Master of Computer Applications

CAPOL403R01: Computer Organization & Architecture

Unit II: Lecture 4 Part 2
Pipelining

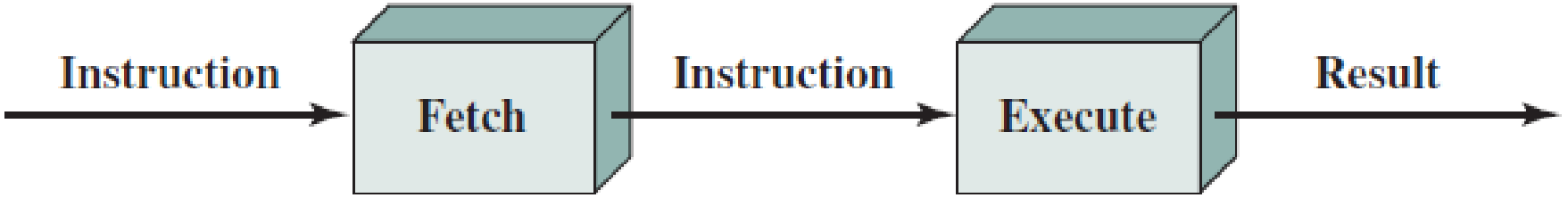
Dr. D. MURALIDHARAN
School of Computing
SASTRA Deemed to be University

Instruction pipelining

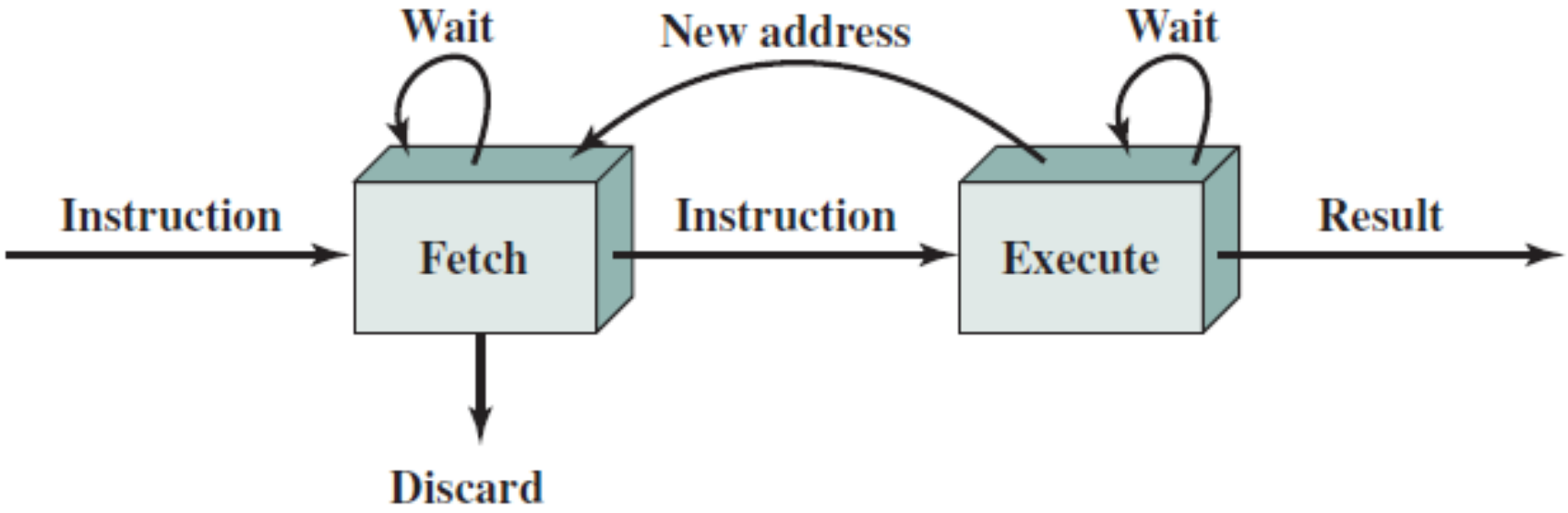
- It is a special processor organization
- It enhances the speed of the program execution
- Instruction execution has many phases such as fetch, decode…
- Instruction pipeline processes the phases of different instructions simultaneously
  - For example, it decodes the first instruction and fetching the second instruction at the same time
- The phases of the instructions are called as ‘stages’
- Pipeline has independent units to process the various stages of the instructions
- Pipeline requires registers to buffer the data between stages

Two stage pipelining – Block diagram

Simplified View



Detailed View



6-stage pipelining

- Consider the following 6 stages
  - Fetch instruction – FI
  - Decode instruction - DI
  - Calculate operands –CO
  - Fetch operands -FO
  - Execute instruction -EI
  - Write operand –WO

- With this decomposition, all stages have almost equal process timing

Speed enhancement of a 6-stage pipelining

- Consider a program with 9 instructions
- A single instruction execution is taking 6 time units
- Without pipelining, the program execution takes 9 x 6 = 54 time units
- Consider a 6-stage pipeline organization of a processor
- Assume all processing units takes same time
  - The processing time of a single stage is 1 time unit
- The same program execution is taking only 14 time units

Speed enhancement of a 6-stage pipelining

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Pipeline performance

$\tau = \max[\tau_i] + d = \tau_m + d \quad 1 \le i \le k$

In general, d is equivalent to 1 clock cycle; $d << \tau_m$; Hence, $\tau \approx \tau_m$

$T_{k,n} = [k+(n-1)]\ \tau$

Where $T_{k,n}$ be the total time to execute n instruction using a k-stage pipelined processor

Without pipelining organization, the processor takes $k\tau$ time to execute a single instruction

The total execution time of a program with 'n' instructions is $T_{1,n} = n.k.\tau$

Speed up $S = \frac{T_{1,n}}{T_{k,n}} = \frac{n.k.\tau}{[k+(n-1)]\ \tau} = \frac{n.k}{k+n-1}$

When n → ∞, n+(k-1) is n;

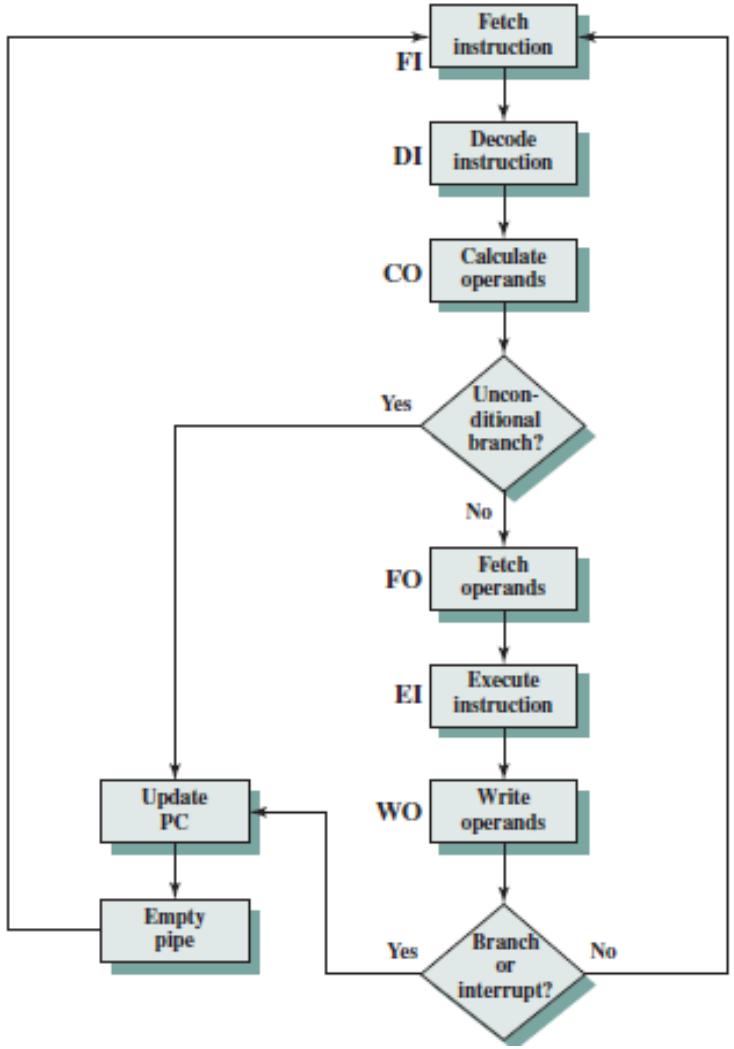
Speed up = k

The speed is increased 'k' times

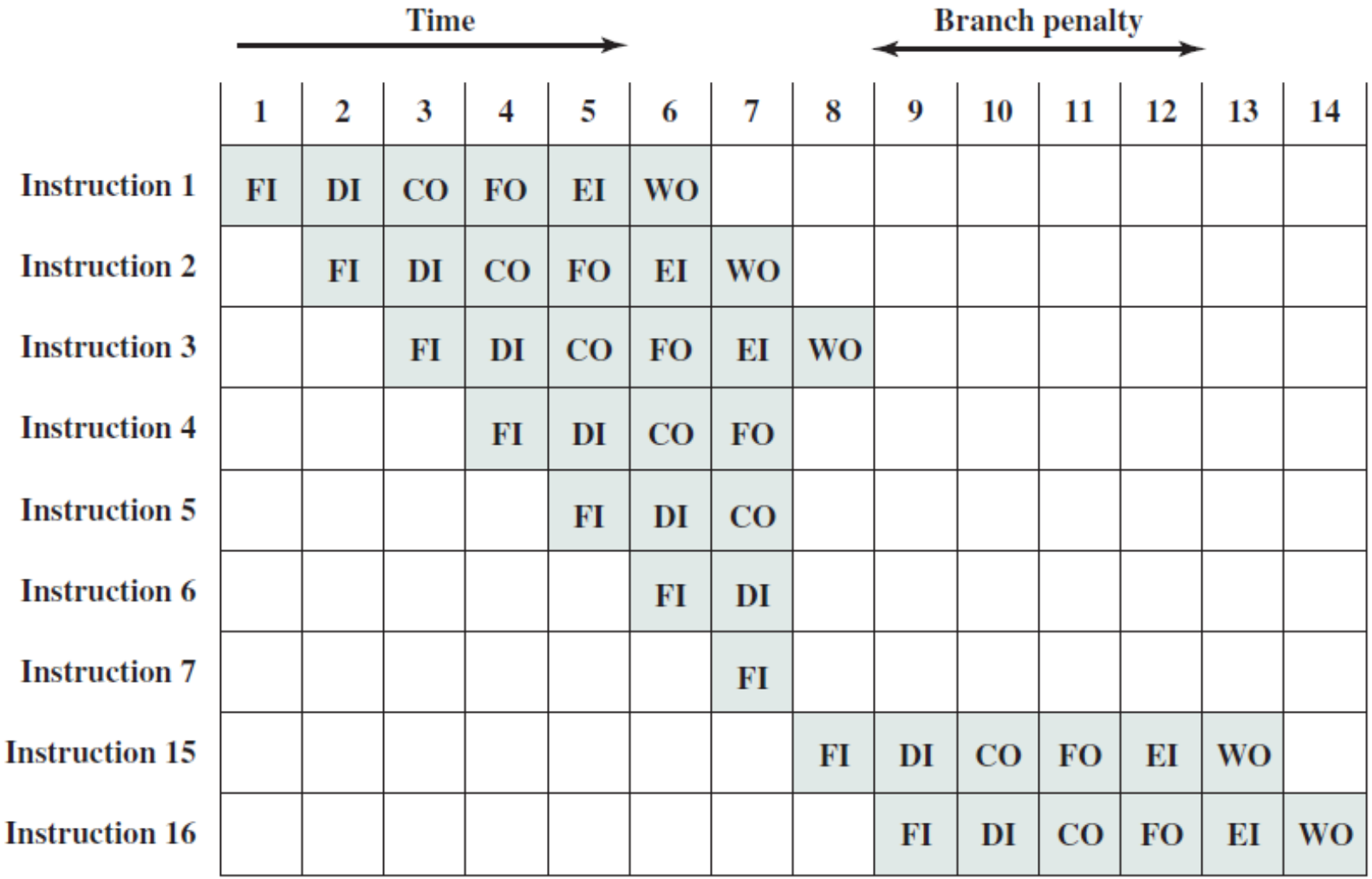
Factors affecting pipeline performance

- Memory conflict
  - When FI, FO and WO compete to access memory, only one unit wins. Other two units are in waiting condition
- Unequal processing time
  - All units have to wait until the unit with worst case processing time completes
- Conditional branch instruction
  - The branching address is not known until the execution is over
  - The previous units may have to flush out and the cycle may begin afresh
- Interrupt
  - Same as conditional branch
- Data dependency

Flow chart of a 6 stage pipeline



Effect of conditional branch



I3 is conditional branch instruction
When branch is not taken, I4 is the next instruction
When it is takes I15 is the next instruction

Pipeline stages – with and without branches

Without branch

Time

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

With conditional branch

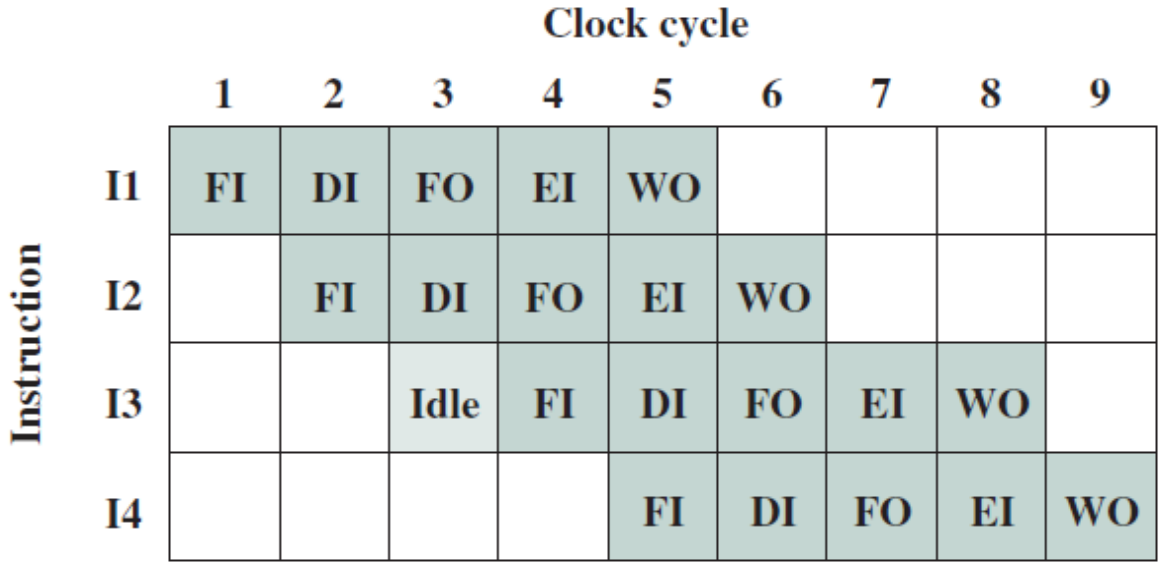
	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

Pipeline hazards

- It occurs when the pipeline (or some portion) must stall because conditions do not permit continued execution
- Types of hazards
  - Resource hazard / Structural hazard
  - Data hazards
  - Control hazards

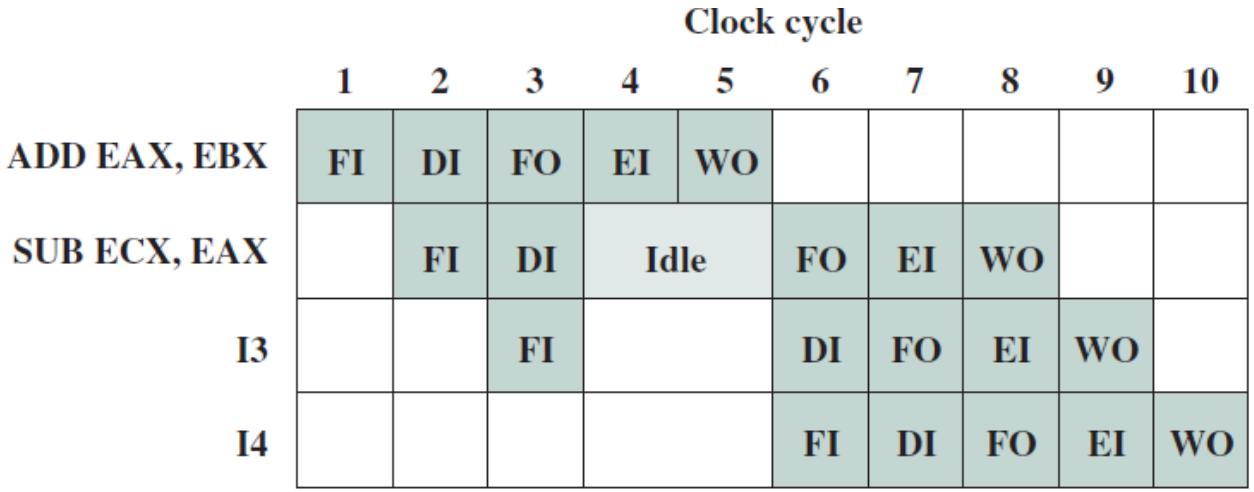
Resource hazards

- It occurs when more than one instruction in the pipeline compete for the same resource
- Assume the operands of first instructions are in memory.
- Consider all other operands are in registers
- Now there will be a conflict between I1 and I3



Data hazards

- It occurs when there is a conflict in the access of an operand location
  - Ex: The destination register of I_n is the source register of I_{n+1} .



- Types of data hazards
  - Read after write (RAW) – True dependency
  - Write after read (WAR) – Anti dependency
  - Write after write (WAW) – output dependency

Control hazard

- It is also known as branch hazard
- It occurs when the instructions in pipeline has to discarded due to the wrong branch target prediction
- Dealing with branches
  - Multiple stream
  - Prefetch branch target
  - Loop buffer
  - Branch prediction
  - Delayed branch

Branch handling methods

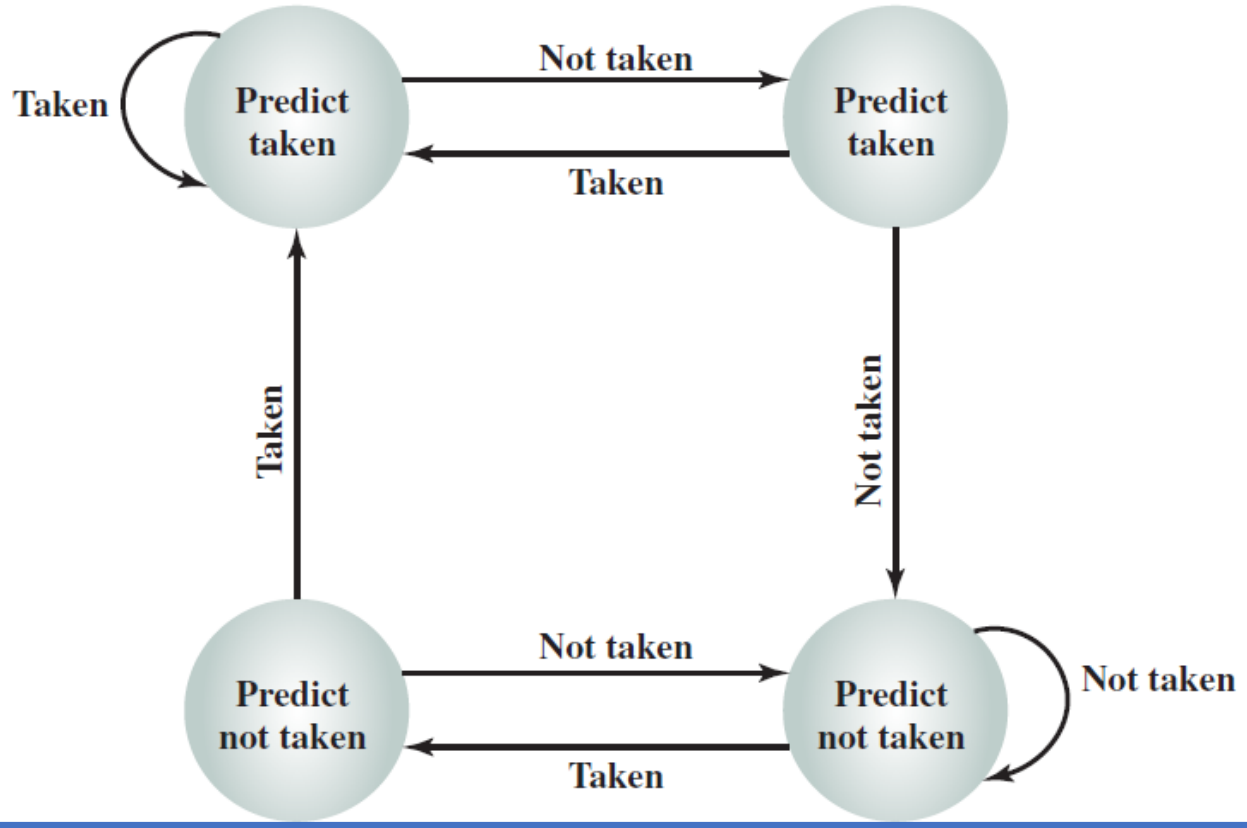
- Multiple streams
  - Duplicate the initial blocks of pipeline and allow two or more streams
- Prefetch branch target
  - The branch target is fetched and saved until the decision is taken
- Loop buffer
  - It is a small, very high speed memory
  - It is maintained by the instruction fetch stage
  - It has the n most recently fetched instructions in sequence
  - If branch has to be taken, the hardware first checks whether the branch target is within the buffer.
  - If it is available, then the next instruction(s) will be fetched from the buffer
  - This is very useful for iterations – hence the name for loop buffer

Branch handling methods

- Branch prediction – static methods
  - Predict never taken
  - Predict always taken
  - Predict by opcode

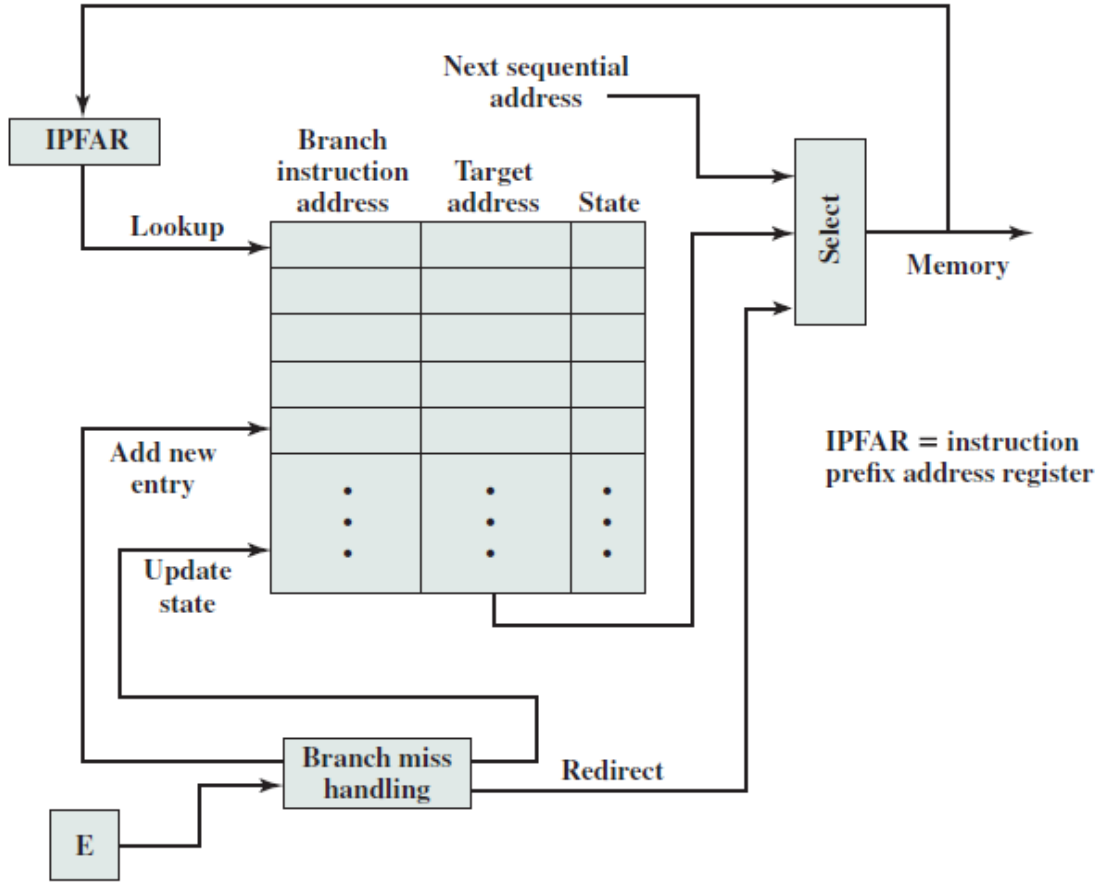
Branch handling methods

- Branch prediction – dynamic methods
  - Taken / not taken switch
  - Consider 2 additional bits are used with conditional branches



Branch handling methods

- Branch prediction – dynamic methods
  - Branch history table



Thank you