ELE 655
Microprocessor System Design

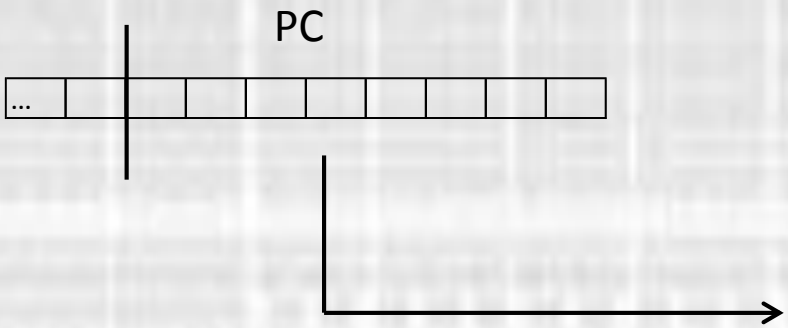
Section 2 – Instruction Level Parallelism

Class 4 – HW Based Speculation

ILP

Instruction Level Parallelism

- Dynamic Branch Prediction – 2 bit
  - Use 2 bits to make prediction decisions
    - Only change the prediction on 2 successive mispredictions
    - Resolves the 2 stall issue of 1 bit prediction



Branch Table

ADDR	B1	B0
0x00000000	0	0
0x00000001	0	0
0x00000010	1	0
...		
0x11111110	0	1
0x11111111	1	1

Option: Include bits in cache block instead of separate location – issue?

ILP
Instruction Level Parallelism

- Correlating Branch Predictor
  - Use information from recent (unrelated) branches in the current branch prediction

```
if  (aa==2)
            aa=0;
if  (bb==2)
            bb=0;
if  (aa! = bb)
            {…
```

  - Clearly the 3rd if is correlated to the first 2

ILP

Instruction Level Parallelism

- Correlating Branch Predictor
  - Use information from the previous *m* branches to choose 1 of 2^m solutions from an *n* bit branch predictor
  - Called a (m,n) correlated branch predictor
  - Ex. (4,2)

	Branch Prediction Table Index					Branch Prediction Table	
		m bit shift register					
Address	Address bottom	Prev #3	Prev #2	Prev #1	Prev #0	Pbit 1	Pbit 0
			…				
10010001	001	0	0	0	0	1	1
		0	0	0	1	0	0
		0	0	1	0	0	0
			…				
		1	1	1	1	1	0
	010	0	0	0	0	0	1
			…				
	111	1	1	1	0	0	0
		1	1	1	1	1	1
101010000	000	0	0	0	0	0	1
			…				
		1	1	1	1	1	0
			…				

ILP
Instruction Level Parallelism

- Correlating Branch Predictor
  - Consider a 4K entry branch table (maps 4K branch addresses) and a (0,2) correlated branch predictor
    - # bits required
      - m bits in the shift register
      - 2^m sets of branch predictors
      - n bits for each predictor

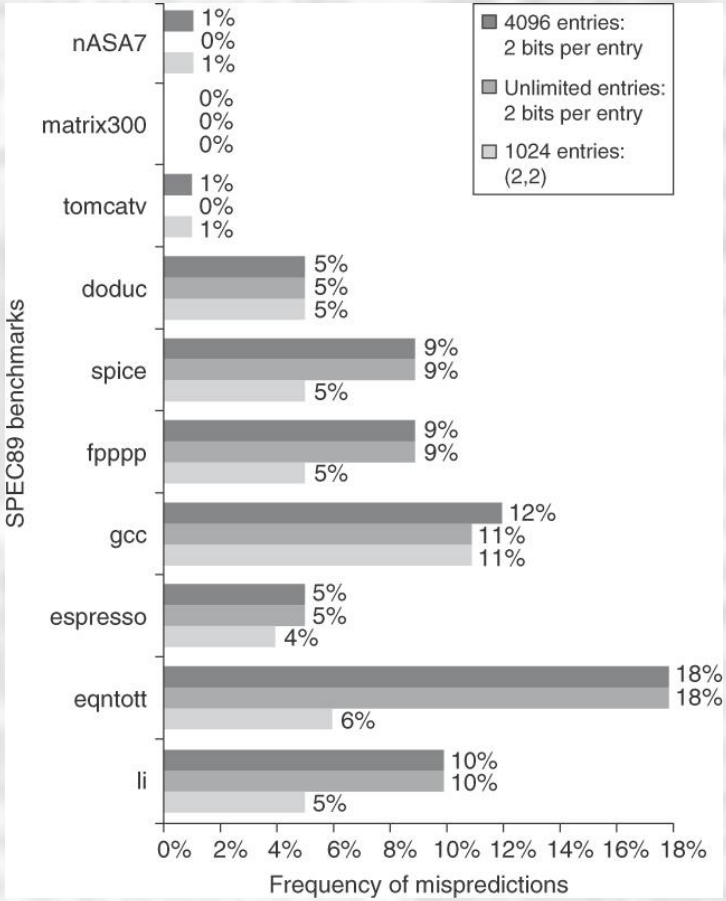
        $= 2^m \times n \times \#\text{ entries} = 2^0 \times 2 \times 4K = 8K + 0 = 8K$
  - Consider a (2,2) correlated branch predictor with the same # of bits
    - $2^2 \times 2 \times \#\text{ entries} = 8K$

      # entries = 1K entries

      4x fewer branch addresses or 4x as many aliases

ILP
Instruction Level Parallelism

- Correlating Branch Predictor
  - (2,2) provides better performance
    Even compared to an infinite uncorrelated predictor



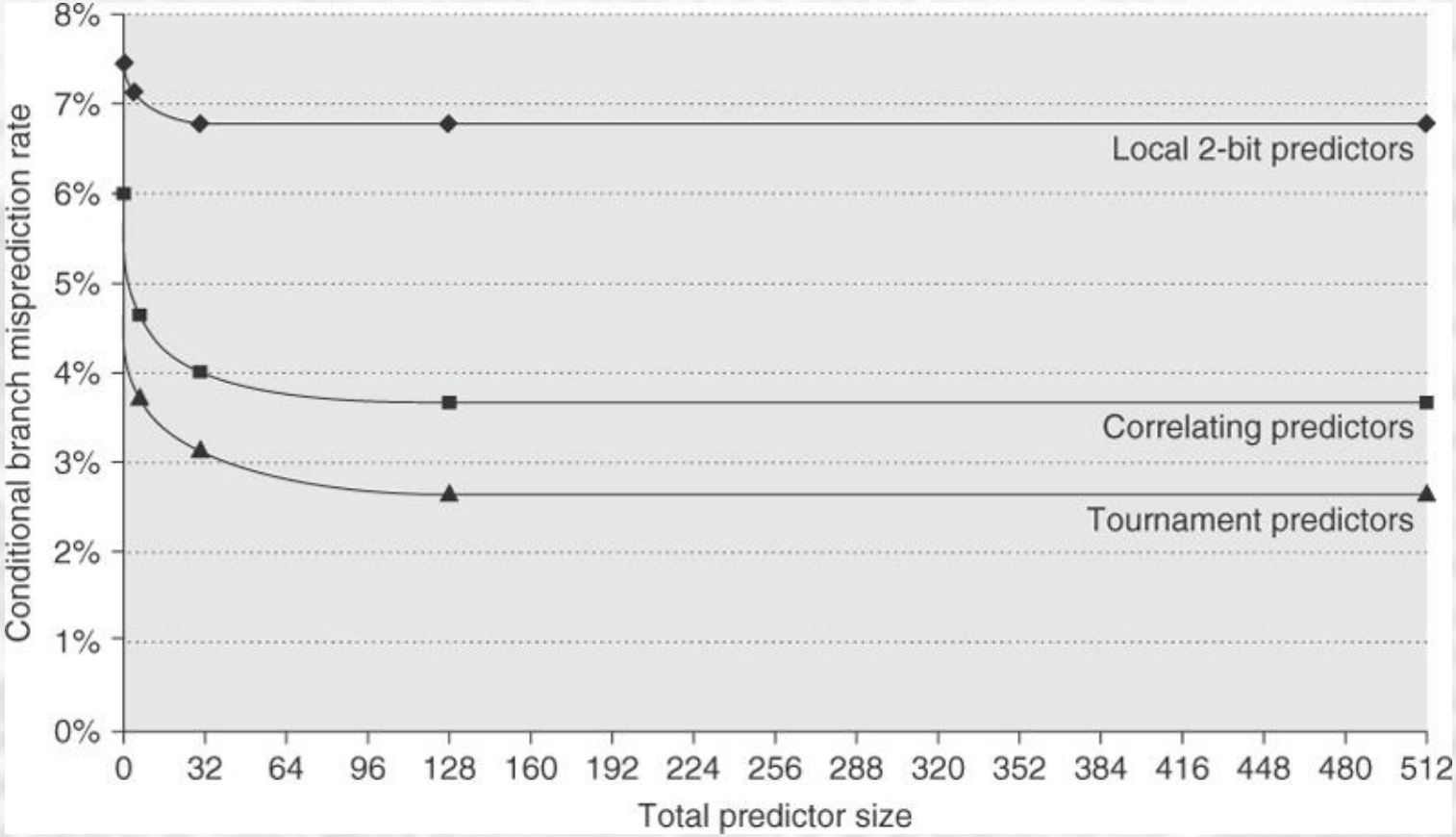
ILP
Instruction Level Parallelism

- Tournament Branch Predictor
  - Choose between multiple types of predictors based on the predictors individual records at predicting correctly
    - Select the global history predictor e.g (2,2)
      vs.
    - Select the local history predictor e.g 8 bit branch predictor
    - Need an additional predictor to model the choice e.g a 2 bit predictor

ILP

Instruction Level Parallelism

- Tournament Branch Predictor



ILP
Instruction level Parallelism

- Control Hazards - Revisited
  - Our primary control hazard is Branches
  - Branch Prediction only goes so far
    - Wide issue processors may have branches to execute every clock cycle

      BR
      Add
      Sub
      LD
      BR
      Add
      Add
      ST
      BR
      …
      → BR every clock

ILP
Instruction level Parallelism

- Control Hazards - Revisited
  - Prediction (with dynamic scheduling)
    - Predict branch direction
      - Fetch instruction
      - Issue instruction
      - Execution must wait on actual branch decision
  - Speculation
    - Predict branch direction
      - Fetch instruction
      - Issue instruction
      - Execute instruction

ILP

Instruction level Parallelism

- Hardware Speculation
  - Dynamic branch prediction
    - To maximize correct predictions
  - Dynamic scheduling
    - To allow out of order execution
  - Speculation
    - To allow execution before control dependencies are resolved
  - Correction
    - A method to undo effects of incorrect predictions

ILP

Instruction level Parallelism

- Hardware Speculation
  - Modify our dynamic scheduling algorithm
  - Separate
    - Forwarding of results (bypassing)
      from
    - Actual completion of the instruction
      - Delay any register or memory writes
      - Called – Commit
  - While instructions are speculative
    - Allow them to provide results to other instructions
  - Once the instructions are no longer speculative
    - Allow them to commit

ILP
Instruction level Parallelism

- Hardware Speculation
  - Modify our dynamic scheduling algorithm
    - Allow instructions to execute out of order
    - Force instructions to commit in order
      - Branches must be resolved before anything in the basic block is commited
    - Instructions (results) waiting to commit are held in the Reorder Buffer

ILP
Instruction level Parallelism

- Hardware Speculation
  - Reorder Buffer
    - Holds instruction results awaiting commit
    - These would have otherwise been written to the register file
      - Which would make the results available to future instructions

    → provide results to the reservation stations

ILP
Instruction level Parallelism

- Hardware Speculation
  - Reorder Buffer
    - Holds results until ready to commit
    - Very similar to the store buffer
      - Holds results until memory is ready
    - Combine the Reorder Buffer and the Store Buffer

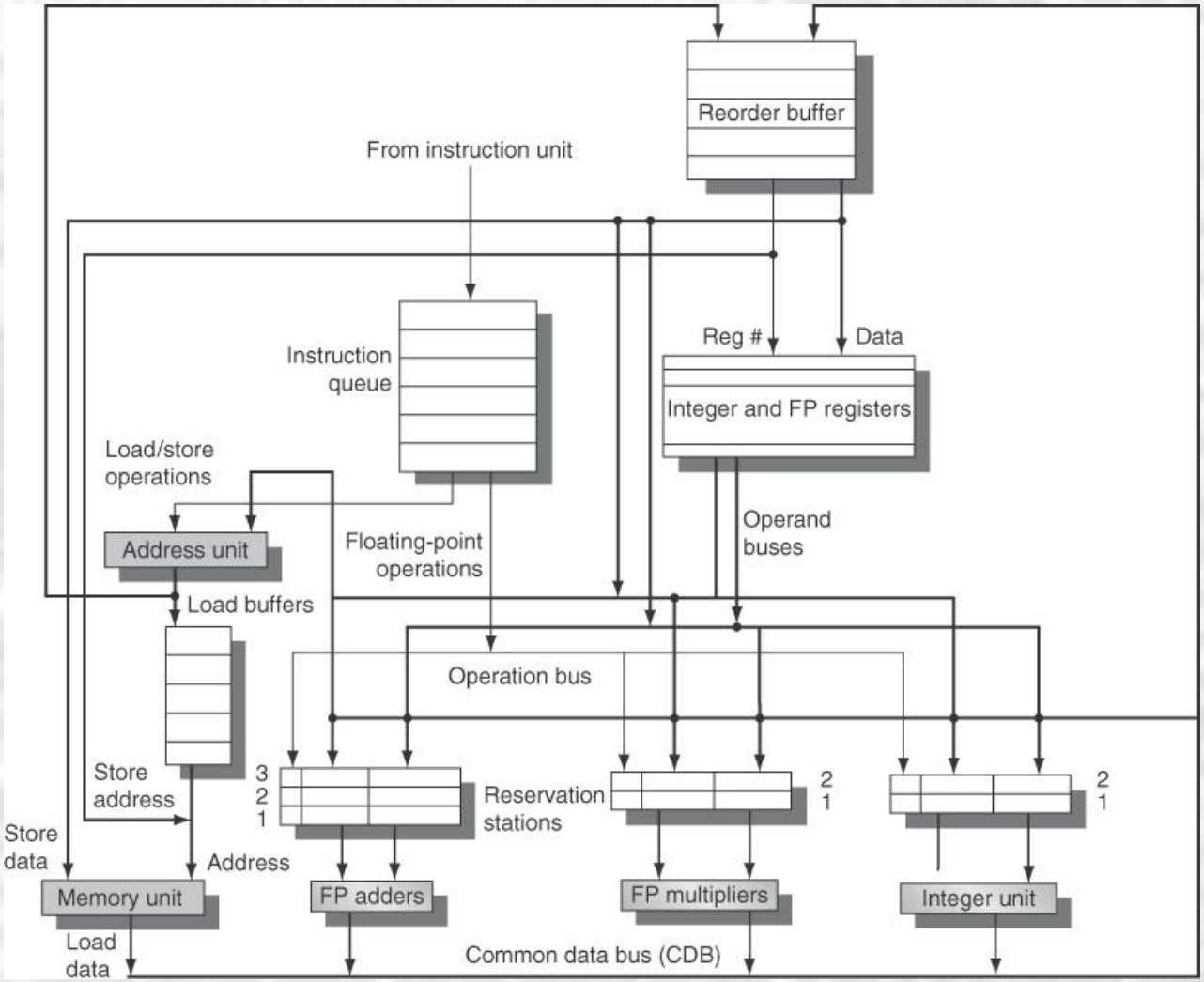
ILP
Instruction level Parallelism

- Hardware Speculation
  - Reorder Buffer - fields
    - Instruction Type
      - Branch
      - Store
      - Register operation (includes loads)
    - Destination
      - Memory address for stores
      - Register (includes loads)
    - Value
    - Ready
      - Instruction complete and value is ready for use or commit

ILP

Instruction level Parallelism

- Hardware Speculation



ILP
Instruction level Parallelism

- Hardware Speculation
  - Register renaming
    - Moves to the Reorder Buffer
      - Every instruction exists in the ROB
    - Reservation stations still needed to buffer operands
      - Reference the reorder buffer instead of the execute unit for missing operands
      - Add a field to indicate the destination in the ROB for the instruction

ILP
Instruction level Parallelism

- Hardware Speculation

  - Issue

    Fetch instruction
    IF – room in ROB and RS
        send operands to the RS if available
        send ROB id to RS for any missing operands
        send ROB id for result to RS
    Else - stall

  - Execute

    IF – missing operand(s)
        Monitor CDB for operands (tagged by ROB entry)
    Else - execute

ILP
Instruction level Parallelism

- Hardware Speculation
  - Write Result
    - When result is available
    - Provide result to CDB along with ROB tag
      - any RS waiting will take the result from the CDB
      - the tagged ROB will take the result from the CDB
    - Mark the RS as free

Note: stores require both the address and the value to eventually end up in the ROB

ILP
Instruction level Parallelism

- Hardware Speculation
  - Commit
    - 3 cases
      - Register instruction (excludes Stores and Branches)

        IF Instruction is at the head of the ROB
        Rob indicates the result is ready for commit
        Then Update the register (register file)
        Mark the ROB entry as available
      - Store

        IF Instruction is at the head of the ROB
        Rob indicates the result is ready for commit
        Then Update the memory
        Mark the ROB entry as available

ILP
Instruction level Parallelism

- Hardware Speculation
  - Commit
    - 3 cases
    - Branch

      IF Prediction was correct
      Then Mark the ROB entry as available

      IF Prediction was wrong
      Then Flush the ROB
      - all speculatively executed instructions are lost
      - no registers or memory has been updated

      Restart at correct branch location

Note – if early branch decisions are possible – only flush the parts of the ROB after the branch

ILP

Instruction level Parallelism

- Hardware Speculation
  - Fig 3.12 vs Fig 3.8
  - Loop example - Fig 3.13
    note RS is empty

ILP
Instruction level Parallelism

- Hardware Speculation
  - Precise Exceptions
    - Add an exception field to the ROB
    - When the instruction reaches the head of the ROB – then execute the exception
    - If the exception was in a mis-predicted branch that executed it will be purged before reaching commit