CSE 490/590 Computer Architecture

ILP II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

- Register renaming
  - Overcoming the restriction caused by the # of registers
  - Reorder buffer & renaming table
- Precise interrupts
  - It must appear as if an interrupt has occurred in-between two instructions

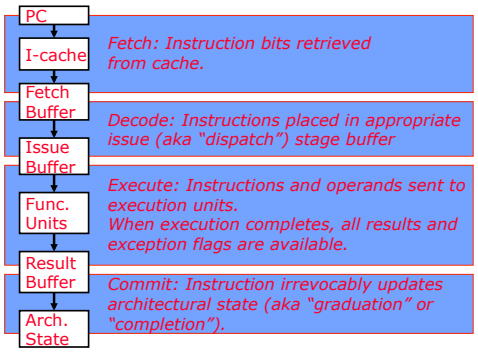
Precise Interrupts

It must appear as if an interrupt is taken between two instructions (say I_i and I_{i+1})

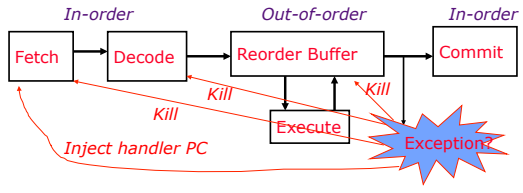
- the effect of all instructions up to and including I_i is totally complete
- no effect of any instruction after I_i has taken place

The interrupt handler either aborts the program or restarts it at I_{i+1} .

Phases of Instruction Execution



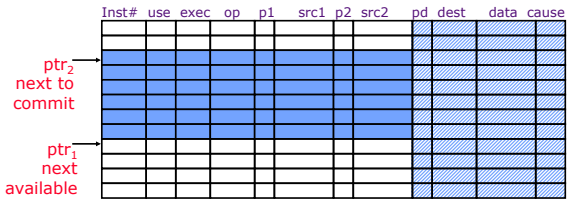
In-Order Commit for Precise Exceptions



- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (⇒ out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory, is in-order

Temporary storage needed to hold results before commit (shadow registers and store buffers)

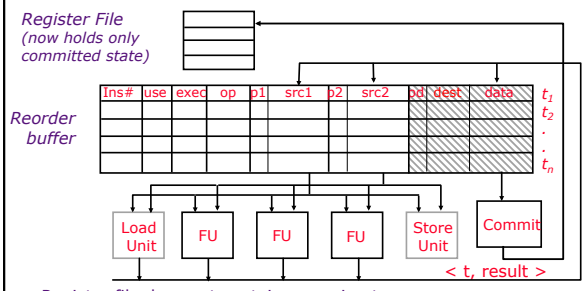
Extensions for Precise Exceptions



Reorder buffer

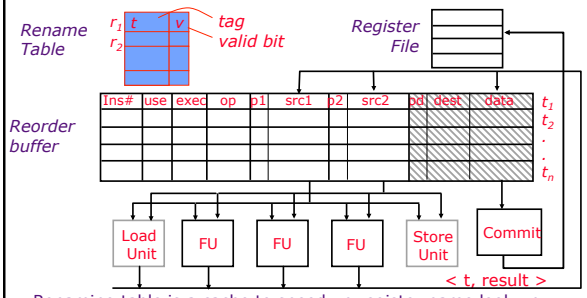
- add <pd, dest, data, cause> fields in the instruction template
- commit instructions to reg file and memory in program order ⇒ buffers can be maintained circularly
- on exception, clear reorder buffer by resetting ptr_1=ptr_2
  (stores must wait for commit before updating memory)

Rollback and Renaming



Register file does not contain renaming tags any more.
How does the decode stage find the tag of a source register?
Search the "dest" field in the reorder buffer

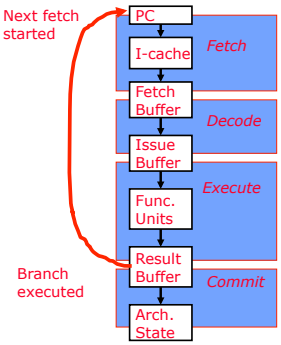
Renaming Table



Renaming table is a cache to speed up register name look up.
It needs to be cleared after each exception taken.
When else are valid bits cleared? *Control transfers*

Control Flow Penalty

Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution !



MIPS Branches and Jumps

Each instruction fetch depends on one or two pieces of information from the preceding instruction:

1) Is the preceding instruction a taken branch?

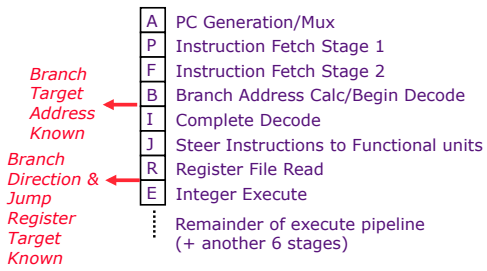
2) If so, what is the target address?

Instruction	*Taken known?*	*Target known?*
J	After Inst. Decode	After Inst. Decode
JR	After Inst. Decode	After Reg. Fetch
BEQZ/BNEZ	After Reg. Fetch*	After Inst. Decode

*Assuming zero detect on register read

Branch Penalties in Modern Pipelines

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



Reducing Control Flow Penalty

Software solutions

- *Eliminate branches - loop unrolling*
  Increases the run length
- *Reduce resolution time - instruction scheduling*
  Compute the branch condition as early as possible (of limited value)

Hardware solutions

- Find something else to do - *delay slots*
  Replaces pipeline bubbles with useful work (requires software cooperation)
- *Speculate - branch prediction*
  Speculative execution of instructions beyond the branch

CSE 490/590 Administrivia

- Project 1 & midterm grading mostly done
  - Will distribute on Wed
  - Regrading -> Jangyoung
- Project 2
  - Start early!

Branch Prediction

Motivation:

Branch penalties limit performance of deeply pipelined processors

Modern branch predictors have high accuracy (>95%) and can reduce branch penalties significantly

Required hardware support:

Prediction structures:

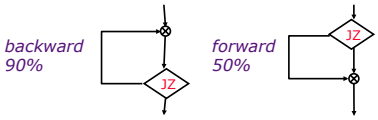
- Branch history tables, branch target buffers, etc.

Mispredict recovery mechanisms:

- *Keep result computation separate from commit*
- Kill instructions following branch in pipeline
- Restore state to state following branch

Static Branch Prediction

Overall probability a branch is taken is ~60-70% but:



ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110
bne0 *(preferred taken)* beq0 *(not taken)*

ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64
typically reported as ~80% accurate

Dynamic Branch Prediction

learning based on past behavior

Temporal correlation

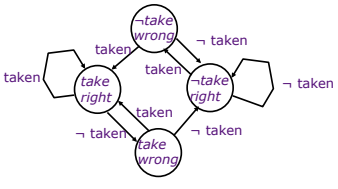
The way a branch resolves may be a good predictor of the way it will resolve at the next execution

Spatial correlation

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

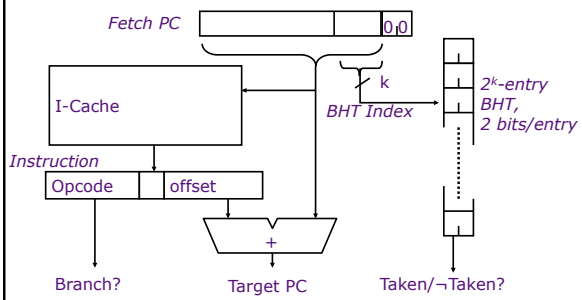
Branch Prediction Bits

- Assume 2 BP bits per instruction
- Change the prediction after two consecutive mistakes!



BP state:
(*predict* take/¬take) x (*last prediction* right/wrong)

Branch History Table



4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252