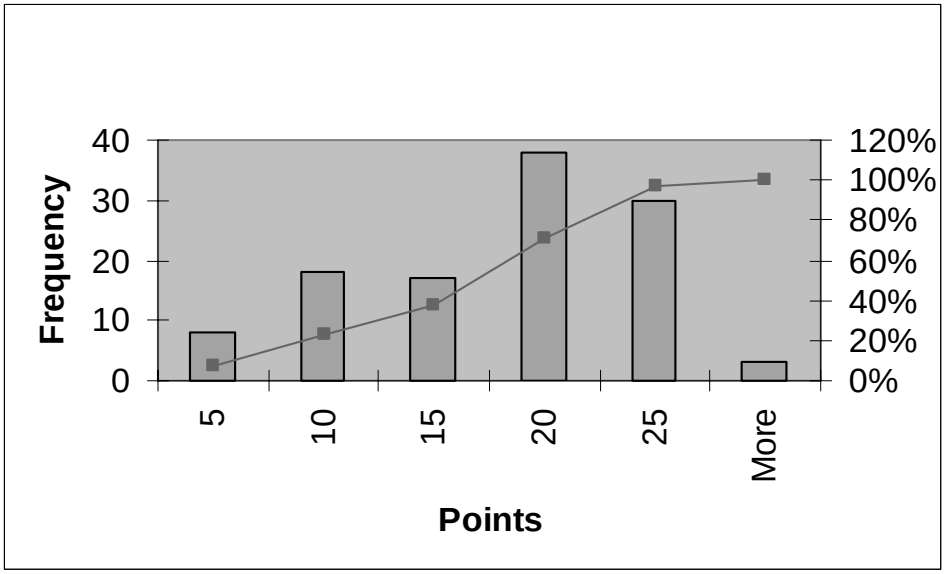
2001 Midterm Results

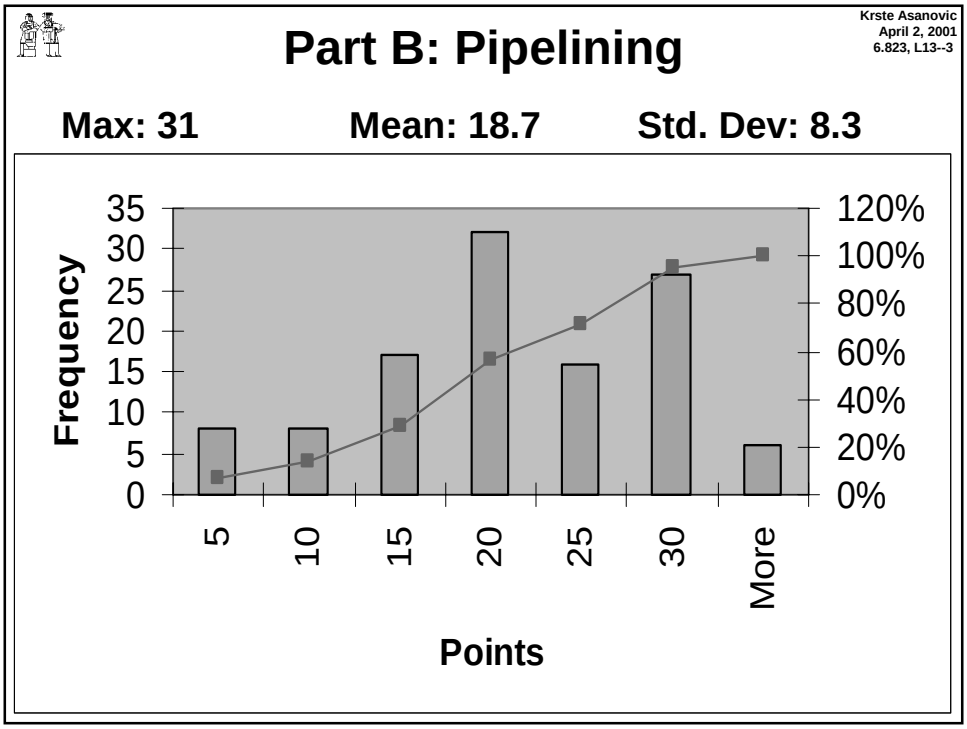
Krste Asanovic
Laboratory for Computer Science
M.I.T.

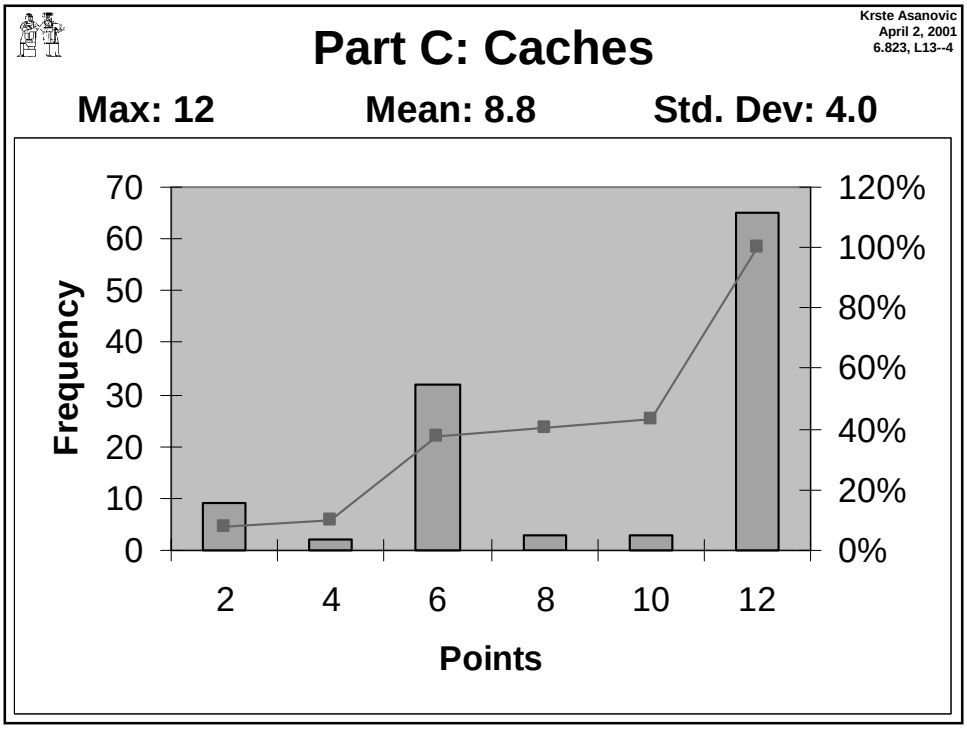
http://www.csg.lcs.mit.edu/6.823

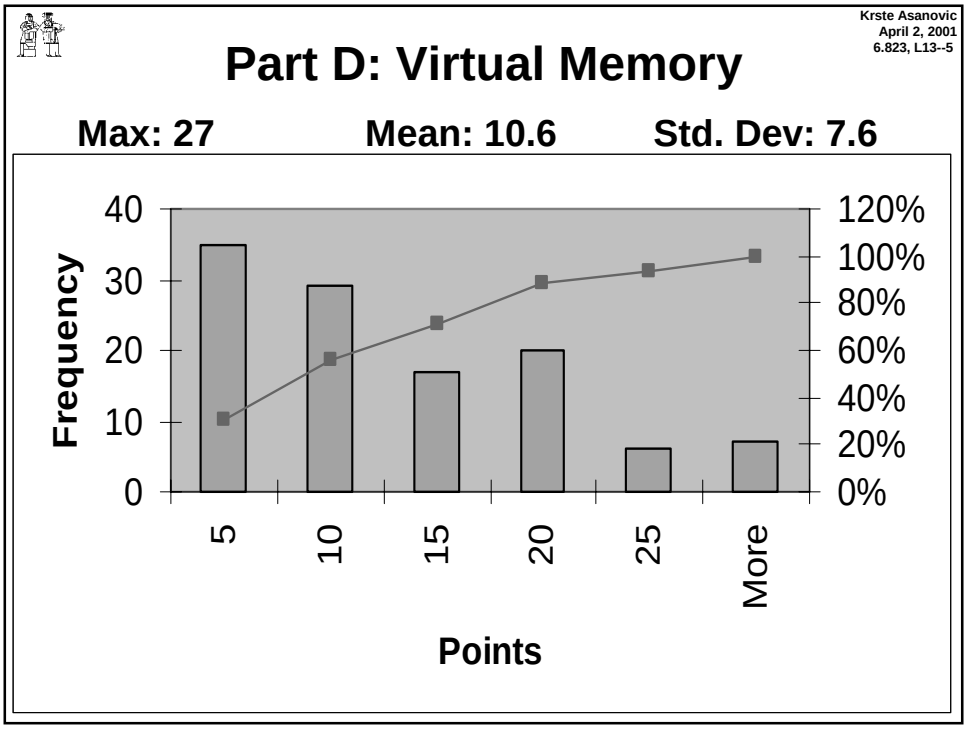
Part A: Microprogramming

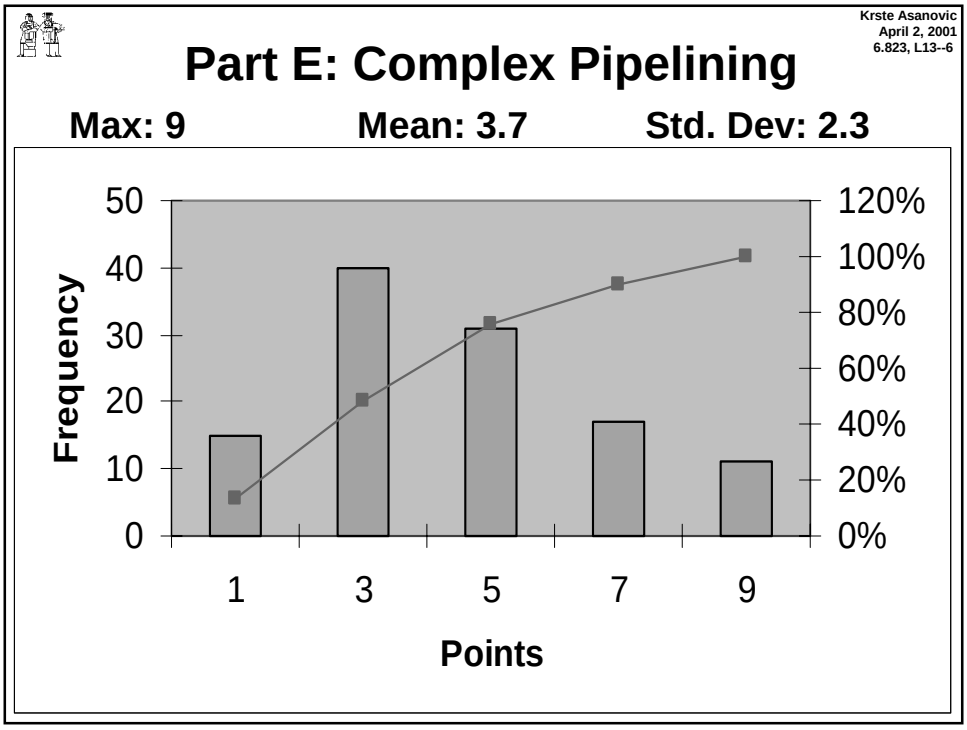
Max: 26 **Mean: 16.0** **Std. Dev: 6.3**




Krste Asanovic
April 2, 2001
6.823, L13--3
Part B: Pipelining
Max: 31
Mean: 18.7
Std. Dev: 8.3
Frequency
35
30
25
20
15
10
5
0
120%
100%
80%
60%
40%
20%
0%
5
10
15
20
25
30
More
Points

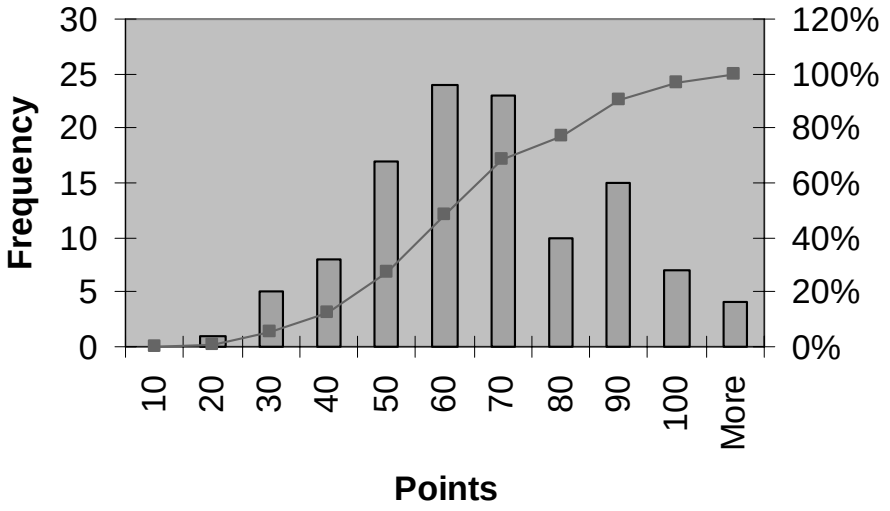

Krste Asanovic
April 2, 2001
6.823, L13--4
Part C: Caches
Max: 12
Mean: 8.8
Std. Dev: 4.0
Frequency
70
60
50
40
30
20
10
0
120%
100%
80%
60%
40%
20%
0%
2
4
6
8
10
12
Points


Krste Asanovic
April 2, 2001
6.823, L13--5
Part D: Virtual Memory
Max: 27
Mean: 10.6
Std. Dev: 7.6
Frequency
0
10
20
30
40
120%
100%
80%
60%
40%
20%
0%
5
10
15
20
25
More
Points


Krste Asanovic
April 2, 2001
6.823, L13--6
Part E: Complex Pipelining
Max: 9
Mean: 3.7
Std. Dev: 2.3
Frequency
0
10
20
30
40
50
120%
100%
80%
60%
40%
20%
0%
1
3
5
7
9
Points

Overall Midterm Statistics

Max: 110 **Mean: 62.7** **Std. Dev: 19.8**



Advanced Superscalar Architectures

Krste Asanovic
Laboratory for Computer Science
M.I.T.

http://www.csg.lcs.mit.edu/6.823

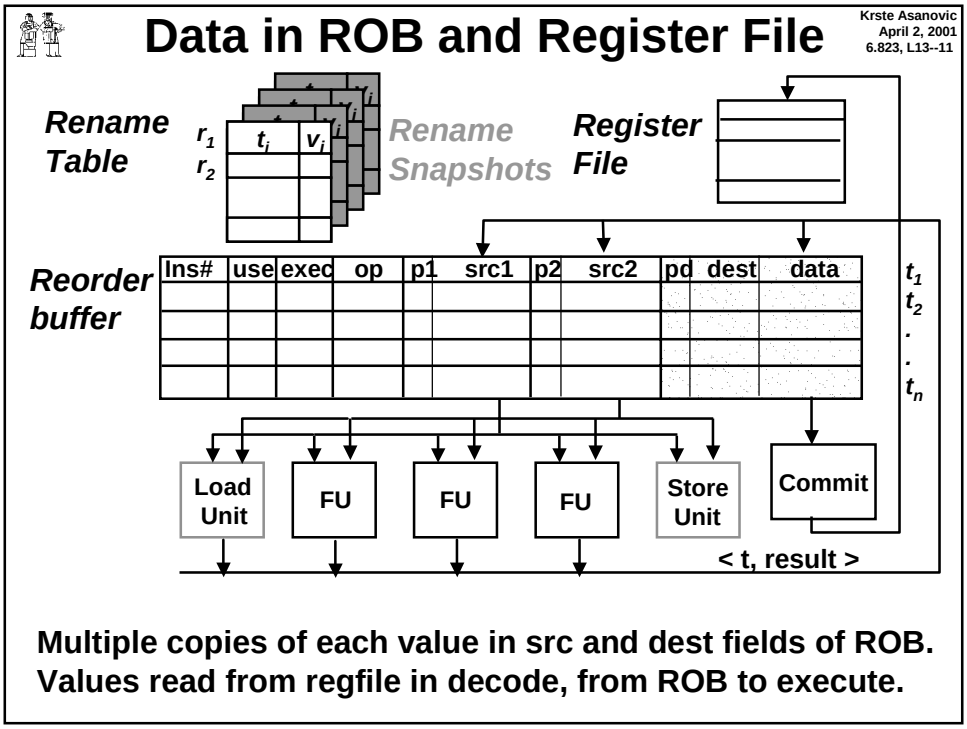
A Modern Superscalar?

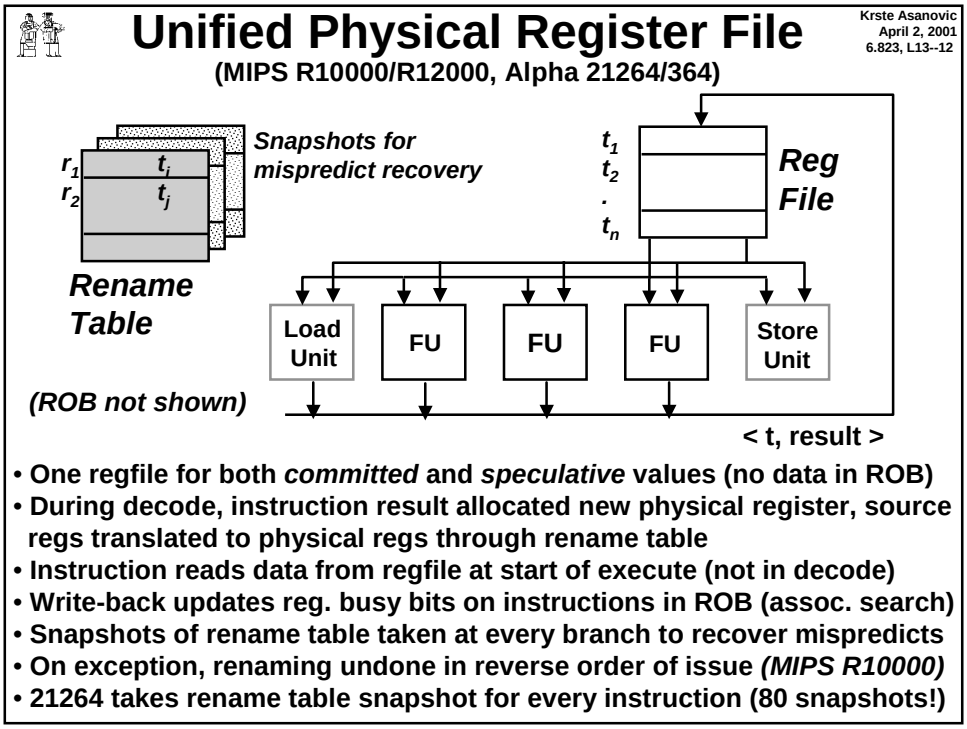
- **Execute up to seven instructions per cycle, out-of-order issue**
- **Register renaming**
- **Predicated execution**
- **Dynamic branch prediction and branch target buffer**
- **Multiway branches**
- **Hardware simultaneous multithreading with second program counter**

ACS Project *(1965-1969):*

IBM's response to CDC6600

- **Execute up to seven instructions per cycle, with out-of-order issue**
  - **three arithmetic instructions issued out-of-order, plus three index instructions (inc. two load/store) & one branch issued in-order**
  - **up to 50 instructions in-flight**
- **Register renaming**
  - **background registers allow loads of next iteration to overlap execution of last iteration (developed independently of Tomasulo's algorithm)**
- **Predicated execution**
  - **avoid pipeline disruption from branches**
- **Dynamic branch prediction and branch target buffer (BTB)**
  - **single-bit predictor to help hide instruction fetch latency, 8-entry BTB**
- **Multiway branches**
  - **specify multiple branch predicates and targets in multiple instructions, then single instruction executes entire branch table**
- **ACS-360** ***(IBM marketing eventually required 360 compatibility)***
  - **added hardware simultaneous multithreading with second program counter**
- **Project cancelled 1969 but had large impact on future IBM designs**
  - **optimizing compiler work (Cocke) led to IBM 801 RISC design**
  - **some features introduced in ASC live on in POWER/PowerPC**


Data in ROB and Register File
Krste Asanovic
April 2, 2001
6.823, L13--11
Rename Table
Rename Snapshots
Register File
Reorder buffer
Ins# use exec op p1 src1 p2 src2 pd dest data
Load Unit
FU
FU
FU
Store Unit
Commit
< t, result >
Multiple copies of each value in src and dest fields of ROB.
Values read from regfile in decode, from ROB to execute.


Unified Physical Register File
(MIPS R10000/R12000, Alpha 21264/364)
Krste Asanovic
April 2, 2001
6.823, L13--12
Snapshots for mispredict recovery
Rename Table
(ROB not shown)
Reg File
Load Unit
FU
FU
FU
Store Unit
< t, result >
• One regfile for both committed and speculative values (no data in ROB)
• During decode, instruction result allocated new physical register, source regs translated to physical regs through rename table
• Instruction reads data from regfile at start of execute (not in decode)
• Write-back updates reg. busy bits on instructions in ROB (assoc. search)
• Snapshots of rename table taken at every branch to recover mispredicts
• On exception, renaming undone in reverse order of issue (MIPS R10000)
• 21264 takes rename table snapshot for every instruction (80 snapshots!)

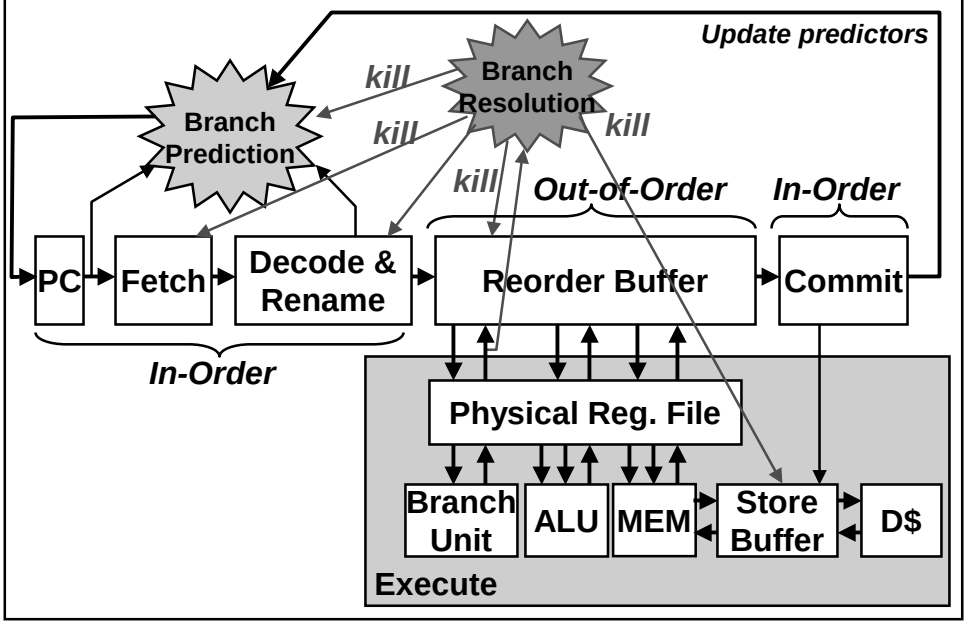
Unified Physical Register File

(MIPS R10K/R12K, Alpha 21264/364)

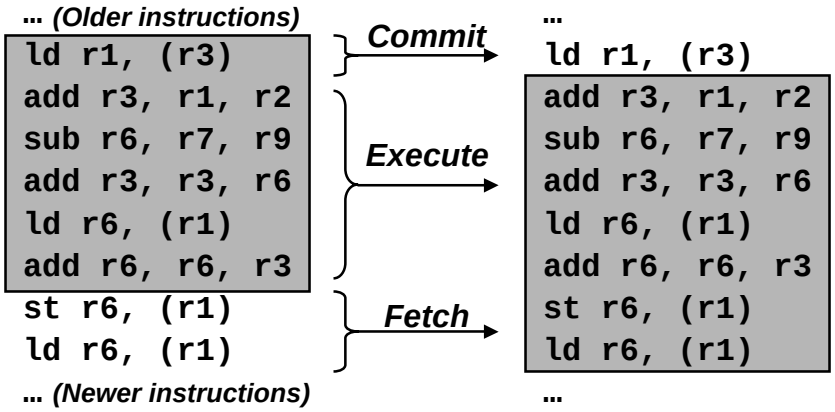
- During decode, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees physical register numbers

```
ld r1, (r3)              ld P1, (Px)
add r3, r1, #4           add P2, P1, #4
sub r6, r7, r9           sub P3, Py, Pz
add r3, r3, r6   Rename  add P4, P2, P3
ld r6, (r1)              ld P5, (P1)
add r6, r6, r3           add P6, P5, P4
st r6, (r1)              st P6, (P1)
ld r6, (r11)             ld P7, (Pw)
```

Speculative and Out-of-Order Execution



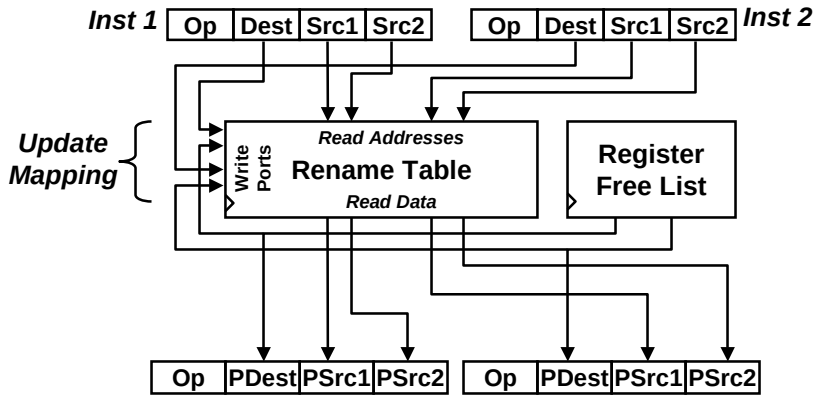
Reorder Buffer Holds Active Instruction Window



Cycle *t* Cycle *t + 1*

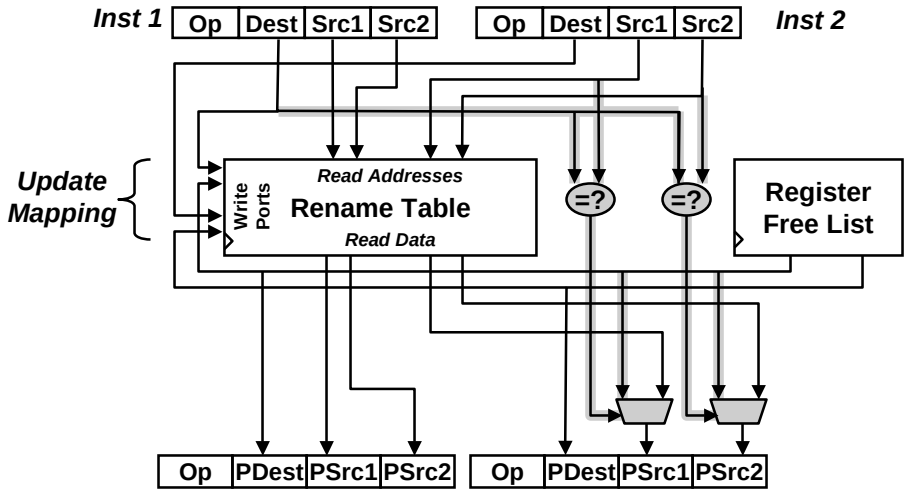
Superscalar Register Renaming

- During decode, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees physical register numbers



Does this work?

Superscalar Register Renaming



Must check for RAW hazards between instructions issuing in same cycle. Can be done in parallel with rename lookup.
(MIPS R10K renames 4 serially-RAW-dependent insts/cycle)

Lifetime of Physical Registers

- **Physical regfile holds committed and speculative values**
- **Physical registers decoupled from ROB entries (no data in ROB)**

```
ld r1, (r3)              ld P1, (Px)
add r3, r1, #4           add P2, P1, #4
sub r6, r7, r9           sub P3, Py, Pz
add r3, r3, r6   Rename  add P4, P2, P3
ld r6, (r1)              ld P5, (P1)
add r6, r6, r3           add P6, P5, P4
st r6, (r1)              st P6, (P1)
ld r6, (r11)             ld P7, (Pw)
```

When can we reuse a physical register?

Physical Register Management

Rename Table

R0	
R1	P4
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4	<R1>	p
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
Pn		

Free List

P0
P1
P3
P2

```
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)
```

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd

(LPRd requires third read port on Rename Table for each instruction)

Physical Register Management

Rename Table

R0	
R1	~~P4~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4	<R1>	p
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
P4

```
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)
```

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
~~Y~~	~~Y~~	ld	p	P7	p	#0	r1	P4	P0
Y		add	p	P0	p	#4	r3	P7	P1
Y		sub	p	P6	p	P5	r6	P5	P3
Y		add		P1		P3	r3	P1	P2

(LPRd requires third read port on Rename Table for each instruction)

Memory Dependencies

```
st r1, (r2)
ld r3, (r4)
```

When can we execute the load?

In-Order Memory Queue

- **Execute all loads and stores in program order**

=> Load and store cannot leave ROB for execution until all previous loads and stores have completed execution

- **Can still execute loads and stores speculatively, and out-of-order with respect to other instructions**

- **Stores held in store buffer until commit**

Conservative Out-of-Order Load Execution

```
st r1, (r2)
ld r3, (r4)
```

- Can execute load before store, if addresses known and r4 != r2
- Split execution of store instruction into two phases: address calculation and data write
- Each load address compared with addresses of all previous uncommitted stores *(can use partial conservative check i.e., bottom 12 bits of address)*
- Don't execute load if any previous store address not known

(MIPS R10K, 16 entry address queue)

Address Speculation

```
st r1, (r2)
ld r3, (r4)
```

- Guess that r4 != r2
- Execute load before store address known
- Need to hold all completed but uncommitted load/store addresses in program order
- If subsequently find r4==r2, squash load and *all* following instructions

  => Large penalty for inaccurate address speculation

Memory Dependence Prediction
(Alpha 21264/364)

```
st r1, (r2)
ld r3, (r4)
```

- Guess that r4 != r2 and execute load before store
- If later find r4==r2, squash load and all following instructions, but mark load instruction as *store-wait*
- Subsequent executions of the same load instruction will wait for all previous stores to complete
- Periodically clear *store-wait* bits (held in I-cache)

Improving Instruction Fetch

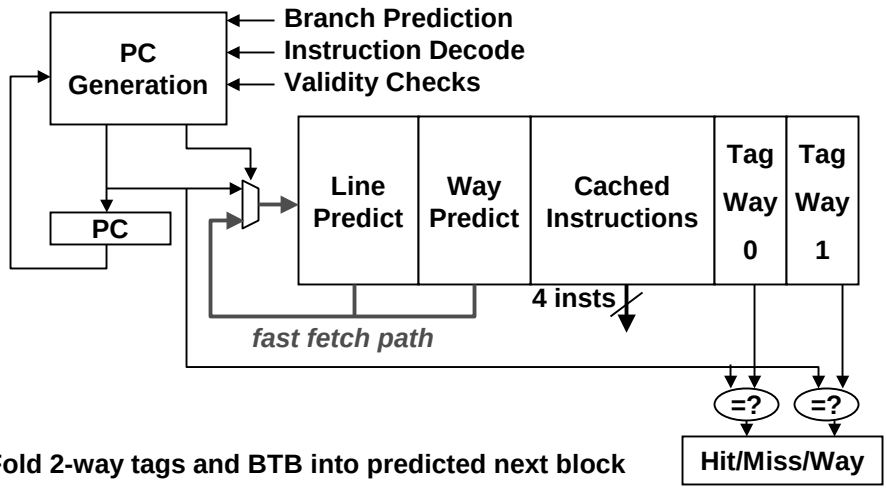
Performance of speculative out-of-order machines often limited by instruction fetch bandwidth

- speculative execution can fetch 2-3x more instructions than are committed
- mispredict penalties dominated by time to refill instruction window
- *taken branches* are particularly troublesome

Krste Asanovic
April 2, 2001
6.823, L13--27

Increasing Taken Branch Bandwidth

(Alpha 21264/364 I-Cache)

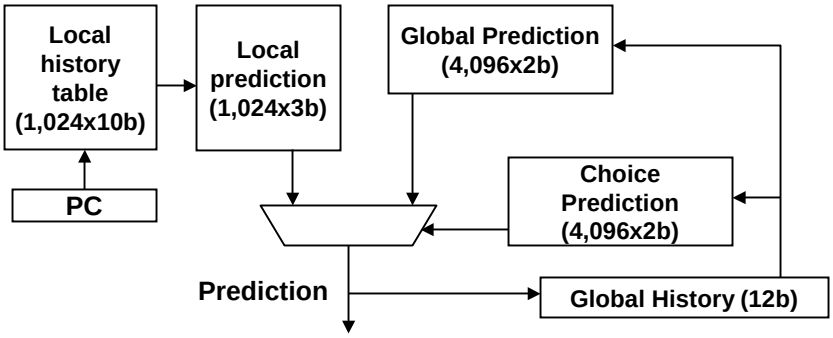


- Fold 2-way tags and BTB into predicted next block
- Take tag checks, inst. decode, branch predict out of loop
- Raw RAM speed on critical loop (1 cycle at ~1 GHz)
- 2-bit hysteresis counter per block prevents overtraining

Krste Asanovic
April 2, 2001
6.823, L13--28

Tournament Branch Predictor

(Alpha 21264/364)

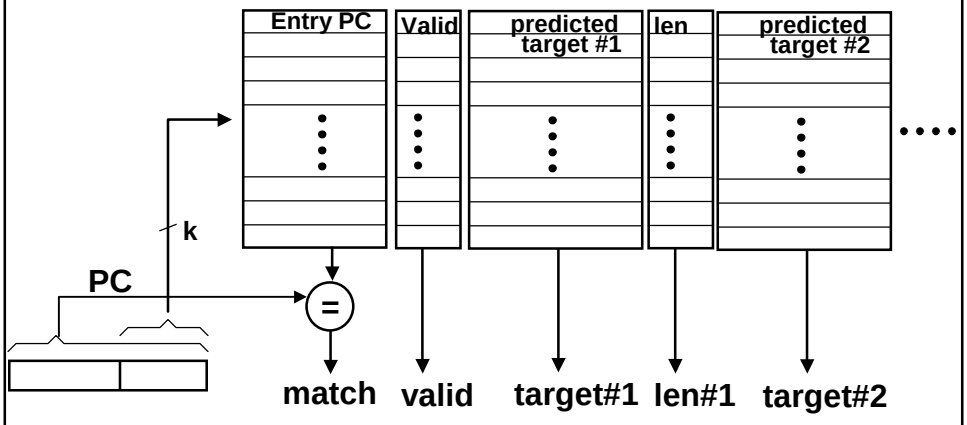


- Choice predictor learns whether best to use local or global branch history in predicting next branch
- Global history is speculatively updated but restored on mispredict
- Claim 90-100% success on range of applications

Taken Branch Limit

- **Integer codes have a taken branch every 6-9 instructions**
- **To avoid fetch bottleneck, must execute multiple taken branches per cycle when increasing performance**
- **This implies:**
  - **predicting multiple branches per cycle**
  - **fetching multiple non-contiguous blocks per cycle**

Branch Address Cache
(Yeh, Marr, Patt)



Extend BTB to return multiple branch predictions per cycle

Fetching Multiple Basic Blocks

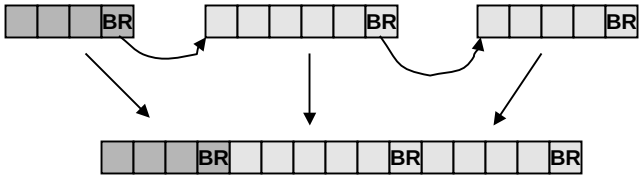
Requires either

- **multiported cache: expensive**
- **interleaving: bank conflicts will occur**

Merging multiple blocks to feed to decoders adds latency increasing mispredict penalty and reducing branch throughput

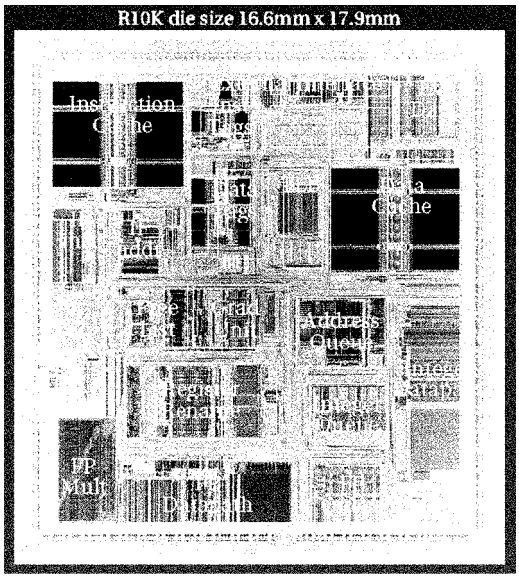
Trace Cache

Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line



- **Single fetch brings in multiple basic blocks**
- **Trace cache indexed by start address *and* next *n* branch predictions**
- **Used in Intel Pentium-4 processor to hold decoded uops**

MIPS R10000 (1995)



- 0.35µm CMOS, 4 metal layers
- Four instructions per cycle
- Out-of-order execution
- Register renaming
- Speculative execution past 4 branches
- On-chip 32KB/32KB split I/D cache, 2-way set-associative
- Off-chip L2 cache
- Non-blocking caches

Compare with simple 5-stage pipeline

- ~1.6x performance SPECint95
- ~5x CPU logic area
- ~10x design effort