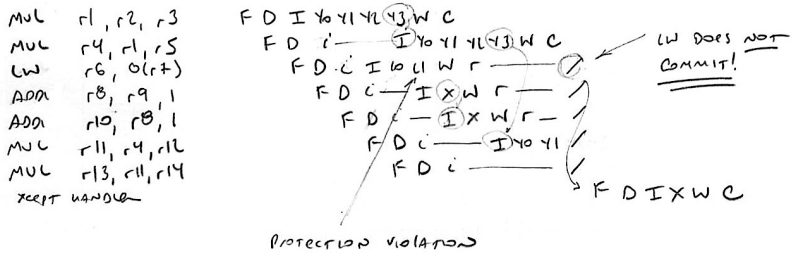
ECE 4750 Computer Architecture, Fall 2022

Topic 11: Advanced Processors Speculative Execution

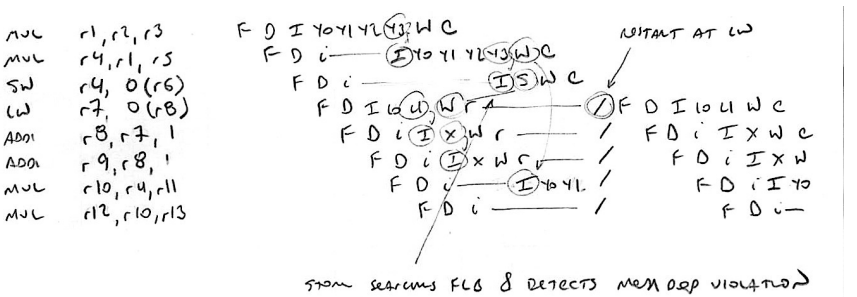
School of Electrical and Computer Engineering
Cornell University

revision: 2022-11-30-12-02

1. Speculative Execution with Late Recovery



- Every instruction is actually speculative because an older in-flight instruction might cause an exception
- We recover from exceptions at the commit point (C-stage) which is late in the pipeline



- With out-of-order load/store issue, loads (and dependent instructions) are also speculative
- We recover from incorrect speculation in the C stage which is late in the pipeline

- Branches also require speculative execution
- Recover mispredictions late in the pipeline?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a: `lw x1, 0(x2)`																
b: `mul x3, x1, x4`																
c: `sw x3, 0(x5)`																
d: `addi x2, x2, 4`																
e: `addi x5, x5, 4`																
f: `addi x6, x6, -1`																
g: `bne x6, x0, loop`																

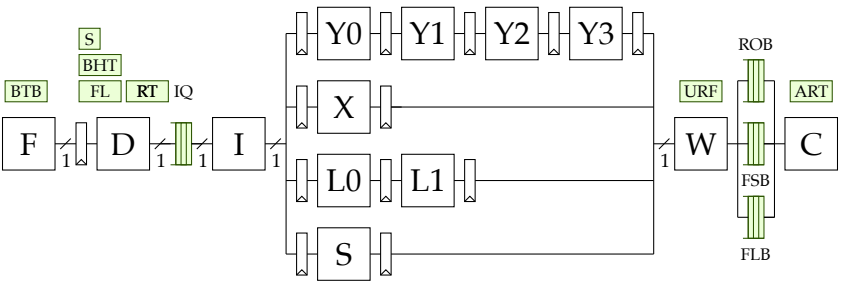
- Branches are far more common than exceptions and memory-dependence violations
- Accurate branch prediction helps, but some branches are just inherently difficult to predict
- Key Idea: Recover from branch mispredictions as soon as possible

2. Speculative Execution with Early Recovery

We will explore early recovery in two steps:

- Adding speculative bits
- Adding rename-table snapshots

2.1. Adding Speculative Bits



- Add a speculative bit to the IQ, ROB, FSB, FLB, and functional units
- Add a speculative mode bit in the D stage
- In D stage for a branch
  - Set speculative mode bit
  - All inst after branch carry speculative bit into IQ, ROB, FSB, LB, func units
- In X stage for a correctly predicted branch
  - Broadcast clear speculative bit from X stage to all data structures
- In X stage for a incorrectly predicted branch
  - Broadcast squash signal from X stage to all of these data structures
  - Each data structure invalidates entry/inst for which speculative bit is set
  - Start fetching from correct address
- Multiple speculative bits enable multiple spec branches in flight
  - Given instruction can be squashed by multiple branches
  - Treat multiple speculative bits as “branch mask”

Do not copy ARF into PRF on branch misprediction recovery

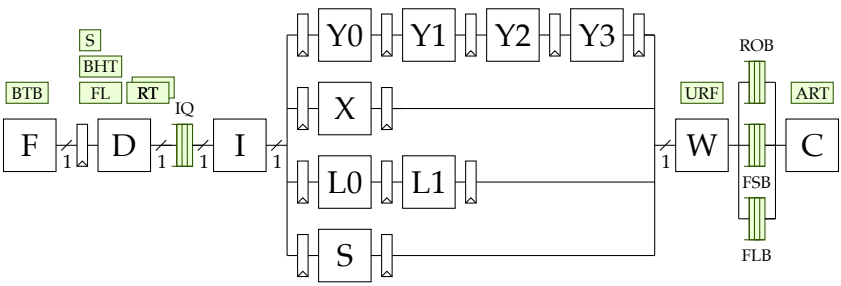
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a : `addi x1, x2, 1`																
b : `branch L1`																
c : `addi x1, x3, 1`																
d : `opA`																
e : `opB`																
f : `opC`																
g : `opD`																
h : `L1:addi x4, x1, 1`																

Copy ARF into PRF on branch misprediction recovery

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a : `addi x1, x2, 1`																
b : `addi x1, x3, 1`																
c : `addi x4, x1, 1`																
d : `branch L1`																
e : `opA`																
f : `opB`																
g : `opC`																
h : `opD`																
i : `L1:addi x5, x6, 1`																

- Need to make copy of “precise” ARF in D on every branch ...
- ... but ARF is not precise in D
- Need “view” of what precise ARF would be in D on every branch ...
- ... this is the rename table!

2.2. Adding Rename-Table Snapshots



- Add a speculative bit to the IQ, ROB, FSB, FLB, and functional units
- Add a speculative mode bit in the D stage
- Add a rename table snapshot in the D stage
- In D stage for a branch
  - Set speculative mode bit
  - All inst after branch carry speculative bit into IQ, ROB, FSB, LB, func units
  - Create a RT snapshot to save "view" of precise ARF for branch
- In X stage for a correctly predicted branch
  - Broadcast clear speculative bit from X stage to all data structures
- In X stage for a incorrectly predicted branch
  - Broadcast squash signal from X stage to all of these data structures
  - Each data structure invalidates entry/inst for which speculative bit is set
  - Restore RT from snapshot
  - Start fetching from correct address
- Need multiple speculative bits and multiple snapshots to support multiple speculative branches in flight

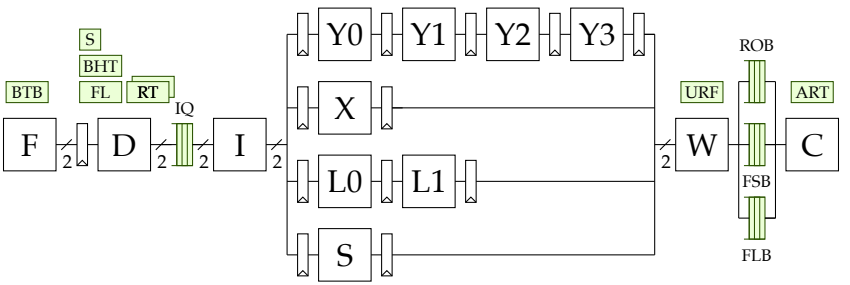
RT snapshots squash speculative state

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a : `addi x1, x2, 1`																
b : `branch L1`																
c : `addi x1, x3, 1`																
d : `opA`																
e : `opB`																
f : `opC`																
g : `opD`																
h : `L1:addi x4, x1, 1`																

RT snapshots prevent overwriting non-speculative state

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a : `addi x1, x2, 1`																
b : `addi x1, x3, 1`																
c : `addi x4, x1, 1`																
d : `branch L1`																
e : `opA`																
f : `opB`																
g : `opC`																
h : `opD`																
i : `L1:addi x5, x6, 1`																

3. Complete Out-of-Order Superscalar TinyRV1 Processor



- Superscalar execution: two-way every stage, aligned fetch blocks
- Out-of-order execution: IO2L with IQ and ROB
- Register renaming: pointer-based scheme with URF and ART
- Memory disambiguation: OOO load/store issue with FSB and FLB
- Branch prediction: BTB with generalized two-level BHT
- Speculative execution: speculative bits with rename table snapshots

Vector-Vector Add Microbenchmark

Microarchitecture	cycles/itr	actual CPI	actual IPC	peak IPC
In-Order Single-Issue TinyRV1	12	1.33	0.75	1
In-Order Dual-Issue TinyRV1	10	1.11	0.90	2
Out-of-Order Dual-Issue TinyRV1	5	0.55	1.80	2