Microprocessor Microarchitecture

Instruction Fetch

Lynn Choi

Dept. Of Computer and Electronics Engineering

Instruction Fetch w/ branch prediction

- ***On every cycle, 3 accesses are done in parallel***
  - Instruction cache access
  - Branch target buffer access
    - If hit, determines that it is a branch and provides target address
    - Else, use fall-through address (PC+4) for the next sequential access
  - Branch prediction table access
    - If taken, instructions after the branch are not sent to back end and next fetch starts from target address
    - If not taken, next fetch starts from fall-through address

Motivation

- ***Wider issue demands higher instruction fetch rate***
- ***However, Ifetch bandwidth limited by***
  - Basic block size
    - Average block size is 4 ~ 5 instructions
    - Need to increase basic block size!
  - Branch prediction hit rate
    - Cost of redirecting fetching
    - *More accurate prediction* is needed
  - Branch throughput
    - *Multiple branch prediction per cycle* is necessary for wide-issue superscalar!
      - Can fetch multiple *contiguous* basic blocks
      - The number of instructions between taken branches is 6 ~ 7
      - Limited by instruction cache line size
  - Taken branches
    - *Fetch mechanism for non-contiguous basic blocks*
  - Instruction cache hit rate
    - Instruction prefetching

Solutions

- ***Solutions***
  - Increase basic block size (using a compiler)
    - Trace scheduling, superblock scheduling, predication
  - Hardware mechanism to fetch multiple non-consecutive basic blocks are needed!
    - Multiple branch predictions per cycle
    - Generate fetch addresses for multiple basic blocks
    - Non-contiguous instruction alignment
      - Need to *fetch and align multiple noncontiguous basic blocks* and pass them to the pipeline

Current Work

- ***Existing schemes to fetch multiple basic blocks per cycle***
  - Branch address cache + multiple branch prediction - Yeh
    - Branch address cache
      - Natural extension of branch target buffer
      - Provides the starting addresses of the next several basic blocks
    - Interleaved instruction cache organization to fetch multiple basic blocks per cycle
  - Trace cache - Rotenberg
    - Caching of dynamic instruction sequences
      - Exploit locality of dynamic instruction streams, eliminating the need to fetch multiple non-contiguous basic blocks and the need to align them to be presented to the pipeline

Branch Address Cache

Yeh & Patt

- ***Hardware mechanism to fetch multiple non-consecutive basic blocks are needed!***
  - Multiple branch prediction per cycle using two-level adaptive predictors
  - Branch address cache to generate fetch addresses for multiple basic blocks
  - Interleaved instruction cache organization to provide enough bandwidth to supply multiple non-consecutive basic blocks
  - Non-contiguous instruction alignment
    - Need to *fetch and align multiple non-contiguous basic blocks* and pass them to the pipeline

Multiple Branch Predictions

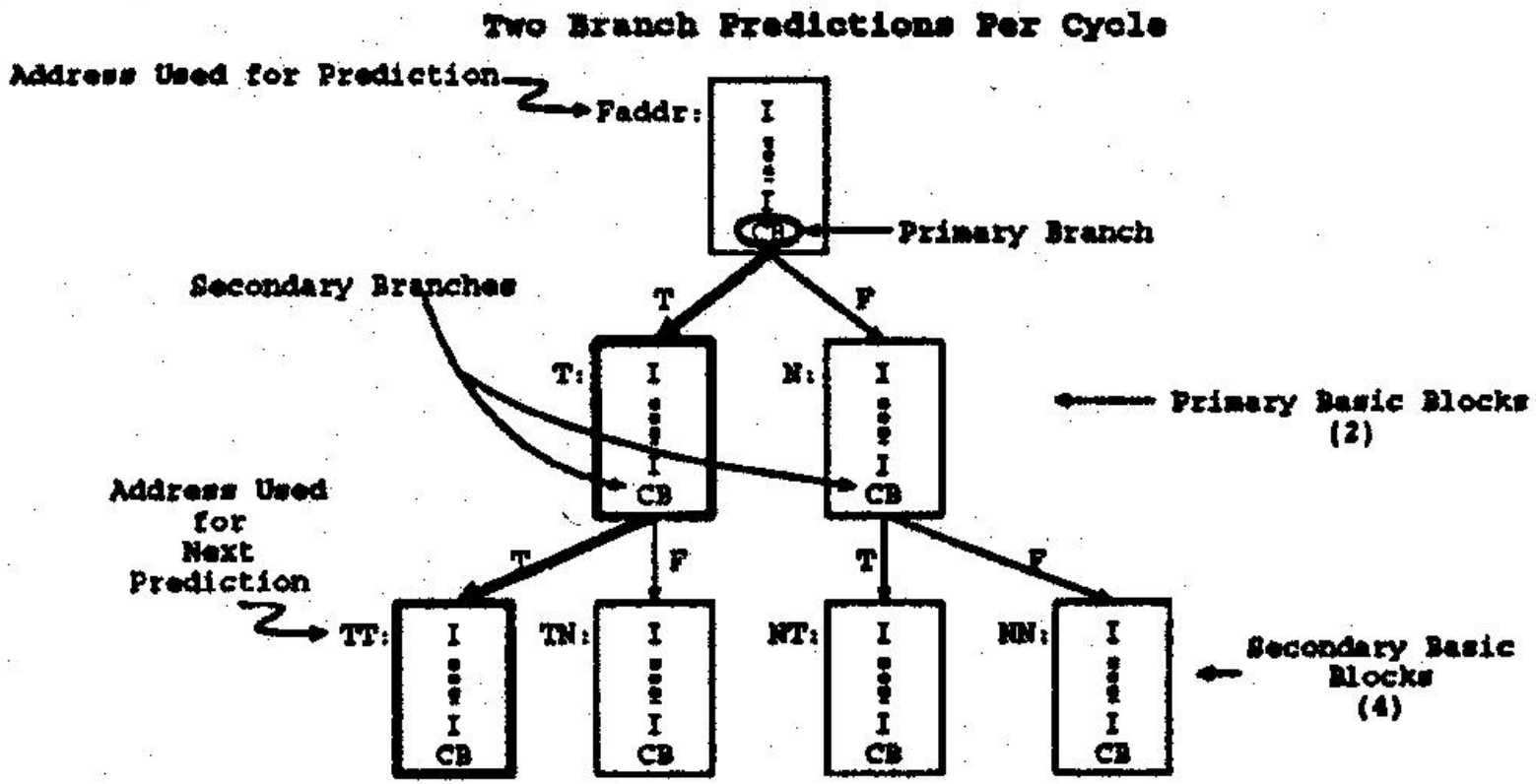

Figure 2: Identification of the primary and secondary branches, and the primary and secondary basic blocks.

Multiple Branch Predictor

- ***Variations of global schemes are proposed***
  - Multiple Branch Global Adaptive Prediction using a Global Pattern History Table (MGAg)
  - Multiple Branch Global Adaptive Prediction using a Per-Set Pattern History Table (MGAs)
- ***Multiple branch prediction based on local schemes***
  - Require more complicated BHT access due to sequential access of primary/secondary/tertiary branches

Multiple Branch Predictors

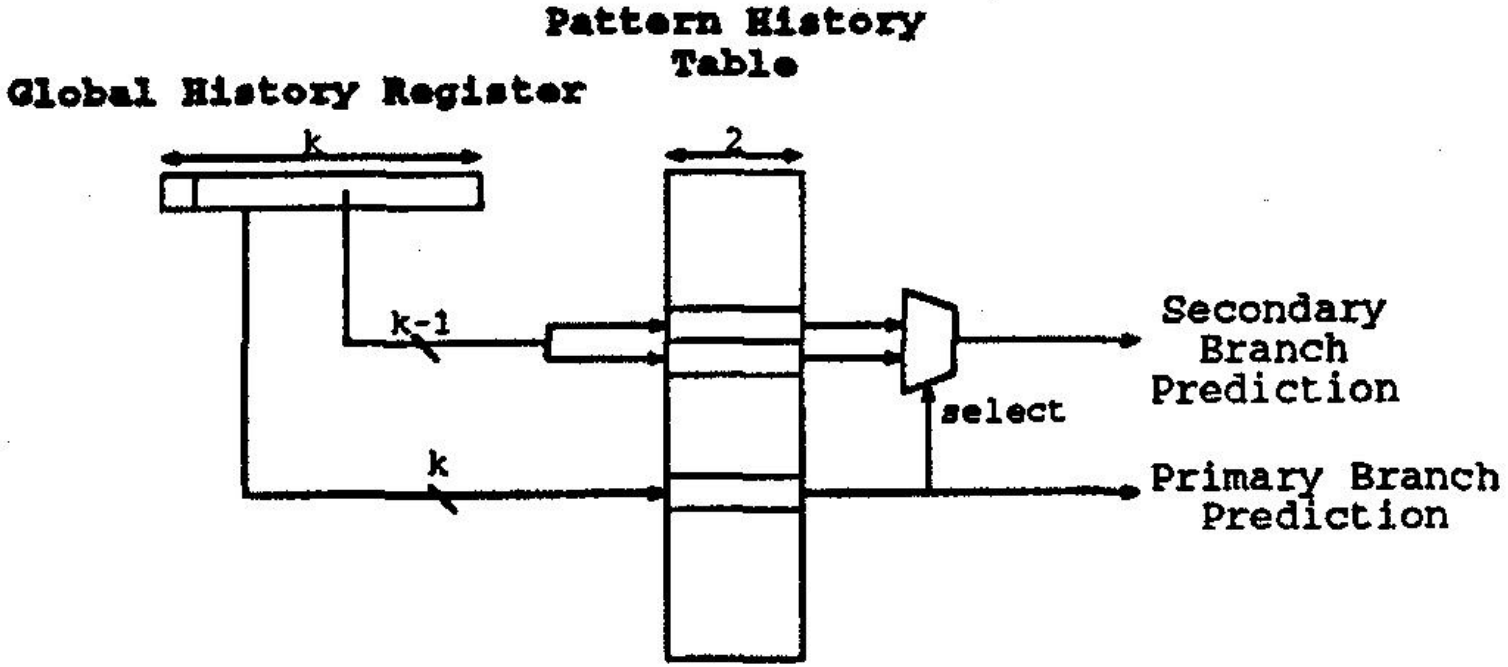

Figure 3: Algorithm to make 2 branch predictions from a single branch history register.

Branch Address Cache

- ***Only a single fetch address is used to access the BAC which provides multiple target addresses***
  - For each prediction level L, BAC provides 2^L of target address and fall-through address
    - For example, 3 branch predictions per cycle, BAC provides 14 (2 + 4 + 8) target addresses
    - For 2 branch predictions per cycle, TAC provides
      - TAG
      - Primary_valid, Primary_type
      - Taddr, Naddr
      - ST_valid, ST_type, SN_valid, SN_type
      - TTaddr, TNaddr, SNaddr, NNaddr

ICache for Multiple BB Access

- ***Two alternatives***
  - Interleaved cache organization
    - As long as there is no bank conflict
    - Increasing the number of banks reduces conflicts
  - Multi-ported cache
    - Expensive
- ***ICache miss rate increases***
  - Since more instructions are fetched each cycle, there are fewer cycles between Icache misses
    - Increase associativity
    - Increase cache size
    - Prefetching

Fetch Performance

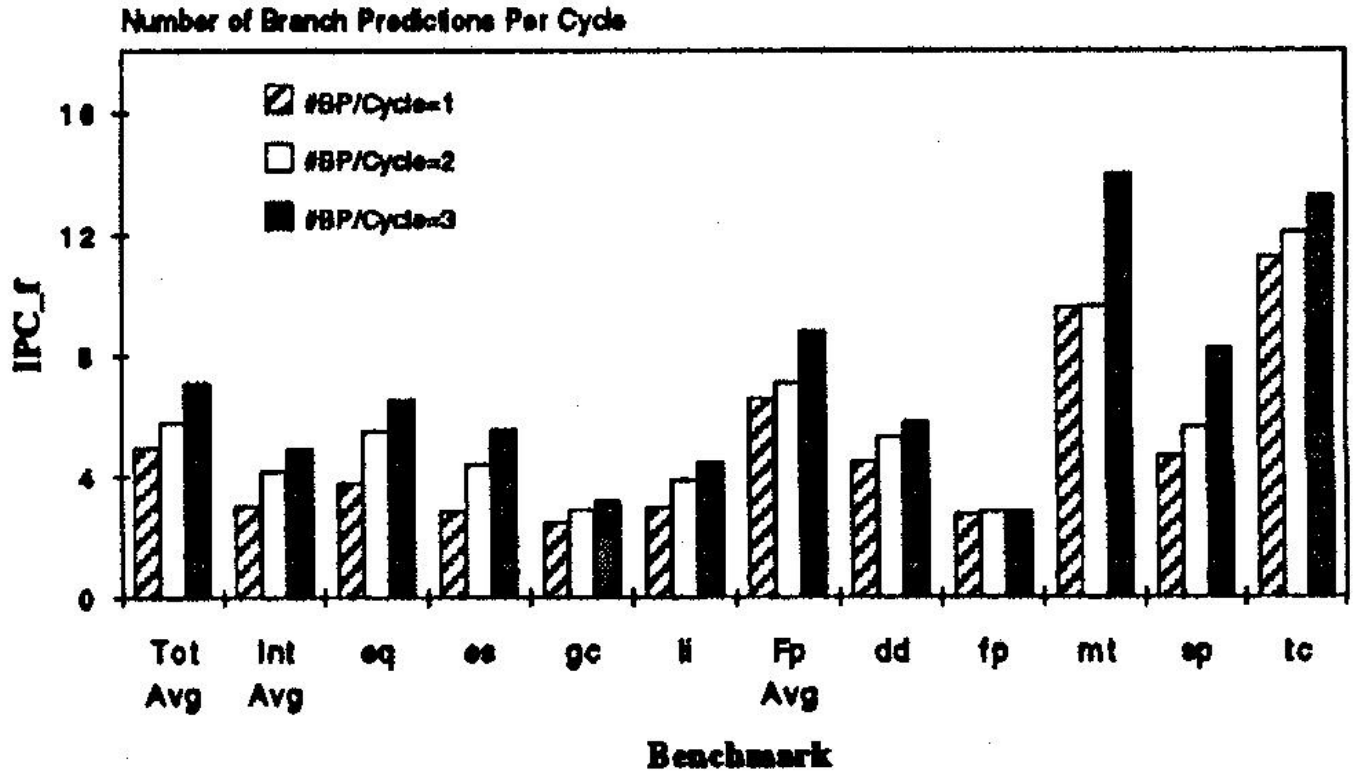

Figure 7: Instructions per cycle when 1, 2, and 3 branches are predicted each cycle.

Issues

- ***Issues of branch address cache***
  - I cache to support simultaneous access to multiple non-contiguous cache lines
    - Too expensive (multi-ported caches)
    - Bank conflicts (interleaved organization)
  - Complex shift and alignment logic to assemble non-contiguous blocks into sequential instruction stream
  - The number of target addresses stored in branch address cache increases substantially as you increase the branch prediction throughput

Trace Cache Rotenberg & Smith

- ***Idea***
  - Caching of dynamic instruction stream (Icache stores static instruction stream)
  - Based on the following two characteristics
    - Temporal locality of instruction stream
    - Branch behavior
      - Most branches tend to be biased towards one direction or another
- ***Issues***
  - Redundant instruction storage
    - Same instructions both in Icache and trace cache
    - Same instructions among trace cache lines

Trace Cache Rotenberg & Smith

- ***Organization***
  - A special top-level instruction cache each line of which stores a *trace*, a dynamic instruction stream sequence
  - Trace
    - A sequence of the dynamic instruction stream
    - At most n instructions and m basic blocks
      - n is the trace cache line size
      - m is the branch predictor throughput
      - Specified by a starting address and m - 1 branch outcomes
  - Trace cache hit
    - If a trace cache line has the same starting address and predicted branch outcomes as the current IP
  - Trace cache miss
    - Fetching proceeds normally from instruction cache

Trace Cache Organization

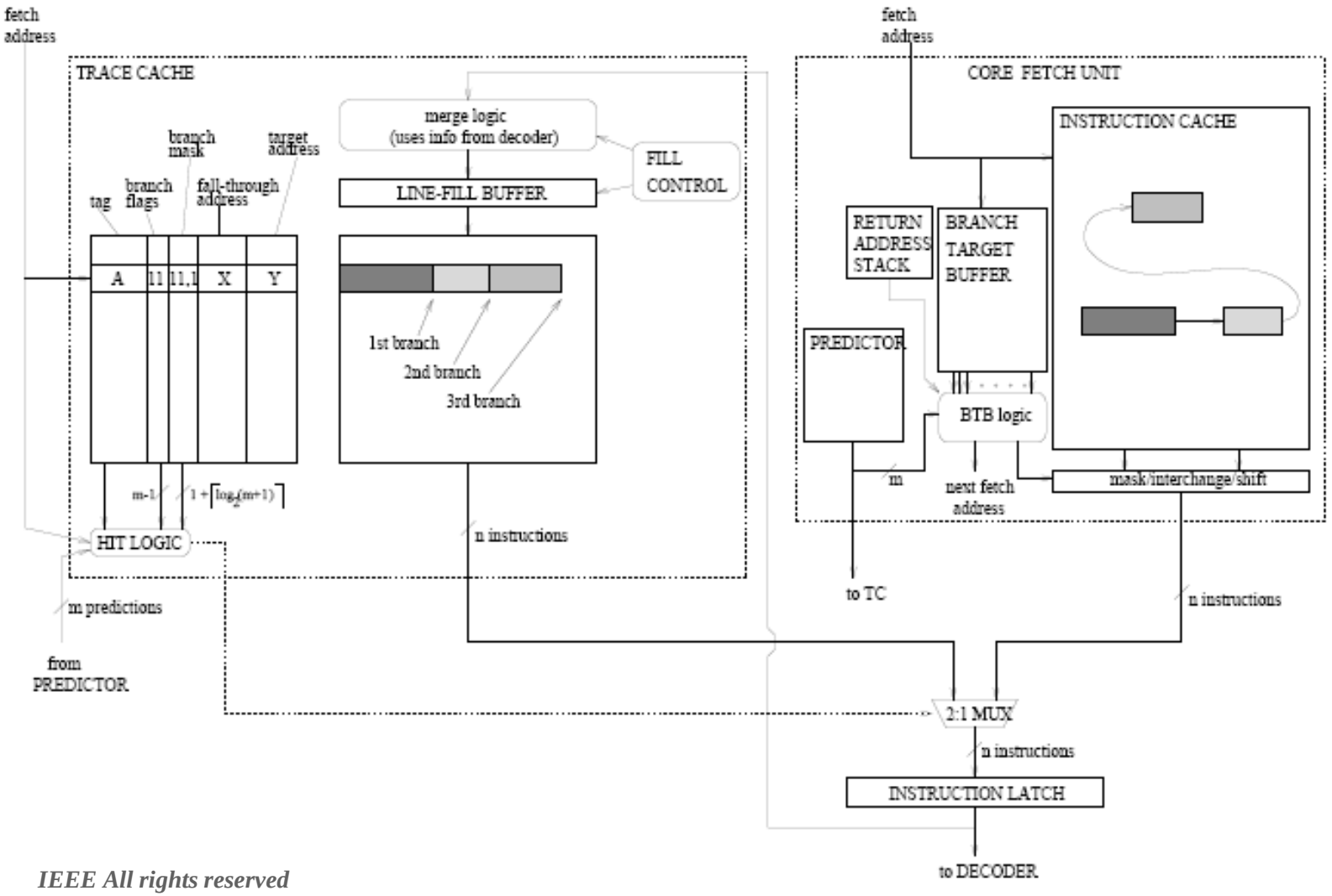

Figure 5: The trace cache fetch mechanism.

Design Options

- Associativity
- Path associativity
  - The number of traces that start at the same address
- Partial matches
  - When only the first few branch predictions match the branch flags, provide a prefix of trace
- Indexing
  - Fetch address vs. fetch address + predictions
- Multiple fill buffers
- Victim trace cache

Experimentation

- ***Assumption***
  - Unlimited hardware resources
  - Constrained by true data dependences
  - Unlimited register renaming
  - Full dynamic execution
- ***Schemes***
  - SEQ1: 1 basic block at a time
  - SEQ3: 3 consecutive basic blocks at a time
  - TC: Trace cache
  - CB: Collapsing buffer (Conte)
  - BAC: Branch address cache (Yeh)

Performance

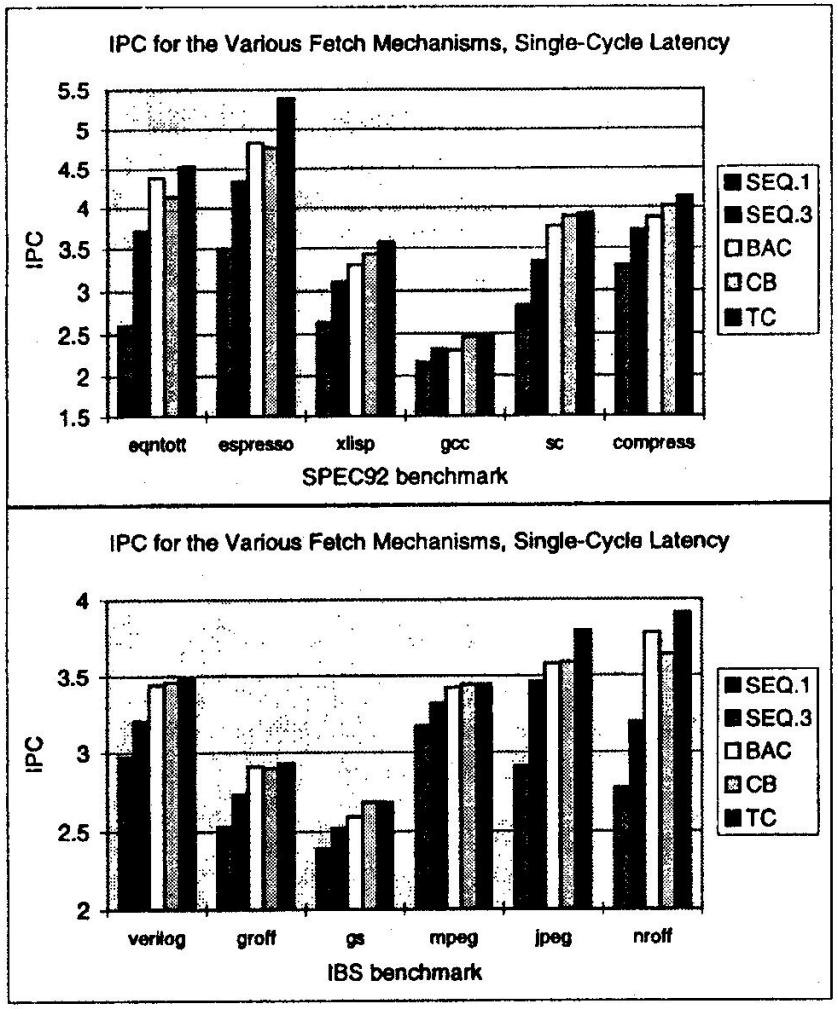

Figure 8. IPC results (fetch latency = 1 cycle).

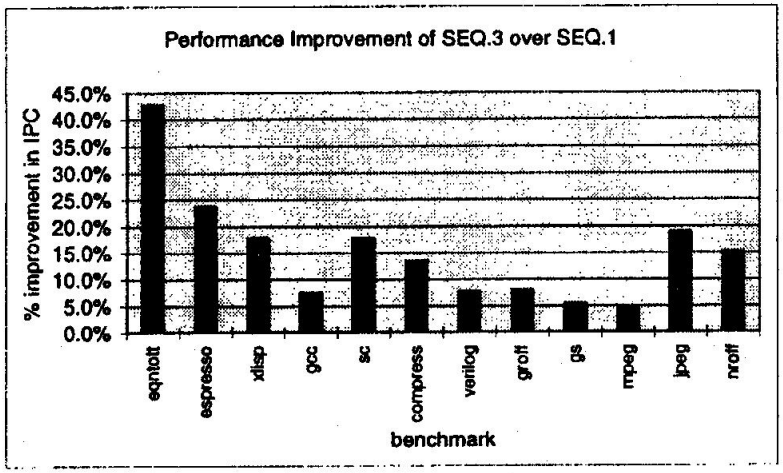

Figure 9. Improvement of SEQ.3 over SEQ.1.

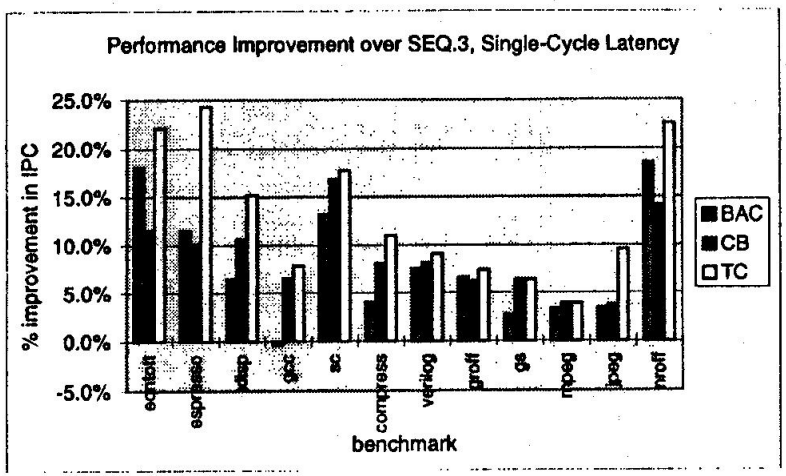

Figure 10. Improvement over SEQ.3.

Trace Cache Miss Rates

- Trace Miss Rate - % accesses missing TC
- Instruction miss rate - % instructions not supplied by TC

IBS	4 KB, 1-way		32 KB, 4-way		Spec	4 KB, 1-way	
	tmr	imr	tmr	imr		tmr	imr
veri	70%	48%	48%	25%	eqn	26%	8%
groff	76%	61%	60%	38%	esp	32%	14%
gs	76%	58%	60%	39%	xlisp	64%	40%
mpeg	70%	54%	51%	29%	gcc	71%	52%
jpeg	64%	43%	53%	25%	sc	50%	28%
nroff	62%	42%	45%	24%	comp	18%	6%

Table 3. Trace cache miss rates.

Exercises and Discussion

- ***Itanium uses instruction buffer between FE and BE? What is the advantages of using this structure?***
- ***How can you add path associativity to the normal trace cache?***