
- adding ports isn't free, though
  - increases area
  - increases access latency

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

Multiple Issue Summary

- superscalar problem spots
  - fetch, branch prediction ⇒ trace cache?
  - decode (N^2 dependence cross-check)
  - execute (N^2 bypass) ⇒ clustering?

next up: dynamic scheduling (out-of-order issue)