CS 597d - Compiler Directed Speculation

Professor David August
Fall, 1999

Course Information

- Project Meeting 3 PM on Thursday - travel.
- Short project meeting Wednesday after class.
- Papers for next two lectures available online and outside 407.
- Volunteer for Machine Description paper - this Wednesday.
- Two volunteers needed for next (not this) Wednesday's lecture on scheduling.

Employment Opportunity

- Intel IA-64 Compiler Research Group - Jesse Fang
- Microprocessor Research Lab in Santa Clara
- Graduate Student Summer Internship (Need B.S.) or Full Time Position (Need Ph.D.)
- Investigation of techniques across the HW/SW interface for performance, throughput, reliability, scalability, compatibility, and power efficiency.

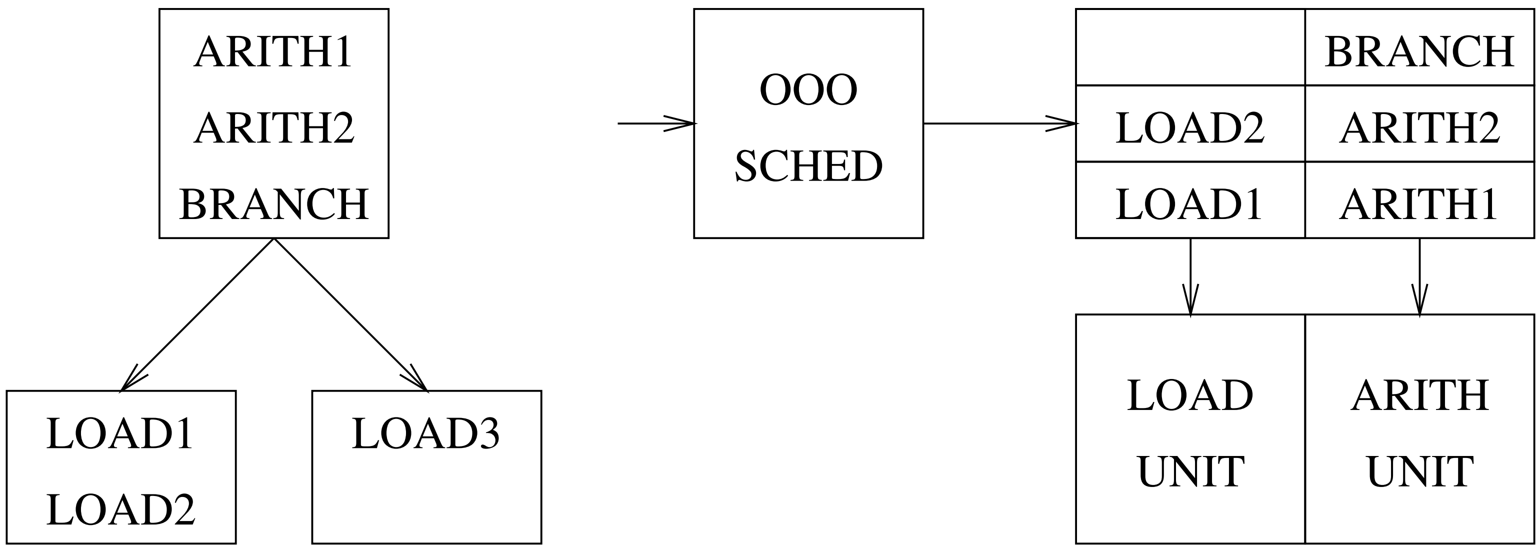
Jordan Parker	Project	Obs1	Obs2	Obs3	Obs4
Shaojie Wang	Project	Obs1	Obs2	Obs3	Obs4
Subramanian Rajagopalan	Project	Obs1		Obs3	
Grant Wallace	Project	Obs1	Obs2	Obs3	Obs4
Zhining Huang	Project	Obs1	Obs2	Obs3	
Manish Vachharajani	Project	Obs1	Obs2	Obs3	Obs4
Zhigang Hu	Project				
Hongli Zhang	L6	Obs1	Obs2	Obs3	Obs4
Zhijie Shi	L5	Obs1	Obs2	Obs3	Obs4
Xiao Yang	L6	Obs1	Obs2	Obs3	Obs4
Dongming Jiang	L5				
Cheng Liao	L4				
David Brooks	Project/L8				
Jiong Luo	Project				
Matthias Jacob					
Xianfeng Zhou					

Control Speculation

- The issue rate is increasing.
- Non-numeric applications have an average of only 4-5 instructions per basic block.
- Traditional basic block code scheduling has produced insufficient ILP.
- Memory latency is becoming a more serious problem. (latency $>$ instruction window $*$ width)
- Control speculation can solve these problems by allowing more scheduling freedom in the presence of branches.
- Control speculation allows code motion of instructions across branches.
  - Loads
  - Stores
  - Others
- It does this by breaking *control dependences*.

Hardware Speculation

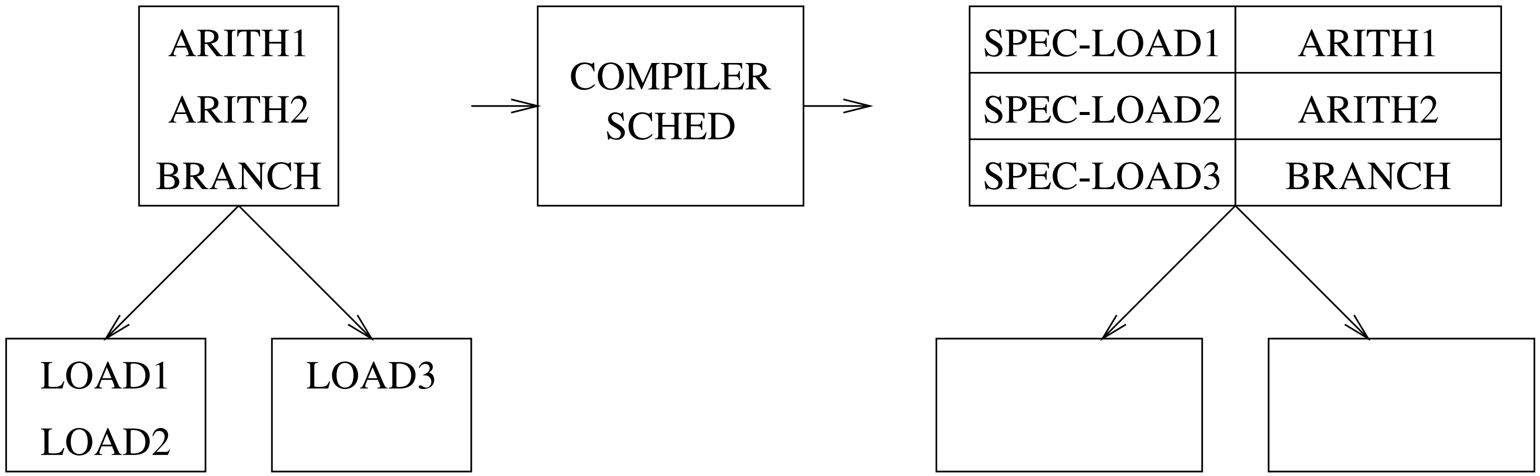
- OOO machines typically retire in order to keep incorrectly speculated instructions from modifying processor state.
- Branch is retired, then speculated instructions are retired.
- Exceptions are reported at retirement.



Limitations of HW speculation?

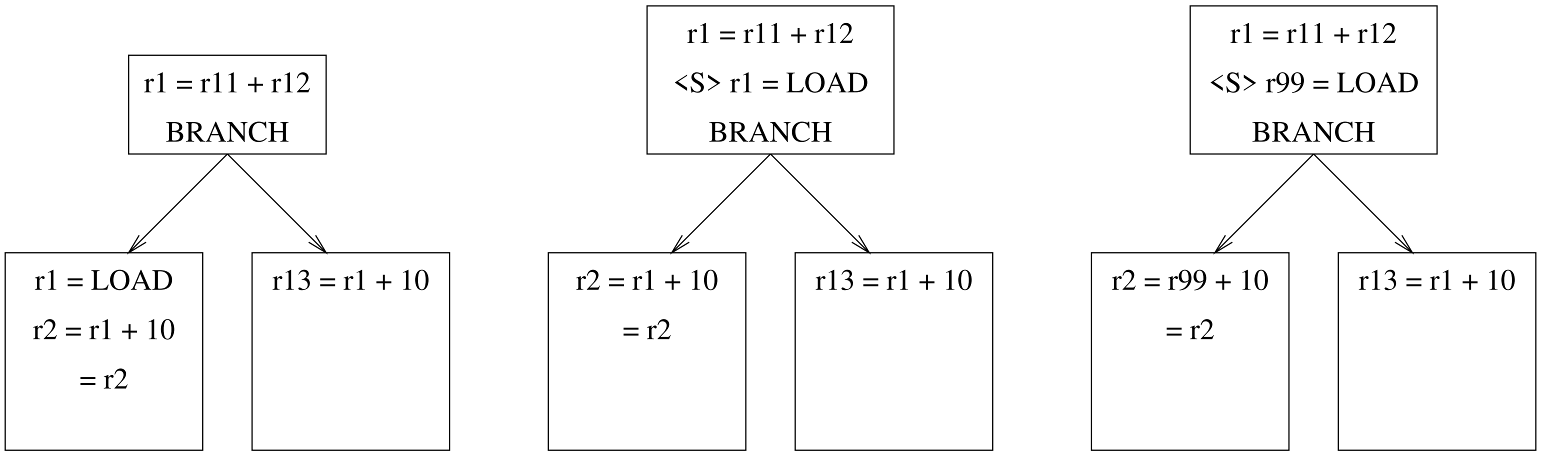
Compiler Controlled Speculation

- + Speculation not limited to single path.
- + Speculation not limited to instructions in instruction window.
- Simply moving an instruction across a branch may be dangerous.
  - Values live-out of the branch must not be changed.
  - The speculated instruction must have no side-effects during extra executions.



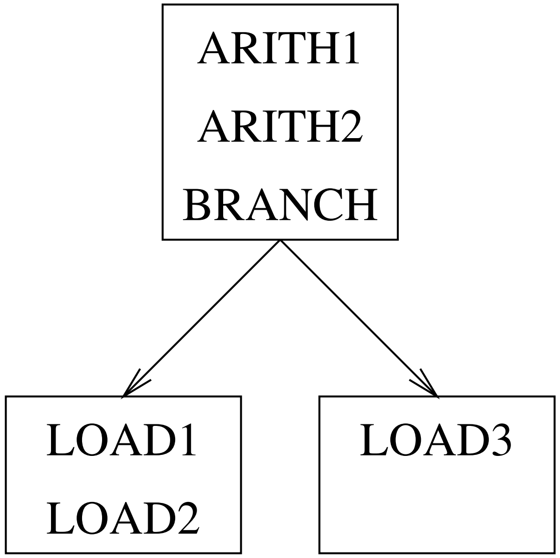
Register Live Ranges and Compiler Controlled Speculation

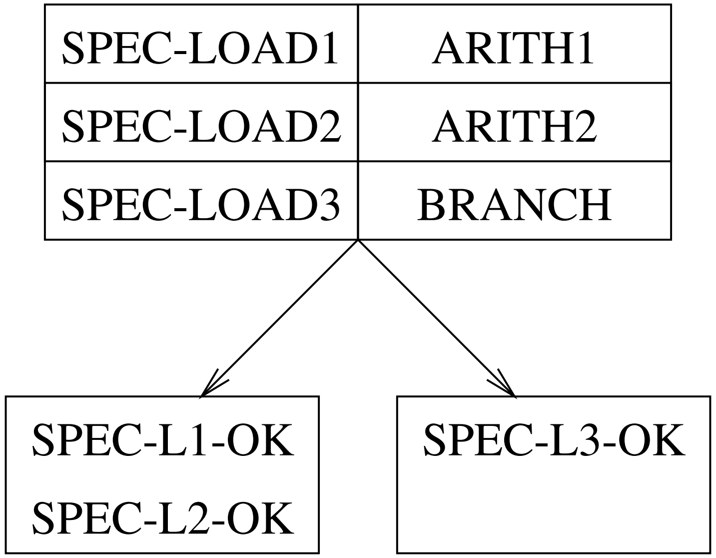
- Values live-out must not be changed.
- Hardware OOO does renaming during dynamic scheduling using tags.
- Compiler can do renaming with large number of registers.
- Large register files are important for compiler extracted ILP.



Exceptions and Compiler Controlled Speculation

- Exceptions must be the same in the original and optimized program.
- Exceptions which occur during the extra executions of speculated instructions are called "spurious".
- Spurious exceptions must not be reported - non-spurious must be reported.



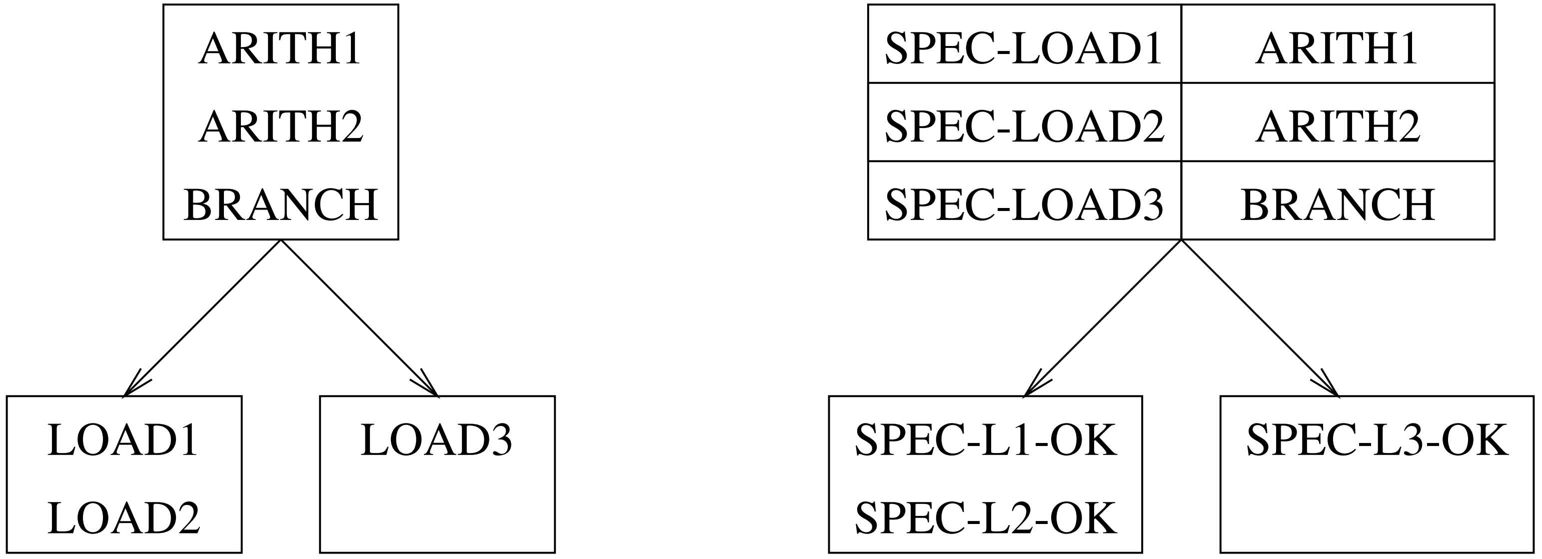


Compiler Controlled Speculation Models

- Restricted Speculation (Available using IMPACT)
  - Cannot speculate PEI's.
- General Speculation (Available using IMPACT)
  - Speculated code may terminate normally, yet produce incorrect results.
  - Exceptions are ignored.
  - PA-RISC supports this for loads - causes debugging difficulties.
- Instruction Boosting - Michael Smith (Stanford)
  - High hardware cost limits essential speculation height.
  - Register pressure not increased.
- Write-Back Suppression - Roger Bringmann (IMPACT)
  - Cannot support speculation over multiple program paths.
  - High hardware cost.
  - Bits in ISA limit speculation height.

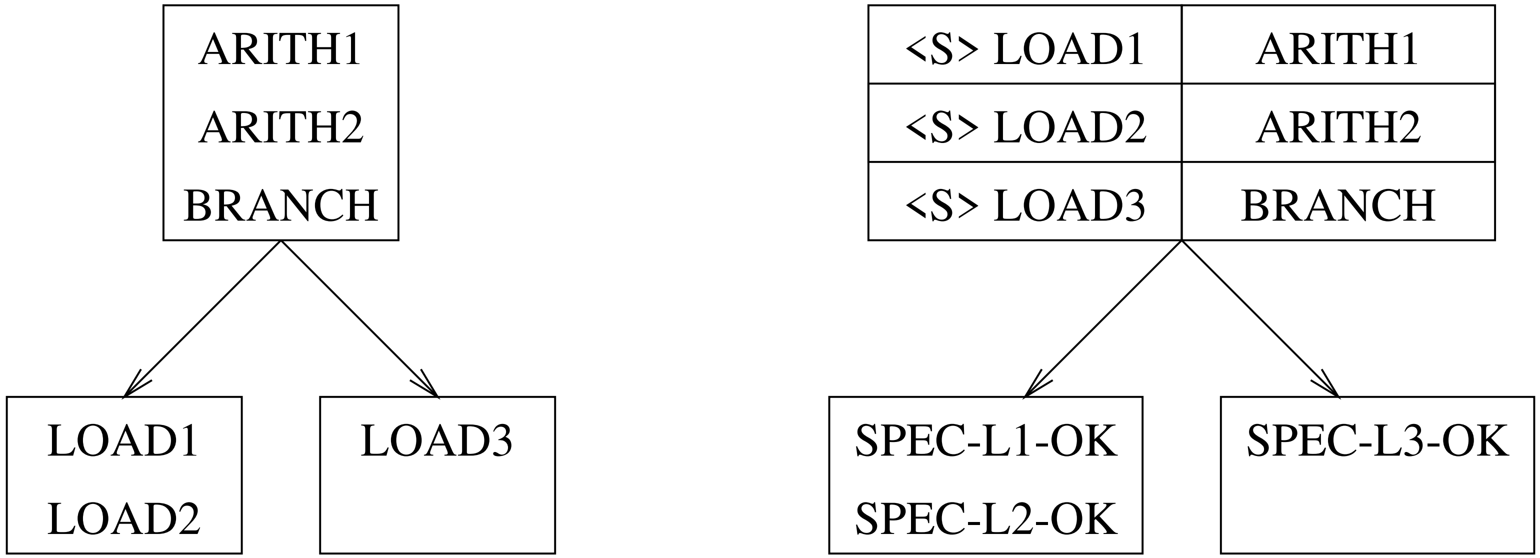
Sentinel Speculation

- Sentinel Scheduling - Scott Mahlke/Bob Rau (IMPACT/HP-LABS)
  - Inline recovery is difficult to compile for and implement in hardware.
  - Increases register pressure, reducing performance for machines with few registers.
  - Variant in IA-64



Sentinel Speculation - The S-bit

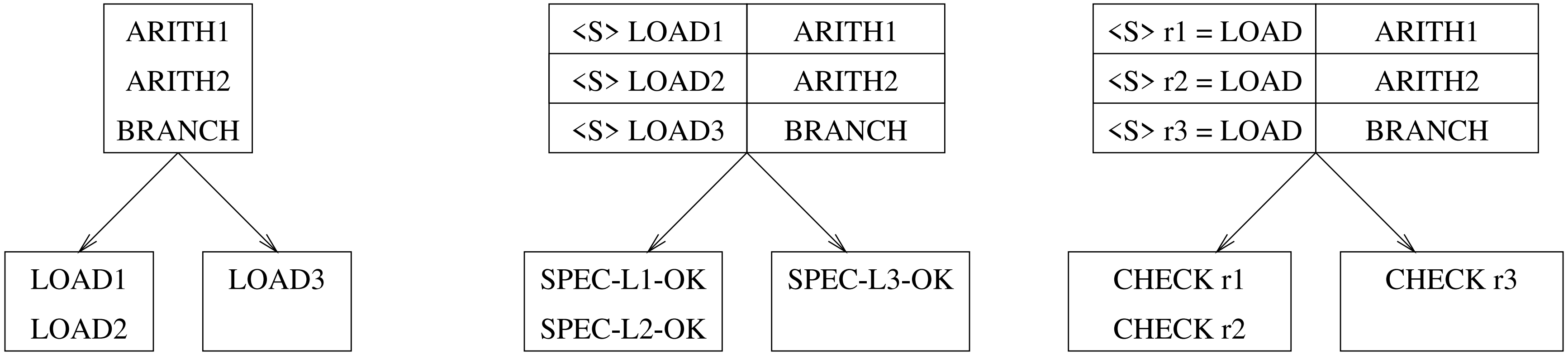
- All Potentially Excepting Instructions (PEIs) have an extra bit called the S-bit.
- The S-bit marks an instruction as speculative.



In IA-64, loads have speculative versions `(ld.s)`.

Sentinel Speculation - The Sentinel

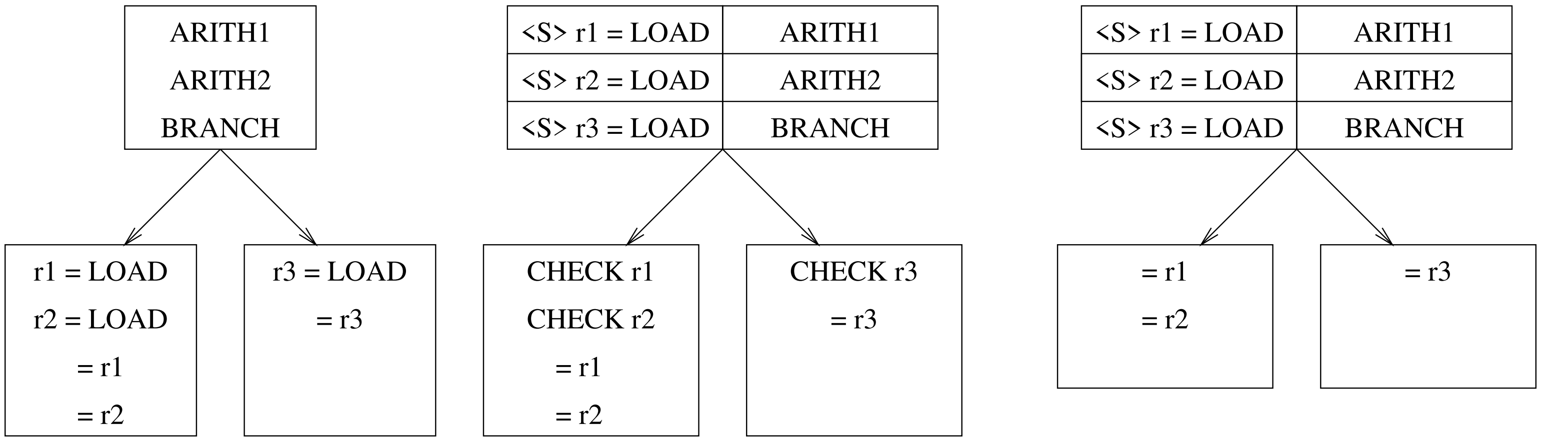
- A new instruction is created to determine whether or not a PEI should have executed.
- Checks are placed in *home blocks*.
- This "sentinel" instruction is called a *check*.
- Checks are associated with PEIs via registers.



In IA-64, checks are `chk.s (chk.s.i, chk.s.m).`

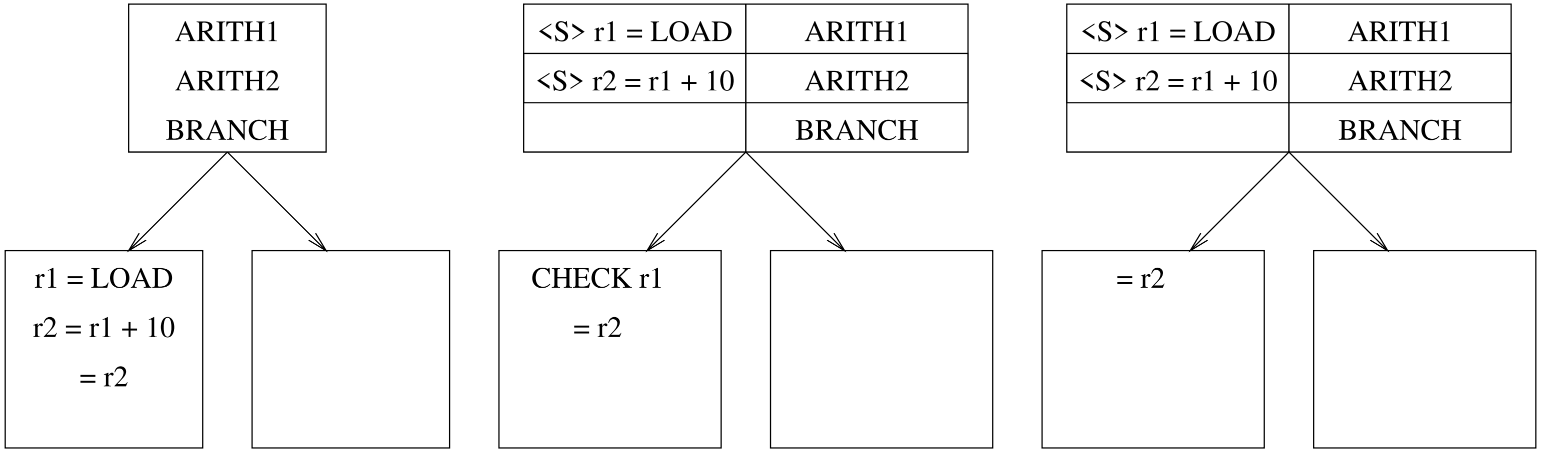
Sentinel Speculation - The E-tag

- Each register has an additional bit called the *E-tag*.
- The E-tag is set when a speculative (S-bit set) instruction “excepts”.
- Any non-speculative (S-bit clear) instruction which sources a register with its E-tag set should initiate exception reporting.
  - Explicit *check* instructions.
  - Implicit sentinels.



E-tag Propagation

- In order to speculate chains of instructions, E-tags should propagate.
- Reporting should not occur for speculated instructions which source a set E-tag.



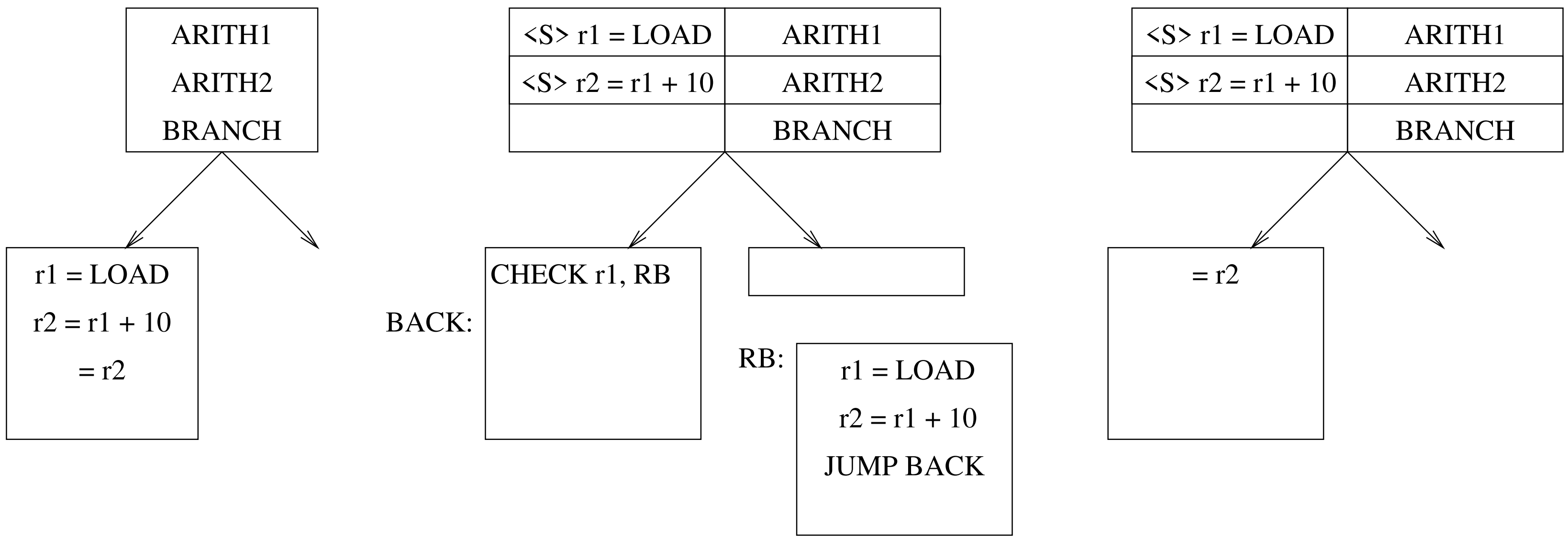
Sentinel Speculation - The E-tag

$spec$	$src(I).except\_tab$	$I\ causes\ except$	$dest(I).except\_tab$	$dest(I).data$	$except.signal$
0	0	0	0	result of *I*	none
0	0	1	0	-	yes, except. pc = pc of *I*
0	1	0	0	-	yes, except. pc = pc *src(I).data*
0	1	1	0	-	yes, except. pc = pc *src(I).data*
1	0	0	0	result of *I*	none
1	0	1	1	pc of *I*	none
1	1	0	1	*src(I).data*	none
1	1	1	1	*src(I).data*	none

In IA-64, the E-tag is termed the NaT-bit

Recovery

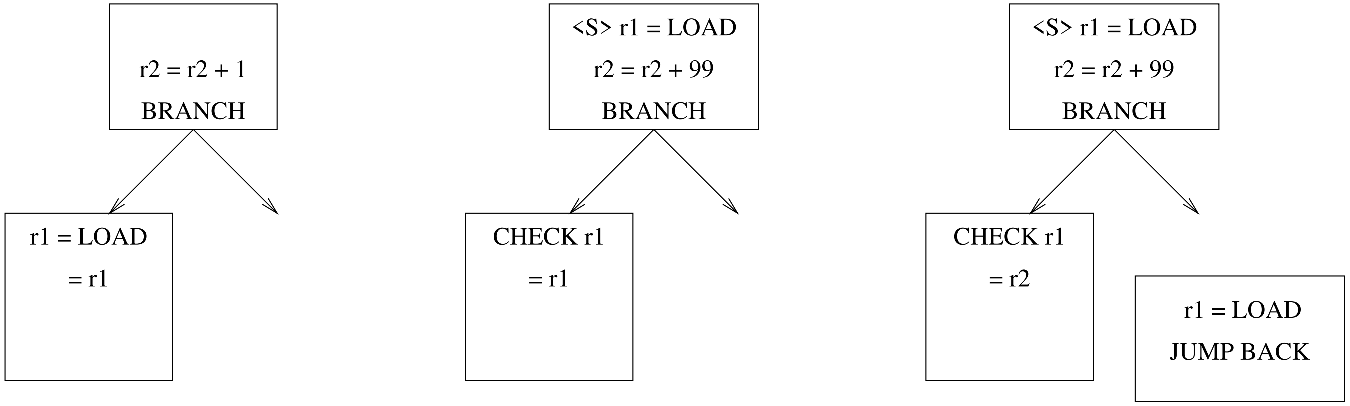
- When an exception occurs, exceptions must be accurately reported.
- This can be done by re-execution of the excepting instructions
  - Exceptions during re-execution are reported immediately.
  - Code executed may be *inline* or *recovery block*



Inline Recovery

spec	*src(I).except_tab*	*I causes except*	*dest(I).except_tab*	*dest(I).data*	*except.signal*
0	0	0	0	result of *I*	none
0	0	1	0	-	yes, except. pc = pc of *I*
0	1	0	0	-	yes, except. pc = pc *src(I).data*
0	1	1	0	-	yes, except. pc = pc *src(I).data*
1	0	0	0	result of *I*	none
1	0	1	1	pc of *I*	none
1	1	0	1	*src(I).data*	none
1	1	1	1	*src(I).data*	none

Self-anti-dependent instructions cannot exist within an inline recovery region.



IA-64 supports recovery blocks.

Recovery Blocks

- Checks always have an explicit target to the recovery block.
- PC propagation not necessary.
- Problems with Sentinel Scheduling are eliminated.
  - To handle multiple exceptions in order, the sentinels need to be in original order, reducing scheduling freedom.
  - Self-anti-dependent instructions cannot exist within an inline recovery region.
- With recovery blocks:
  - The sentinels can be in any order and shared.
  - A single recovery block is necessary for all speculated instructions from a given home block.
  - The re-execution of non-speculative versions of the PEIs are trivially placed in original program order with recovery block.

Recovery blocks are much easier to handle in hardware and compiler.

Recovery Blocks

Hardware is much simpler.

$spec$	$src(I).except\_tab$	I $causes$ $except$	$dest(I).except\_tab$	$dest(I).data$	$except.signal$
0	0	0	0	result of I	none
0	0	1	0	-	yes, except.
0	1	0	0	-	none
0	1	1	0	-	none
1	0	0	0	result of I	none
1	0	1	1	-	none
1	1	0	1	-	none
1	1	1	1	-	none

However...

- Checks must be explicit - larger code size.
- Recover blocks - larger code size.

Solutions exist which should be studied...

Live Range Extension

- For both recovery mechanisms, live ranges must be extended from PEI to Sentinel.
  - For all speculated instructions.
  - For all branch conditions.
  - For all non-speculated instructions in some models.
- For inline recovery, no self-anti-dependent instructions can exist between PEI and Sentinel.
  - Split self-anti-dependent instructions into two instructions.
  - Code growth/scheduling freedom/resource saturation.

An Example

1 $beq\ r5, 1, 50$
2 $r10 \leftarrow mem(r11)$
3 $r3 \leftarrow r2/r1$
4 $r6 \leftarrow r6 + r3$
5 $r9 \leftarrow r10 + r6$
6 $mem(r7) \leftarrow r6$
7 $bne\ r10, 0, 100$

A. Original Code Sequence

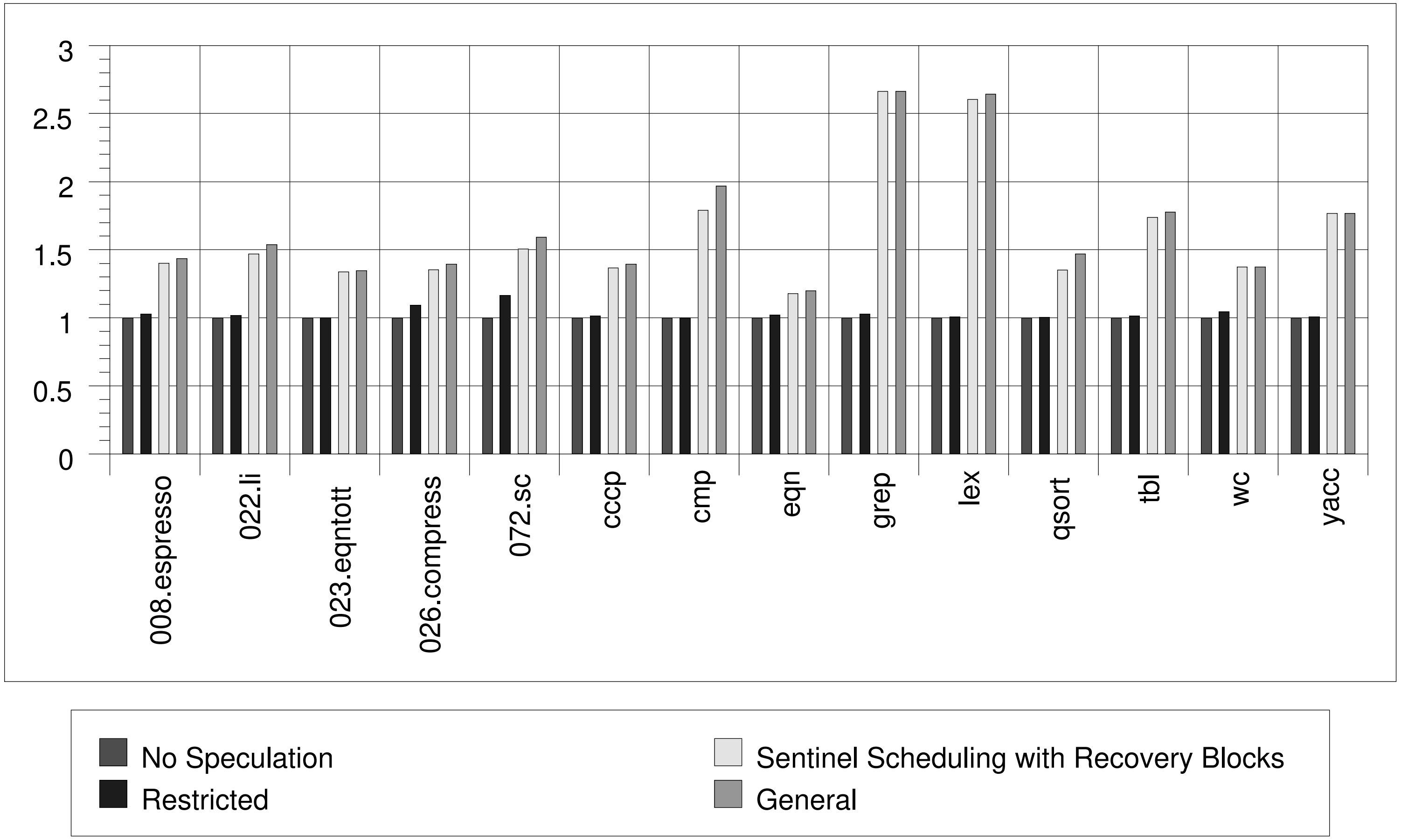
2 $\langle S \rangle\ r10 \leftarrow mem(r11)$
3 $\langle S \rangle\ r3 \leftarrow r2/r1$
4 $\langle S \rangle\ r6 \leftarrow r6 + r3$
5 $\langle S \rangle\ r9 \leftarrow r10 + r6$
1 $beq\ r5, 1, 50$
6 $mem(r7) \leftarrow r6$
8 $check\ r9, inst2'$
7 $bne\ r10, 0, 100$

B. Scheduled Code Sequence

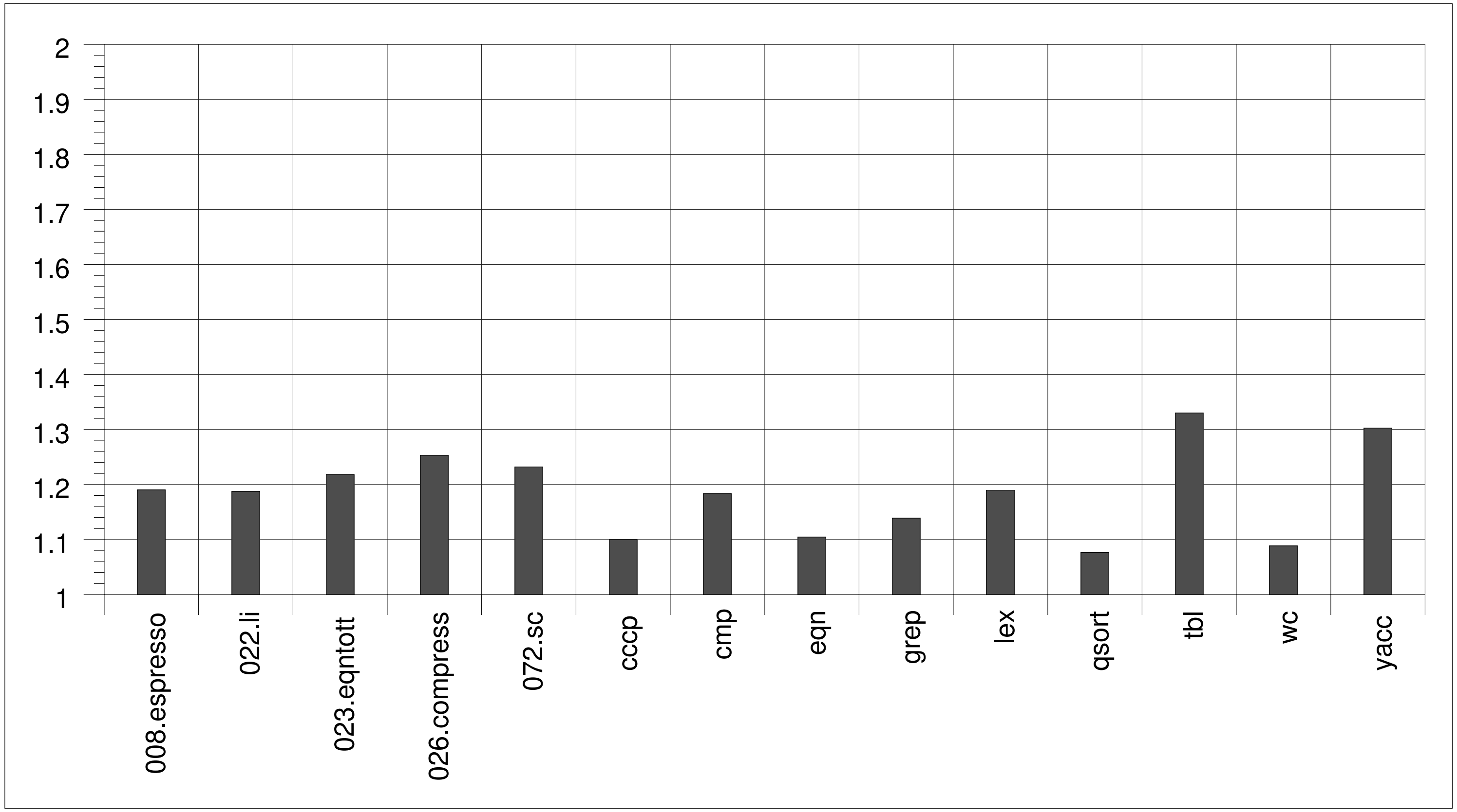
$2'$ $r10 \leftarrow mem(r11)$
$3'$ $r3 \leftarrow r2/r1$
$4'$ $r6 \leftarrow r6 + r3$
$5'$ $r9 \leftarrow r10 + r6$
$6'$ $jump\ inst7$

C. Recovery Block

Performance



Code Growth



2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
008.espresso
022.li
023.eqntott
026.compress
072.sc
cccp
cmp
eqn
grep
lex
qsort
tbl
wc
yacc

Speculating Stores

- Store buffer
  - Circular queue holds N pending stores.
  - Exception occur immediately. (page fault or access violation)
  - Data forwarded for loads with matching address.
- Speculative store operation
  - Speculative stores are probationary. (confirmation bit clear)
  - E-tag and PC are held in buffer.
  - Additional *confirm_store* instruction which checks E-tag.
  - Cancellation mechanism necessary - for example an unexpected branch direction.
  - Exception recovery may not be in original order.
  - Store and confirm may not be more than $N-1$ stores apart.
  - N becomes architectural and must be available to the scheduler.

Speculating Stores - Reality

- Speculating stores involves costly hardware.
- Speculation may only be across one path, unless more complicated mechanisms are employed.
- Speculating stores is not very important.
  - Stores are not on critical path.
  - Load - computation - Store
  - Store - Load dependences
    - Dependence is broken with data speculation.
    - Optimization can transform dependence as well.

IA-64 does not support store speculation.