Branch Prediction

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen
Updated by Mikko Lipasti

Lecture Overview

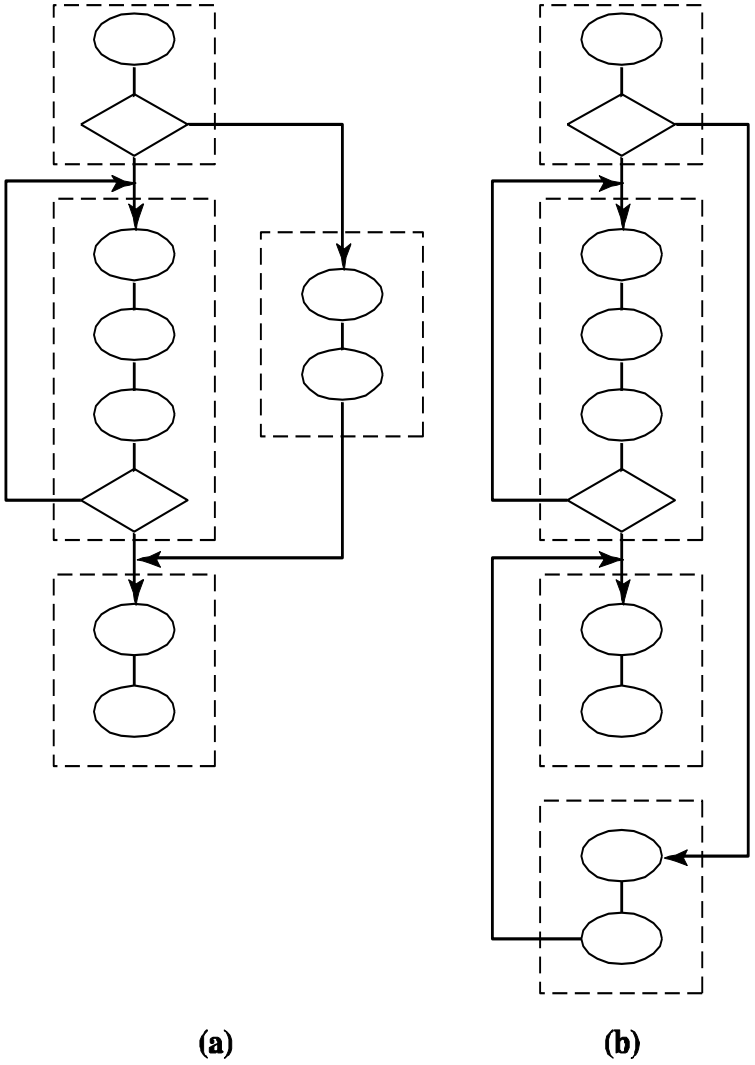
- Program control flow
  - Implicit sequential control flow
  - Disruptions of sequential control flow
- Branch Prediction
  - Branch instruction processing
  - Branch instruction speculation
- Key historical studies on branch prediction
  - UCB Study [Lee and Smith, 1984]
  - IBM Study [Nair, 1992]
- Branch prediction implementation (PPC 604)
  - BTAC and BHT design
  - Fetch Address Generation

Program Control Flow

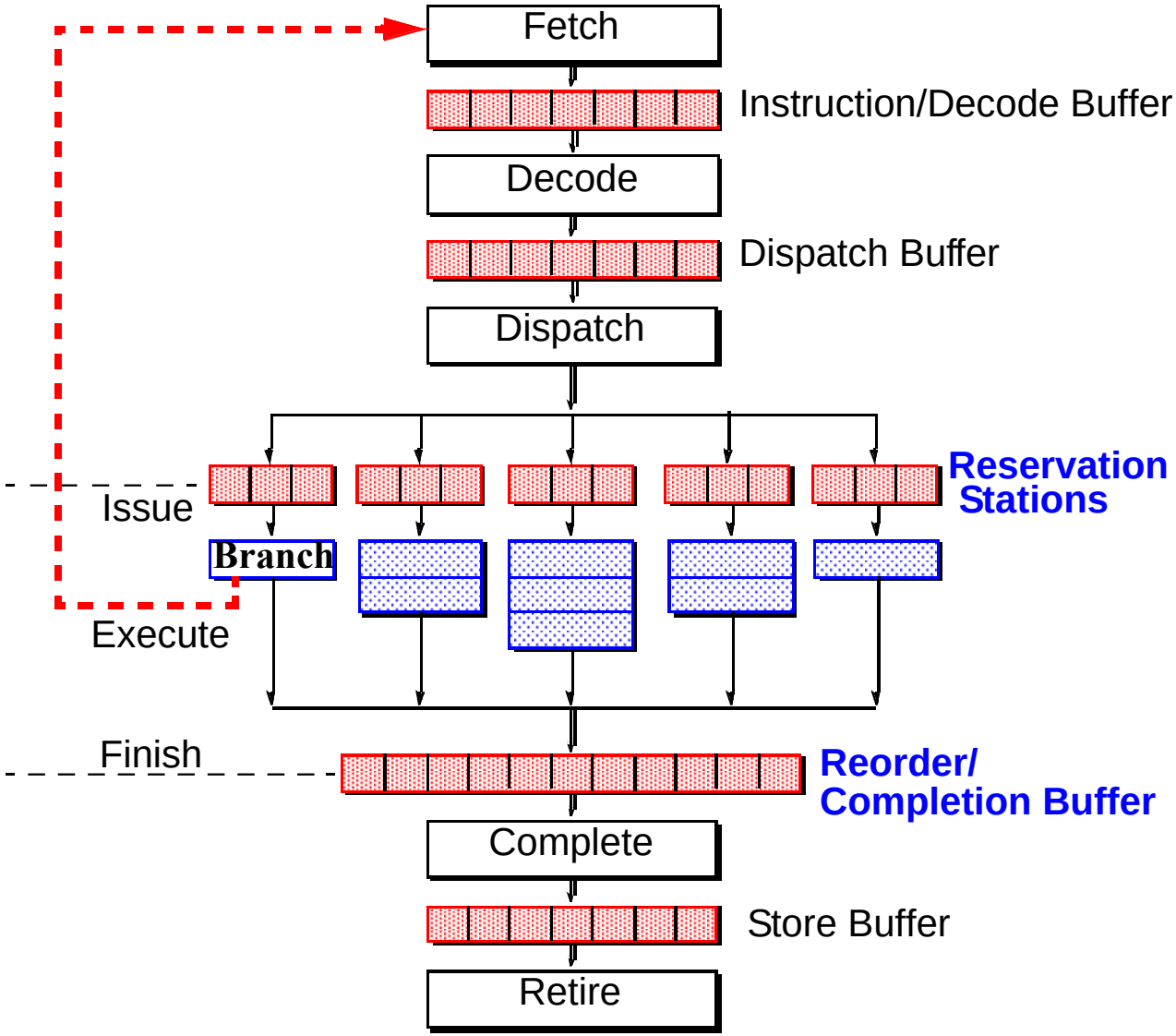
- Implicit Sequential Control Flow
  - Static Program Representation
    - Control Flow Graph (CFG)
    - Nodes = basic blocks
    - Edges = Control flow transfers
  - Physical Program Layout
    - Mapping of CFG to linear program memory
    - Implied sequential control flow
  - Dynamic Program Execution
    - Traversal of the CFG nodes and edges (e.g. loops)
    - Traversal dictated by branch conditions
  - Dynamic Control Flow
    - Deviates from sequential control flow
    - Disrupts sequential fetching
    - Can stall IF stage and reduce I-fetch bandwidth

Program Control Flow

- Dynamic traversal of static CFG
- Mapping CFG to linear memory



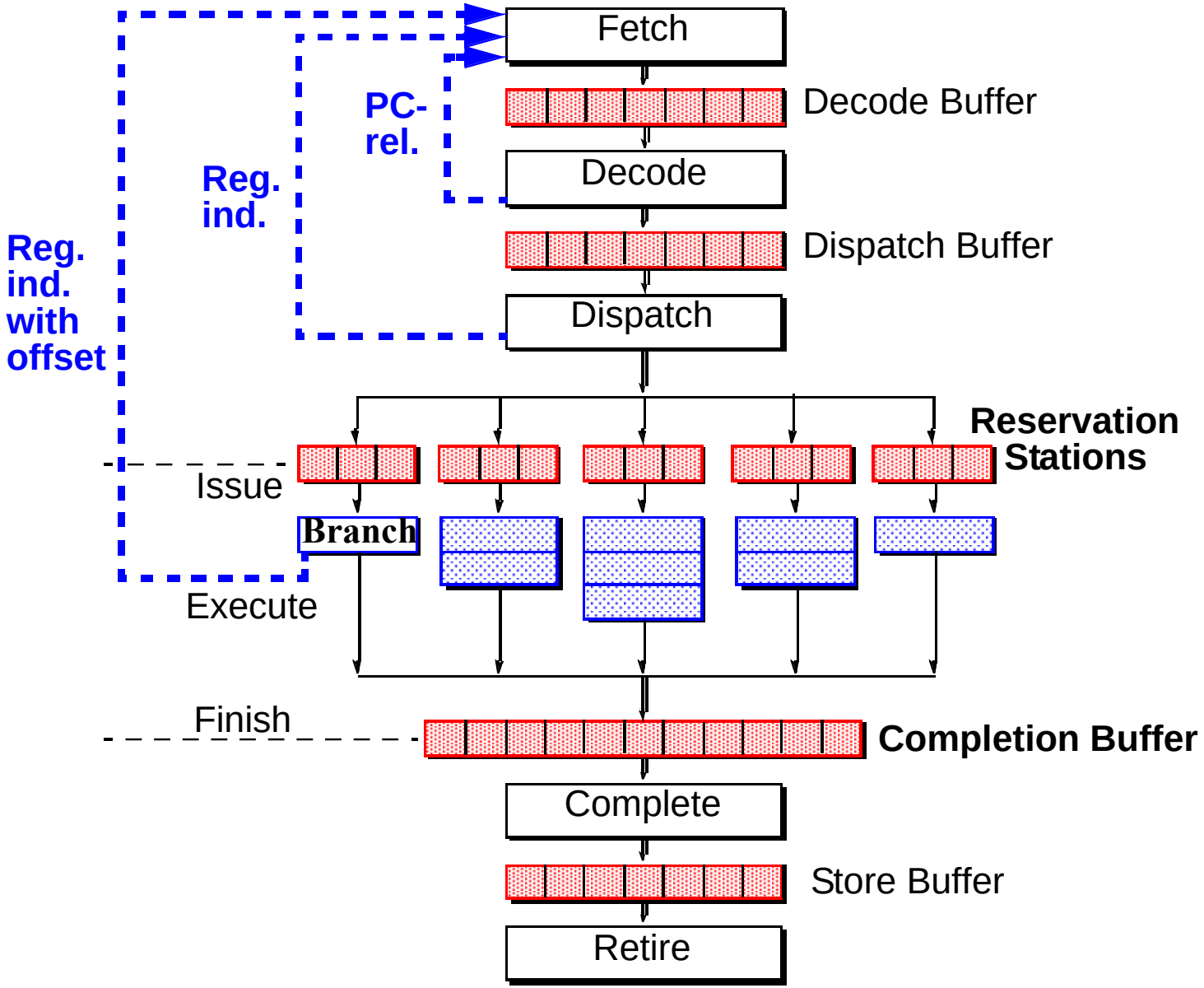
Disruption of Sequential Control Flow



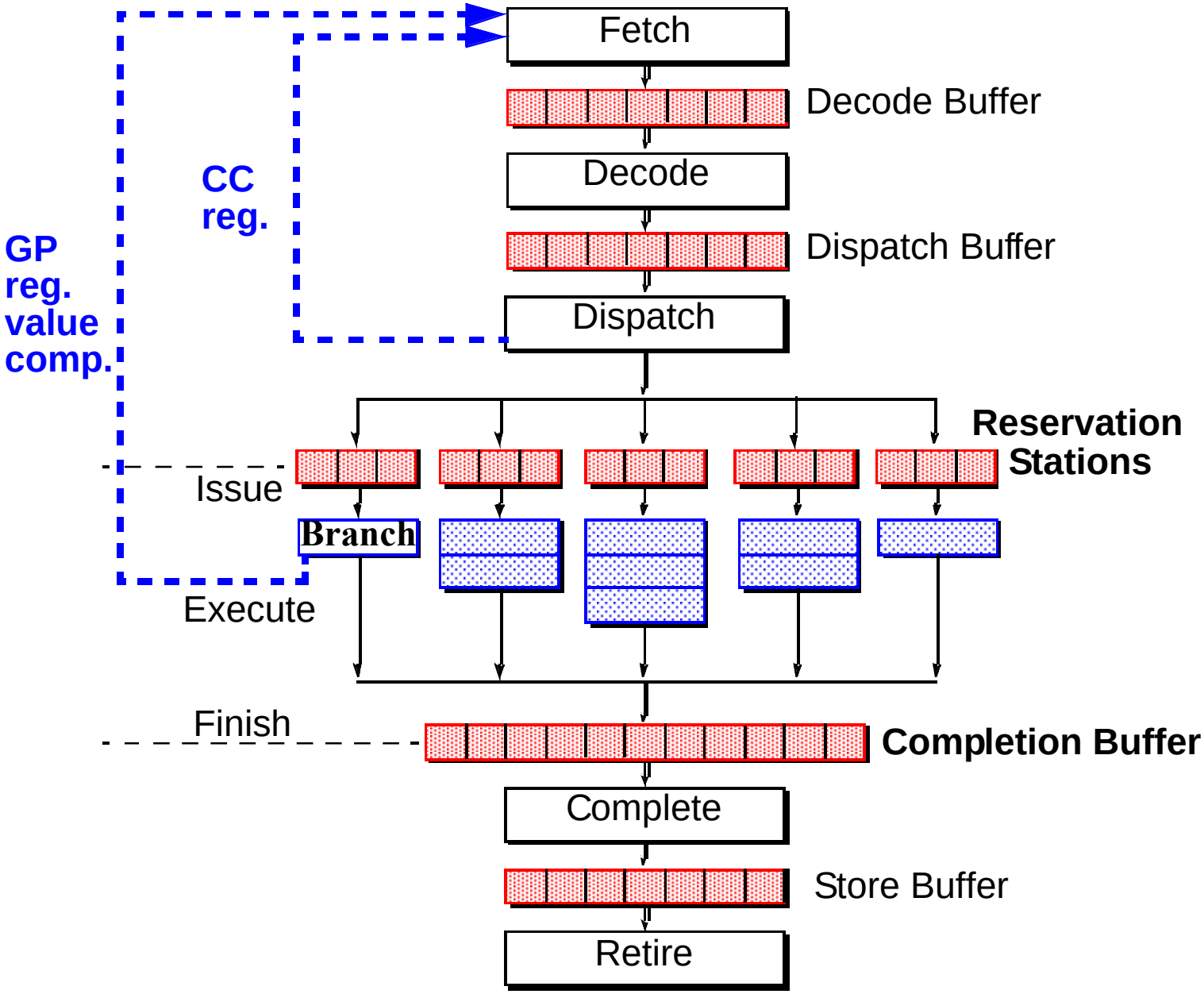
Branch Prediction

- Target address generation → Target Speculation
  - Access register:
    - PC, General purpose register, Link register
  - Perform calculation:
    - +/- offset, autoincrement, autodecrement
- Condition resolution → Condition speculation
  - Access register:
    - Condition code register, General purpose register
  - Perform calculation:
    - Comparison of data register(s)

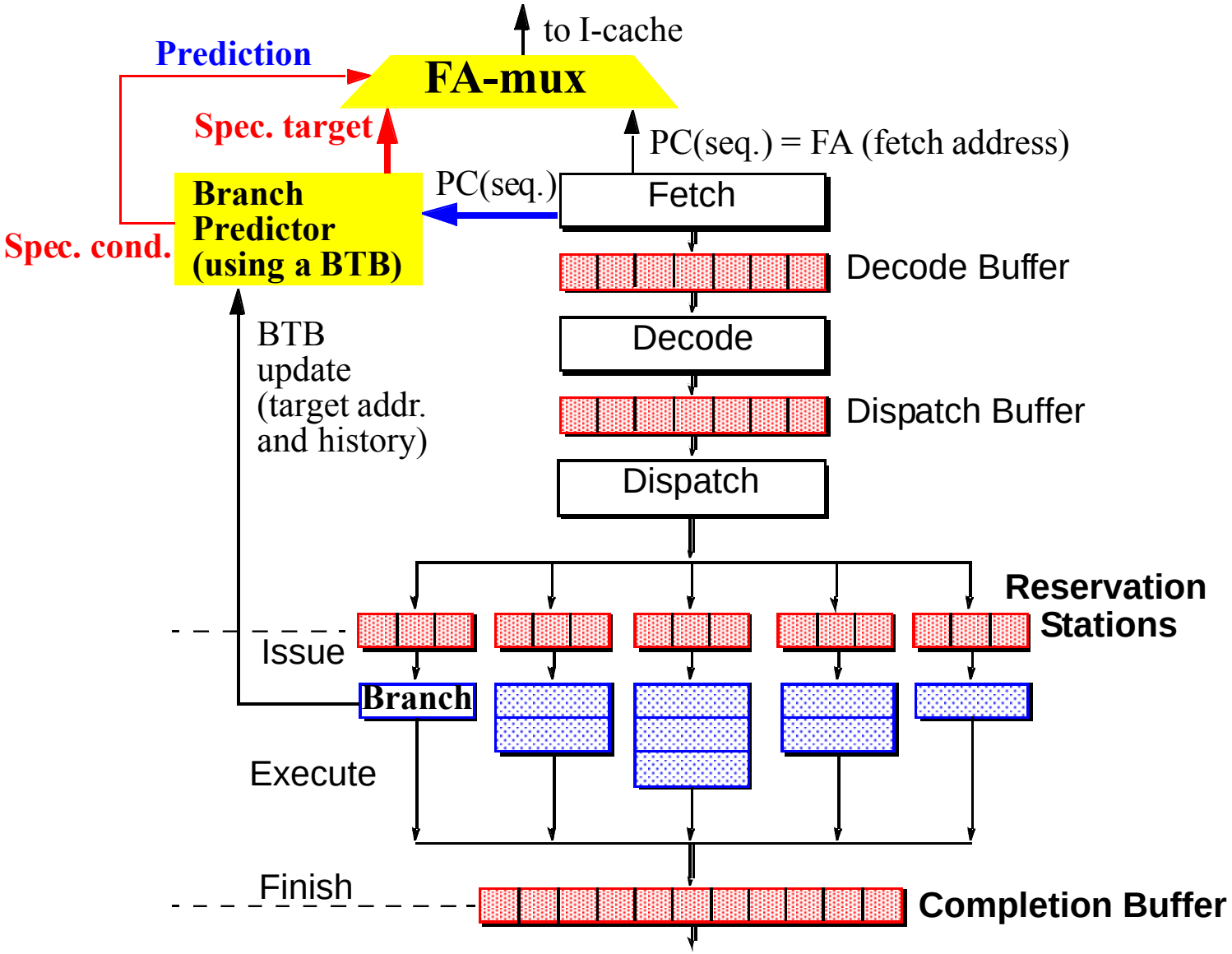
Target Address Generation



Condition Resolution



Branch Instruction Speculation



Branch/Jump Target Prediction

0x0348	0101 (NTNT)	0x0612

Branch inst. address | Information for predict. | **Branch target address** (most recent)

- Branch Target Buffer: small cache in fetch stage
  - Previously executed branches, address, taken history, target(s)
- Fetch stage compares current FA against BTB
  - If match, use prediction
  - If predict taken, use BTB target
- When branch executes, BTB is updated
- Optimization:
  - Size of BTB: increases hit rate
  - Prediction algorithm: increase accuracy of prediction

Branch Prediction: Condition Speculation

1. Biased Not Taken
   - Hardware prediction
   - Does not affect ISA
   - Not effective for loops
2. Software Prediction
   - Extra bit in each branch instruction
     - Set to 0 for not taken
     - Set to 1 for taken
   - Bit set by compiler or user; can use profiling
   - Static prediction, same behavior every time
3. Prediction based on branch offset
   - Positive offset: predict not taken
   - Negative offset: predict taken
4. Prediction based on dynamic history

UCB Study [Lee and Smith, 1984]

- Benchmarks used
  - 26 programs (IBM 370, DEC PDP-11, CDC 6400)
  - 6 workloads (4 IBM, 1 DEC, 1 CDC)
  - Used trace-driven simulation
- Branch types
  - Unconditional: always taken or always not taken
  - Subroutine call: always taken
  - Loop control: usually taken
  - Decision: either way, if-then-else
  - Computed goto: always taken, with changing target
  - Supervisor call: always taken
  - Execute: always taken (IBM 370)

IBM1: compiler
IBM2: cobol (business app)
IBM3: scientific
IBM4: supervisor (OS)

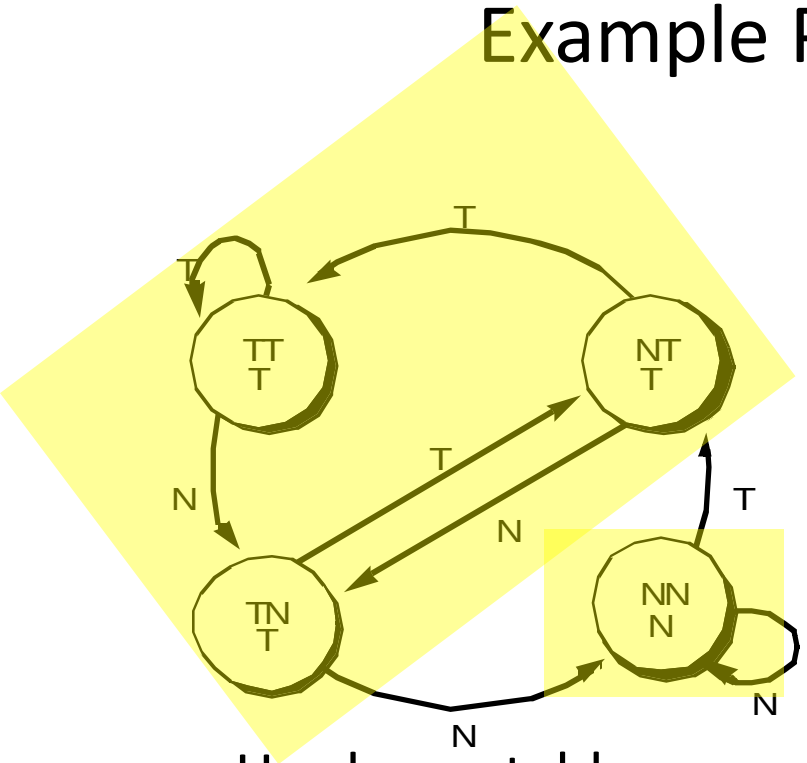
	IBM1	IBM2	IBM3	IBM4	DEC	CDC	Avg
T	0.640	0.657	0.704	0.540	0.738	0.778	0.676
NT	0.360	0.343	0.296	0.460	0.262	0.222	0.324

Branch Prediction Function

- Prediction function F(X1, X2, …)
  - X1 – opcode type
  - X2 – history
- Prediction effectiveness based on opcode only, or history

	IBM1	IBM2	IBM3	IBM4	DEC	CDC
Opcode only	66	69	71	55	80	78
History 0	64	64	70	54	74	78
History 1	92	95	87	80	97	82
History 2	93	97	91	83	98	91
History 3	94	97	91	84	98	94
History 4	95	97	92	84	98	95
History 5	95	97	92	84	98	96

Example Prediction Algorithm

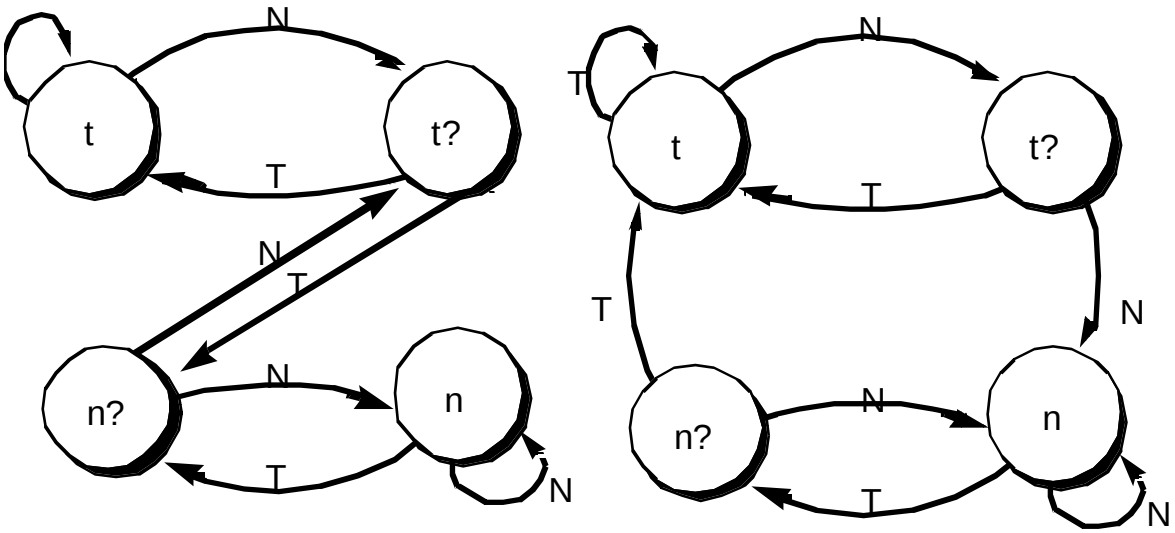


Branch inst. address	**Information for predict.**	Branch target address

- Hardware table remembers last 2 branch outcomes
  - History of past several branches encoded by FSM
  - Current state used to generate prediction
- Results:

Workload	IBM1	IBM2	IBM3	IBM4	DEC	CDC
Accuracy	93	97	91	83	98	91

Other Prediction Algorithms



- Combining prediction accuracy with BTB hit rate (86.5% for 128 sets of 4 entries each), branch prediction can provide the net prediction accuracy of approximately 80%. This implies a 5-20% performance enhancement.

IBM Study [Nair, 1992]

- Branch processing on the IBM RS/6000
  - Separate branch functional unit
  - Five different branch types
    - b: unconditional branch
    - bl: branch and link (subroutine calls)
    - bc: conditional branch
    - bcr: conditional branch using link register (returns)
    - bcc: conditional branch using count register
  - Overlap of branch instructions with other instructions
    - Zero cycle branches
  - Two causes for branch stalls
    - Unresolved conditions
    - Branches downstream too close to unresolved branches

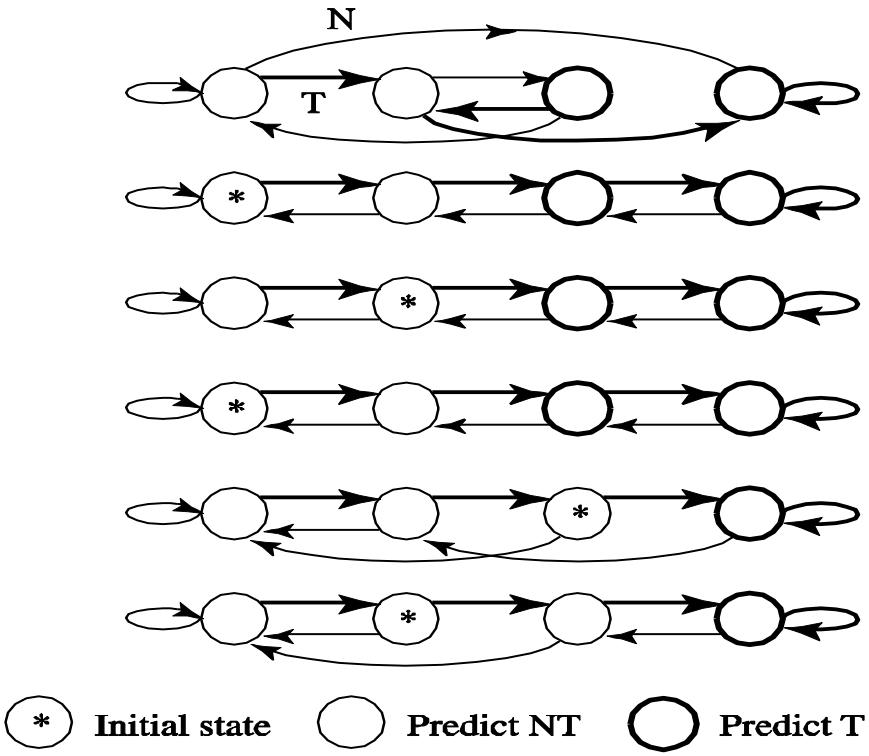
Branch Instruction Distribution

	% of each branch type				% bc with penalty cycles		
Benchmark	b	bl	bc	bcr bcc	3 cyc	2 cyc	1 cyc
spice2g6	7.86	0.30	12.58	0.32	13.82	3.12	0.76
doduc	1.00	0.94	8.22	1.01	10.14	1.76	2.02
matrix300	0.00	0.00	14.50	0.00	0.68	0.22	0.20
tomcatv	0.00	0.00	6.10	0.00	0.24	0.02	0.01
gcc	2.30	1.32	15.50	1.81	22.46	9.48	4.85
espresso	3.61	0.58	19.85	0.68	37.37	1.77	0.31
li	2.41	1.92	14.36	1.91	31.55	3.44	1.37
eqntott	0.91	0.47	32.87	0.51	5.01	11.01	0.80

Exhaustive Search for Optimal 2-bit Predictor

- There are 2^{20} possible state machines of 2-bit predictors
- Some machines are uninteresting, pruning them out reduces the number of state machines to 5248
- For each benchmark, determine prediction accuracy for all the predictor state machines
- Find optimal 2-bit predictor for each application

Benchmark	Optimal
spice2g6	97.2
doduc	94.3
gcc	89.1
espresso	89.1
li	87.1
eqntott	87.9

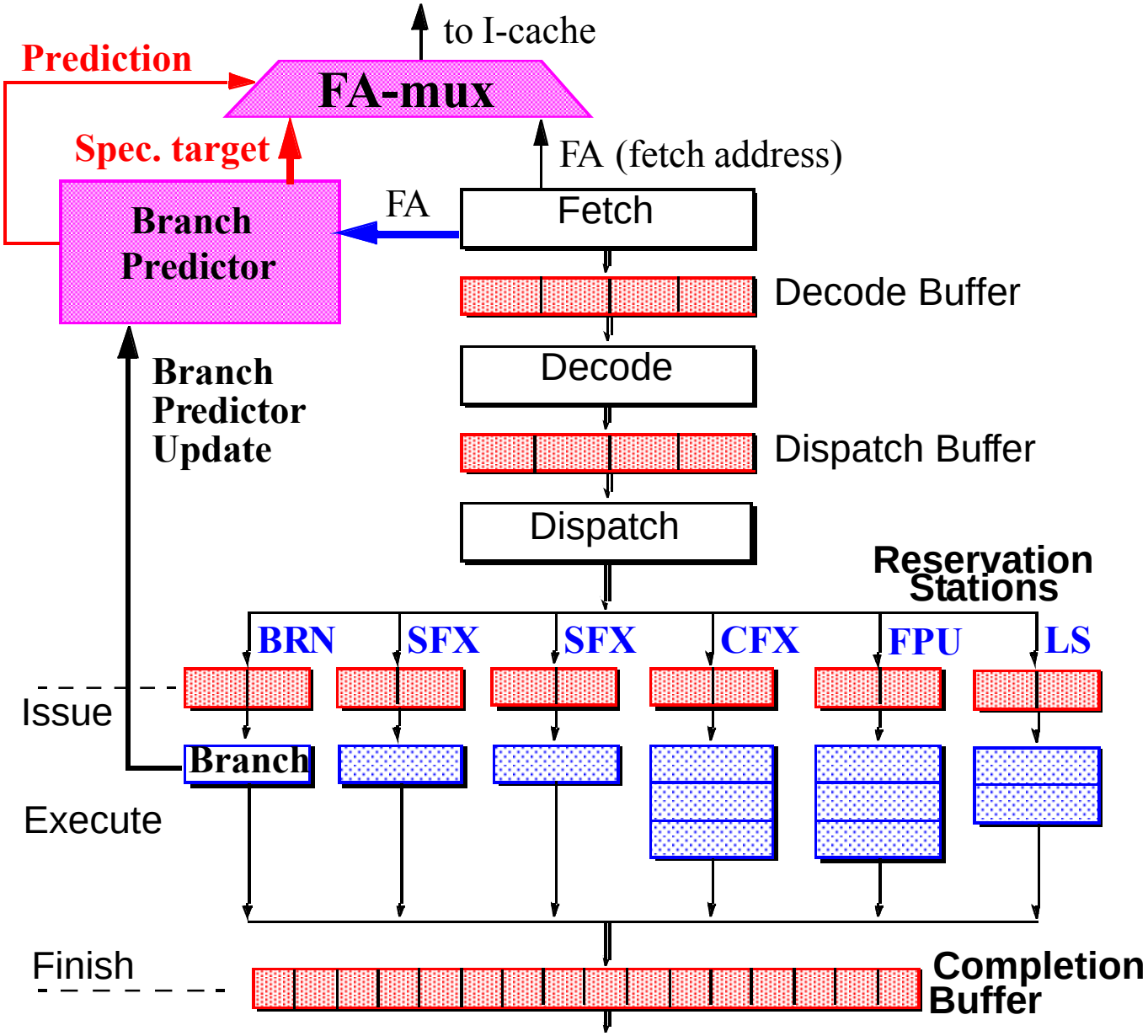


Number of History Bits Needed

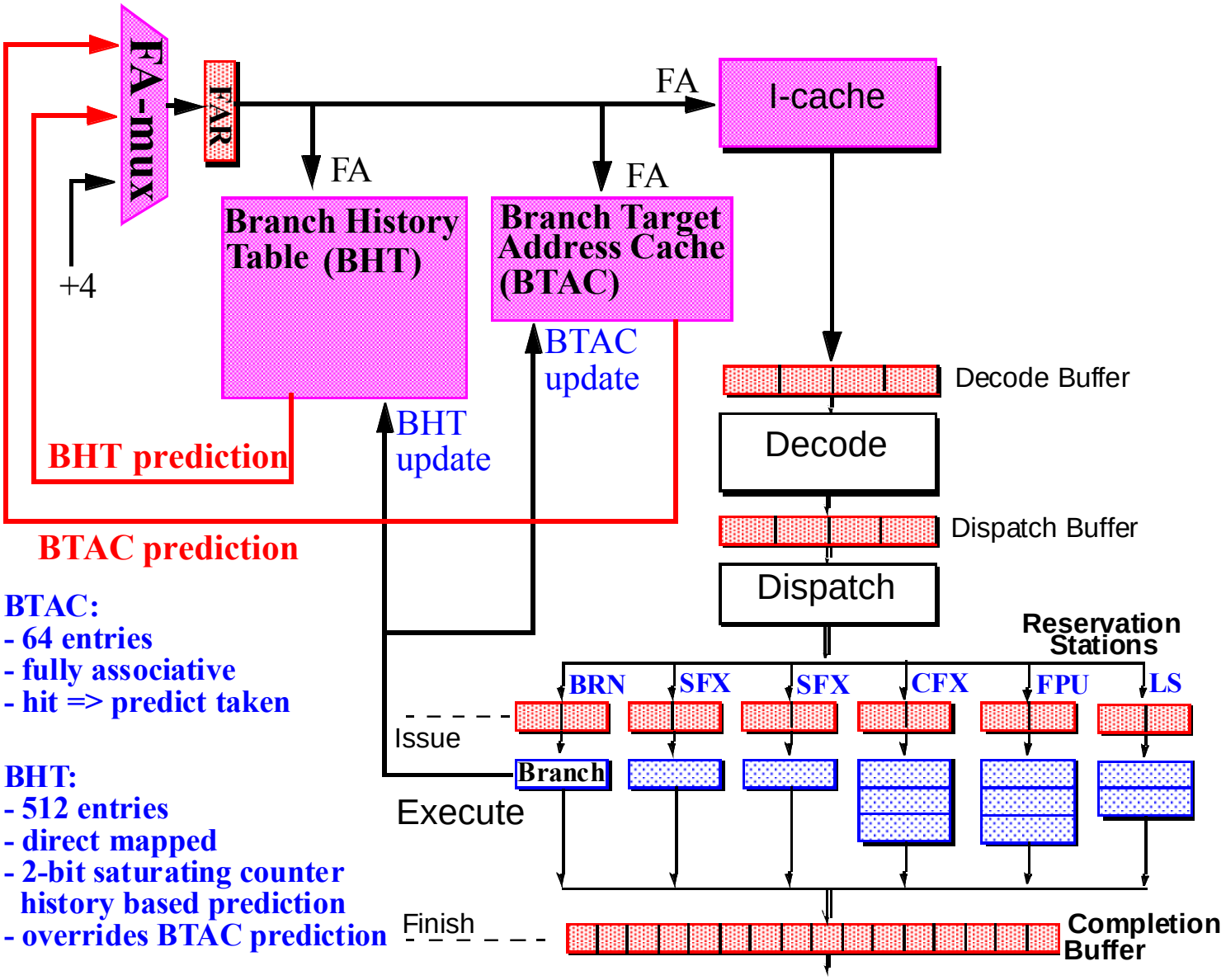
	Prediction Accuracy (Overall CPI Overhead)			
Benchmark	3 bit	2 bit	1 bit	0 bit
spice2g6	97.0 (0.009)	97.0 (0.009)	96.2 (0.013)	76.6 (0.031)
doduc	94.2 (0.003)	94.3 (0.003)	90.2 (0.004)	69.2 (0.022)
gcc	89.7 (0.025)	89.1 (0.026)	86.0 (0.033)	50.0 (0.128)
espresso	89.5 (0.045)	89.1 (0.047)	87.2 (0.054)	58.5 (0.176)
li	88.3 (0.042)	86.8 (0.048)	82.5 (0.063)	62.4 (0.142)
eqntott	89.3 (0.028)	87.2 (0.033)	82.9 (0.046)	78.4 (0.049)

- **Branch history table size**: Direct-mapped array of 2^k entries
- Some programs, like gcc, have over 7000 conditional branches
- In collisions, multiple branches share the same predictor
  - Constructive and destructive interference
  - Destructive interference
- Marginal gains beyond 1K entries (for these programs)

Branch Prediction Implementation (PPC 604)



BTAC and BHT Design (PPC 604)



BTAC:
- 64 entries
- fully associative
- hit => predict taken

BHT:
- 512 entries
- direct mapped
- 2-bit saturating counter history based prediction
- overrides BTAC prediction

BTAC and BHT Design (PPC 604)