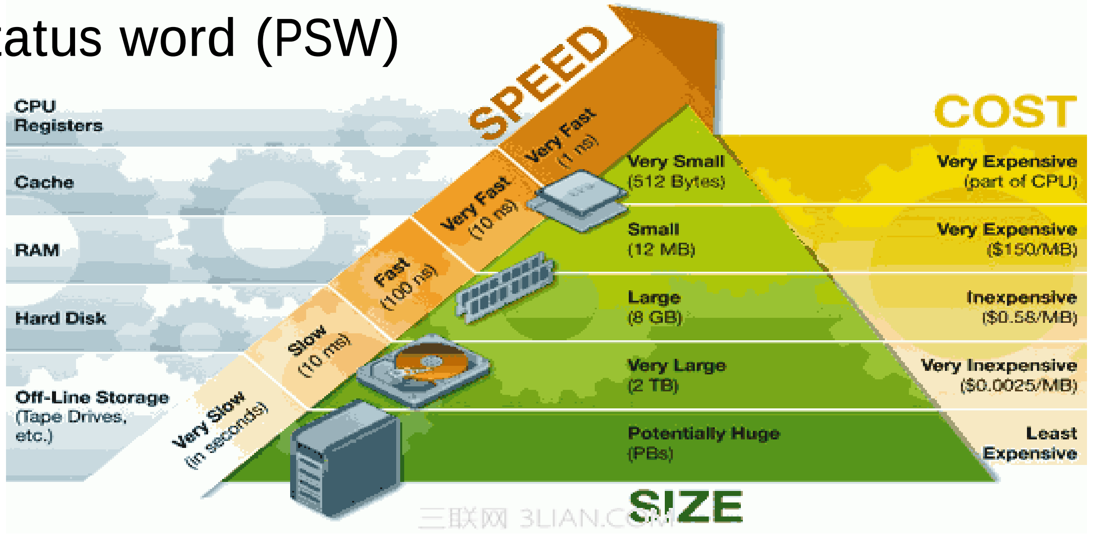
William Stallings
Computer Organization
and Architecture
7th Edition

Chapter 12

CPU Structure and Function

Key points

- User-visible registers and control/status registers
  - —User-visible registers
    - – General purpose
    - – Special use: e.g., fixed point num, floating point num, address, index, segment pointer
  - —Control/status registers, e.g.,
    - – Program counter (PC)
    - – Program status word (PSW)

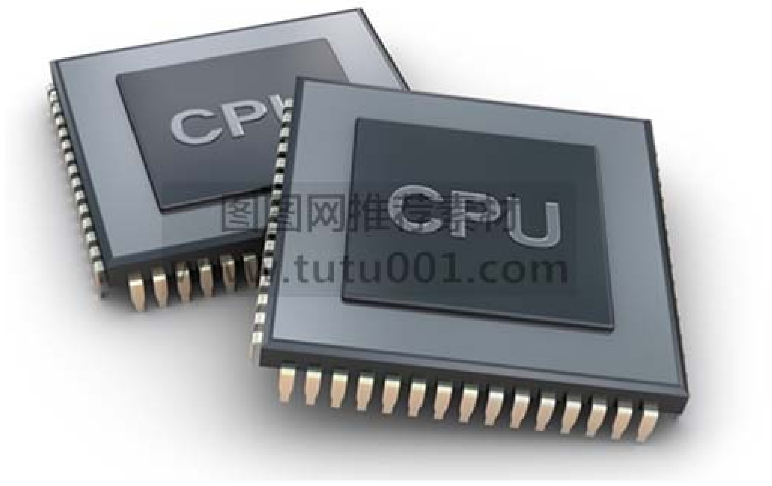


Key points

- Pipeline
  - —Each stage works on different instruction at the same time
  - —Branches complicate the design

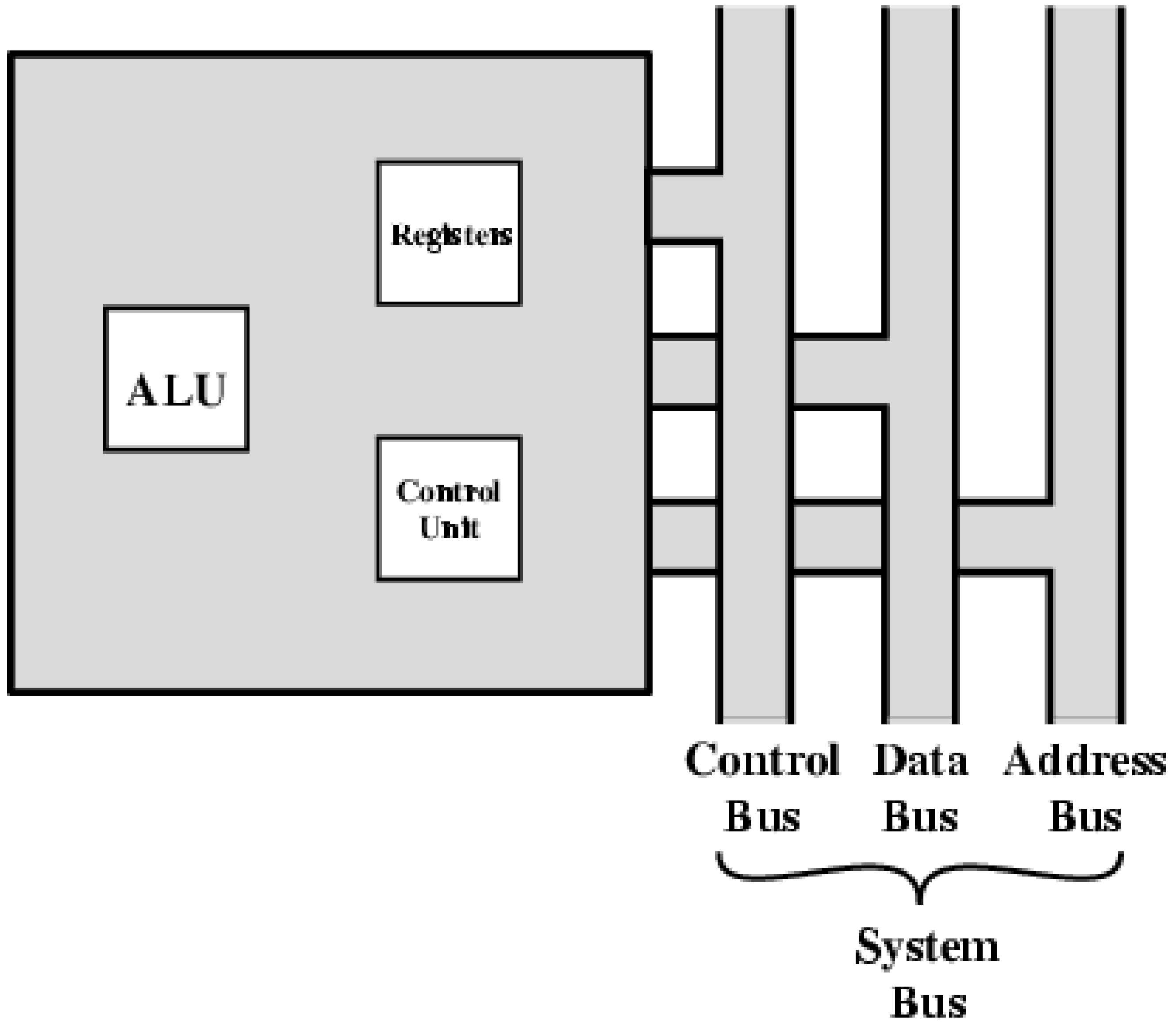
12.1 Processor organization

- CPU must:
  - —Fetch instruction
  - —Interpret instruction
  - —Fetch data
  - —Process data
  - —Write data

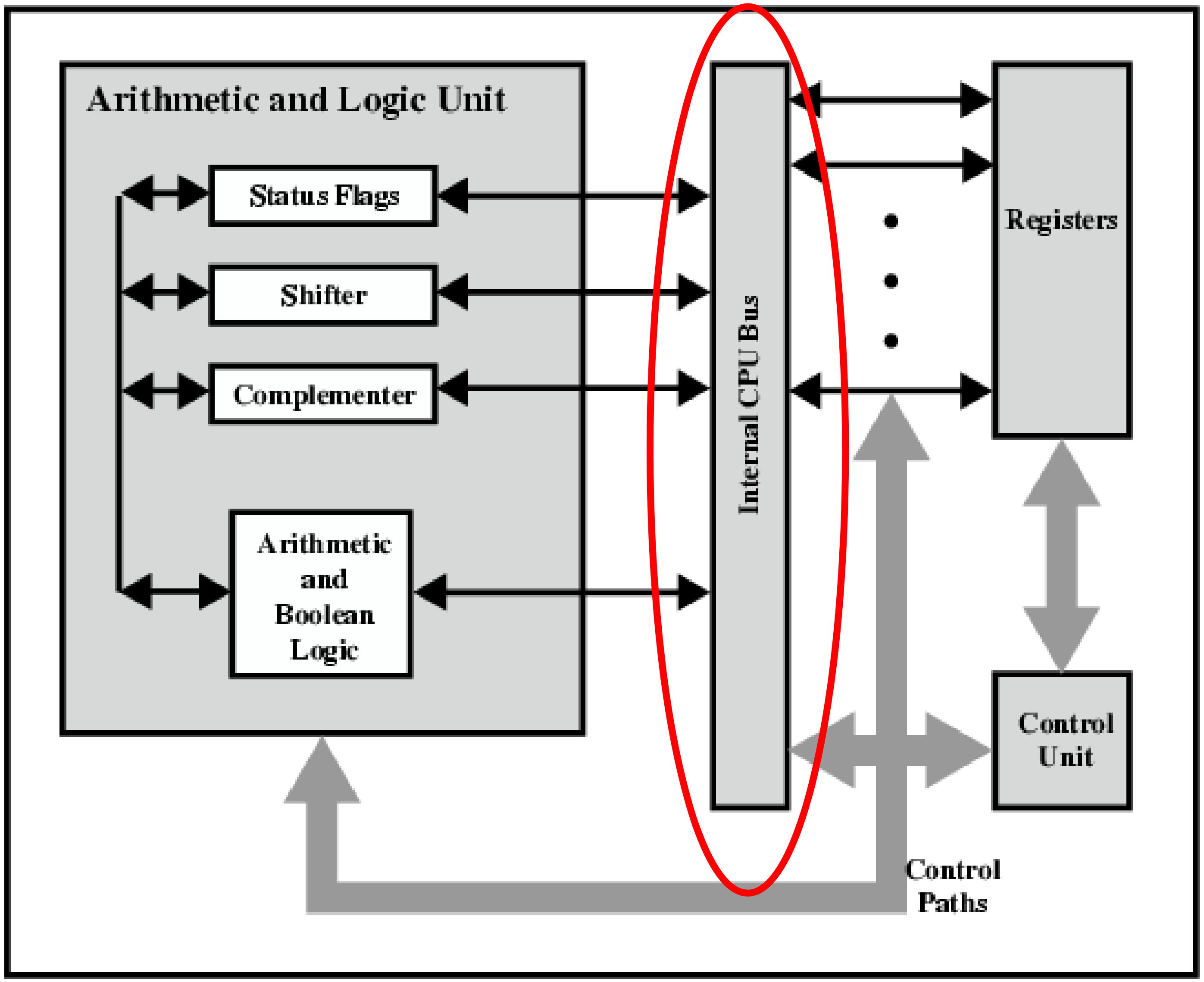


- CPU needs a *small internal memory*
  - —Store *data* temporarily
  - —Locate next *instruction* (*address*)
  - —Store *instructions* temporarily

CPU With Systems Bus



CPU Internal Structure



12.2 Register organization

- User-visible registers （*To whom? Why?*）
- Control and status registers （*Why?*）

Registers

- Top level of *memory hierarchy*
- Temporary storage for CPU
- *Number* and *function* vary between processor designs
- One of the major design decisions

User Visible Registers

- General Purpose
- Data
- Address
- Condition Codes

General Purpose Registers (1)

- True general purpose: *orthogonal to operation* （*any Opcode to any Operand*）
- Restricted

  e.g. dedicated for *floating-point* and *stack* operations
- *Data* registers
- *Address* registers
  - —Segment pointers
  - —Index registers
  - —Stack pointer

Generalized or specialized

- *Reg* $\Longleftrightarrow$ *Instruction set*
- Make them general purpose (*explicit* Operand)
  - —Increase flexibility and programmer options
  - —Increase instruction size & complexity
- Make them specialized (*implicit* Operand)
  - —Smaller (faster) instructions
  - —Less flexibility

How Many GP Registers?

- *Reg* $\Longleftrightarrow$ *Instruction set*
- Between 8 - 32
- Fewer $\Longrightarrow$ more memory references
- More *does not* reduce memory references and takes up processor real estate
- See also RISC

How big?

- Large enough to hold full *address*
- Large enough to hold full *word*
- Often possible to combine two data registers
  - C++ programming
  - int a; (32bits, 4bytes)
  - double int a; (64bits, 8bytes)
  - long double a; (96bits, 12bytes)

Condition Code Registers

- Sets of individual bits
  - e.g. result of last operation was zero, positive, negative, or overflow
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

Register save and restore

- For *call & return* procedure,

  When “Call”: automatic saving

  When “Return”: automatic restoring

- Others,

  Saving and restoring by user

Control & Status Registers

- Program Counter
  - —Always *points to* the next instruction
- Instruction Register
  - —Contains *instruction*, to decode and analyze
- Memory Address Register
  - —Contains *memory address*, and connects to address bus
- Memory Buffer Register
  - —Contains *data*, and connects to data bus

Program Status Word

- A set of bits
- Includes *Condition Codes*
- Sign of last result
- Zero
- Carry
- Equal
- Overflow
- Interrupt enable/disable
- Supervisor

Supervisor Mode

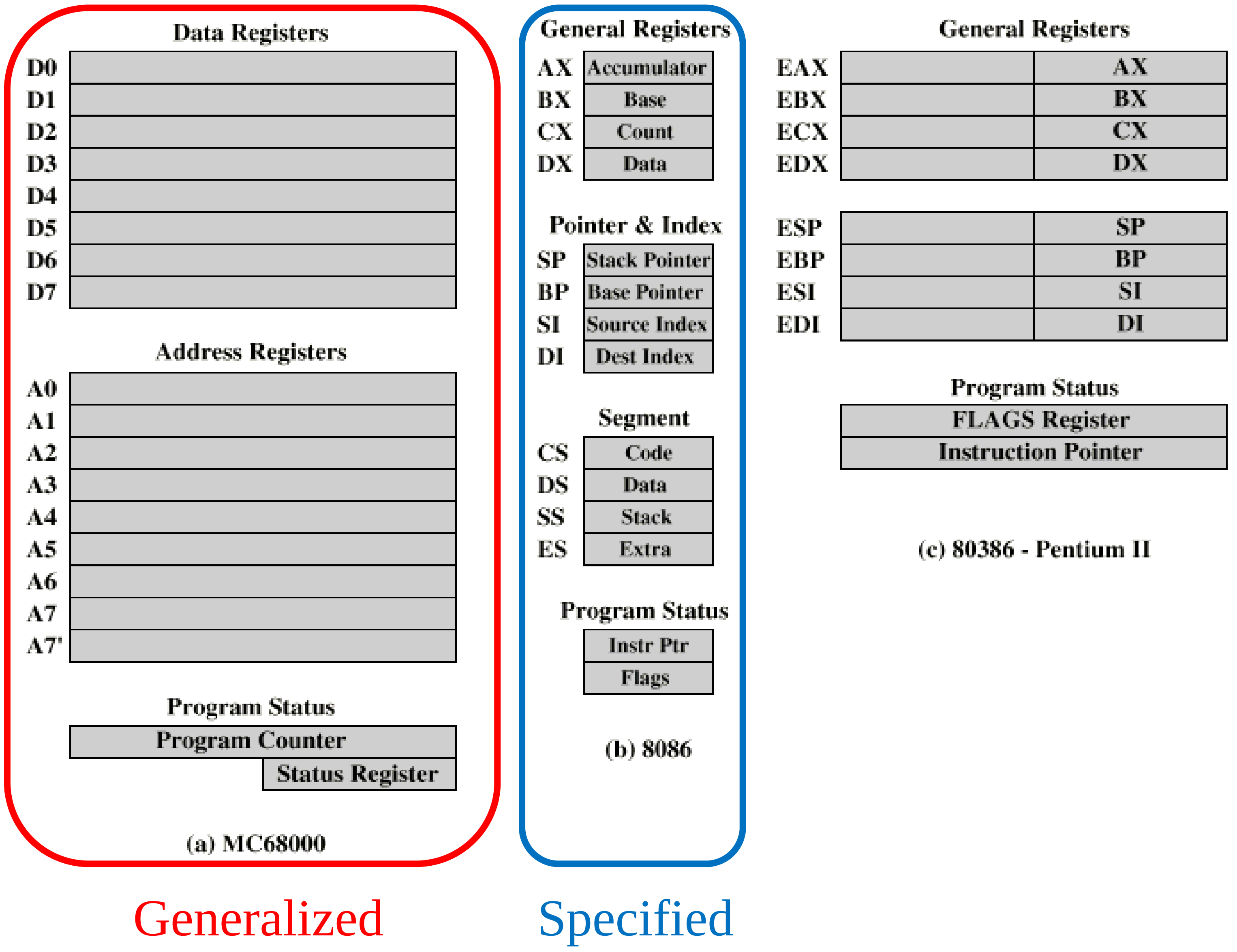
- Intel ring zero
- Kernel mode
- Allows *privileged instructions* to execute
- Used by operating system
- Not available to user programs

Other Registers

- May have registers pointing to:
  - —Process control blocks (see O/S)
  - —Interrupt Vectors (see O/S)

- CPU design and operating system design are closely linked （*compatibility*）

Example Register Organizations



Homework

- Reading Chapter 12
- Key Terms

- Review Questions: 5,6,7

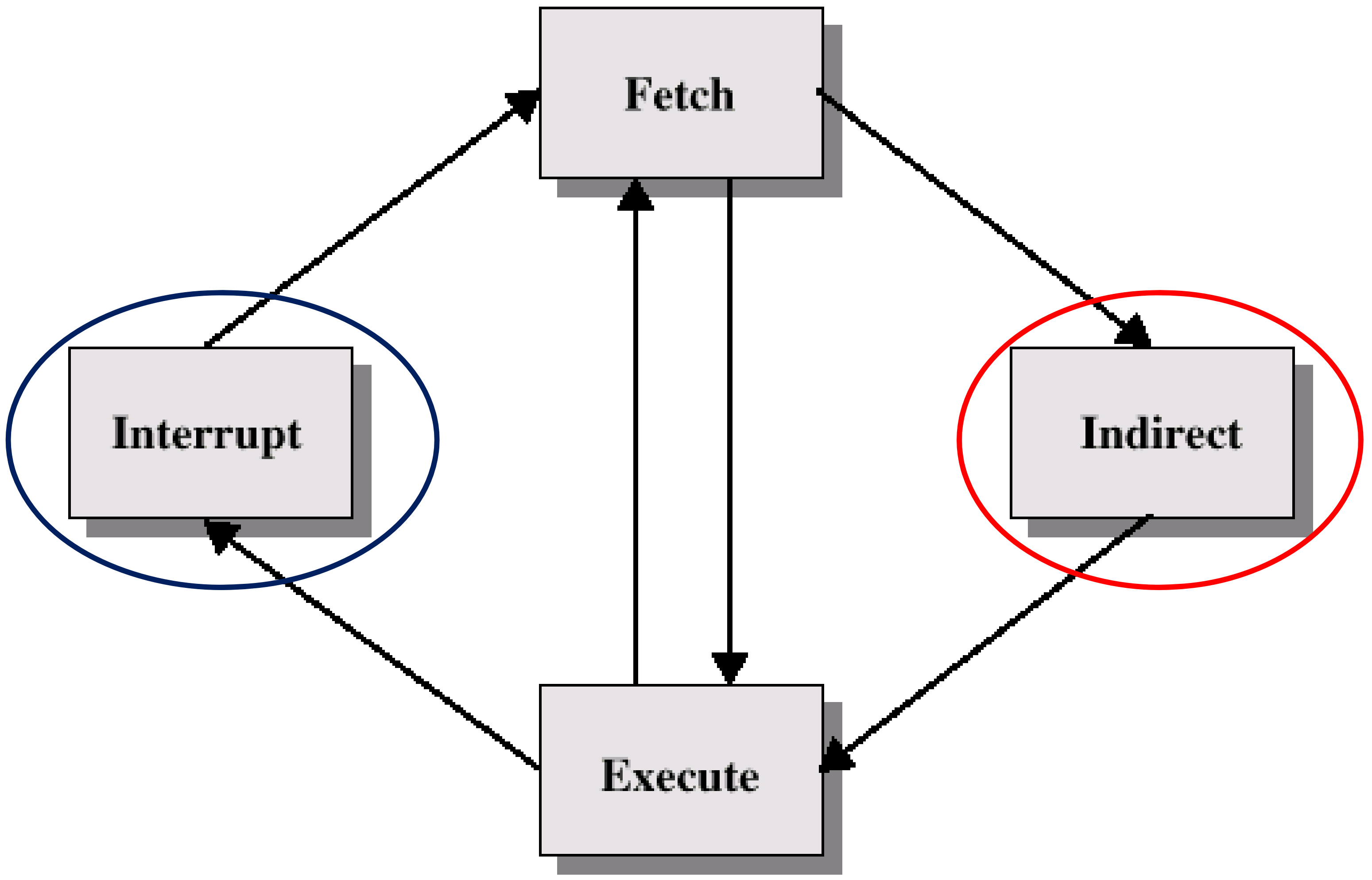
12.3 Instruction Cycle

- Fetch
- Execute
- Interrupt

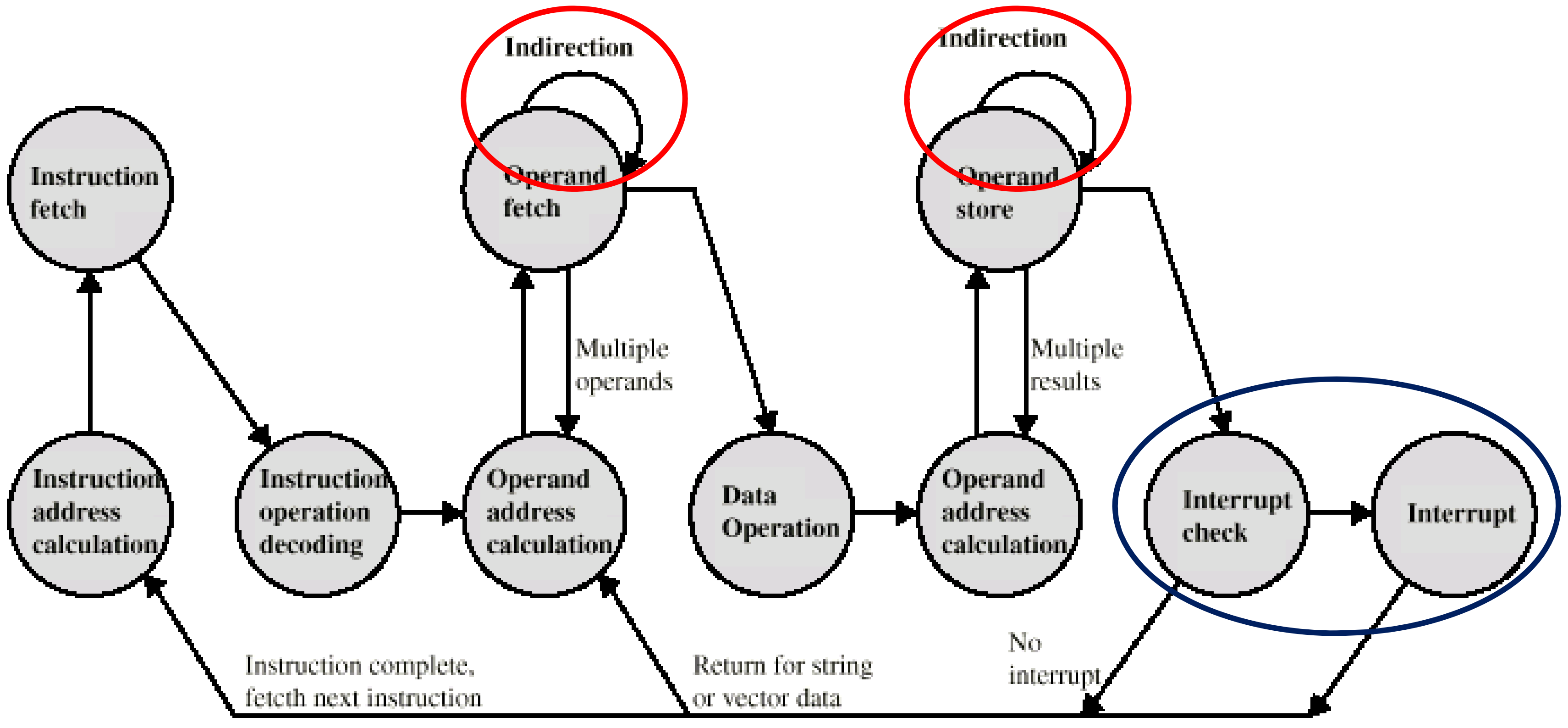
Indirect Cycle

- May require memory access to fetch operands
- Indirect addressing requires more memory accesses
- Can be thought of as *additional instruction subcycle*

Instruction Cycle with Indirect



Instruction Cycle State Diagram

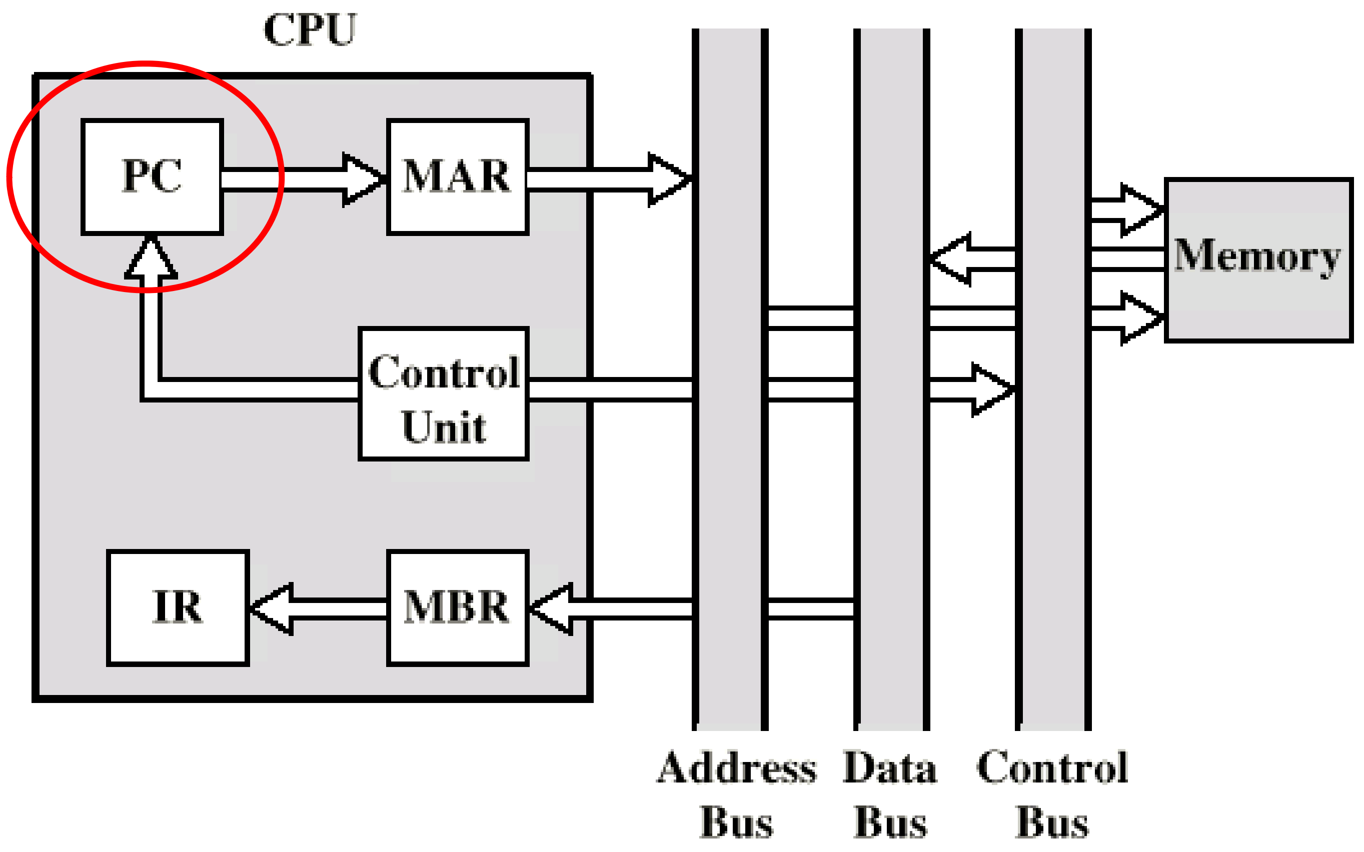

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

Data Flow (Instruction Fetch)

- Depends on *CPU design*
- In general: the followings must happen

- Fetch Instruction
  - —*PC* contains address of next instruction
  - —Address moved to *MAR*
  - —Address placed on address bus
  - —Control unit requests memory read
  - —Result placed on data bus, copied to *MBR*, then to *IR*
  - —Meanwhile *PC* incremented by 1

Data Flow (Instruction Fetch)

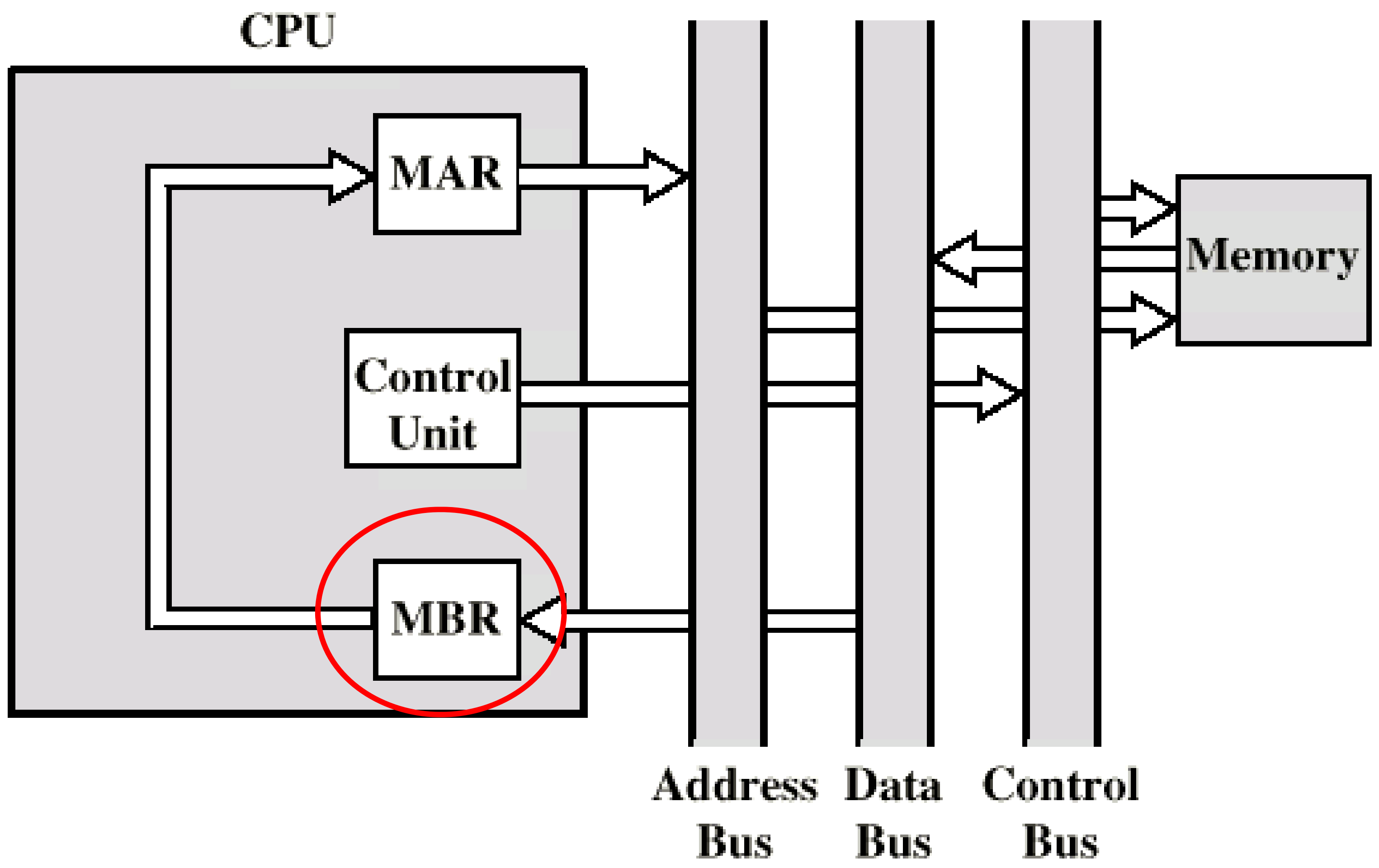


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Data Flow (Indirect)

- *IR* is examined
- If indirect addressing, *indirect cycle* is performed
  - —*Right most N bits* （*why?*） of **MBR** transferred to **MAR**
  - —*Control unit* requests memory read
  - —Result (address of operand) moved to **MBR**

Data Flow (Indirect Diagram)



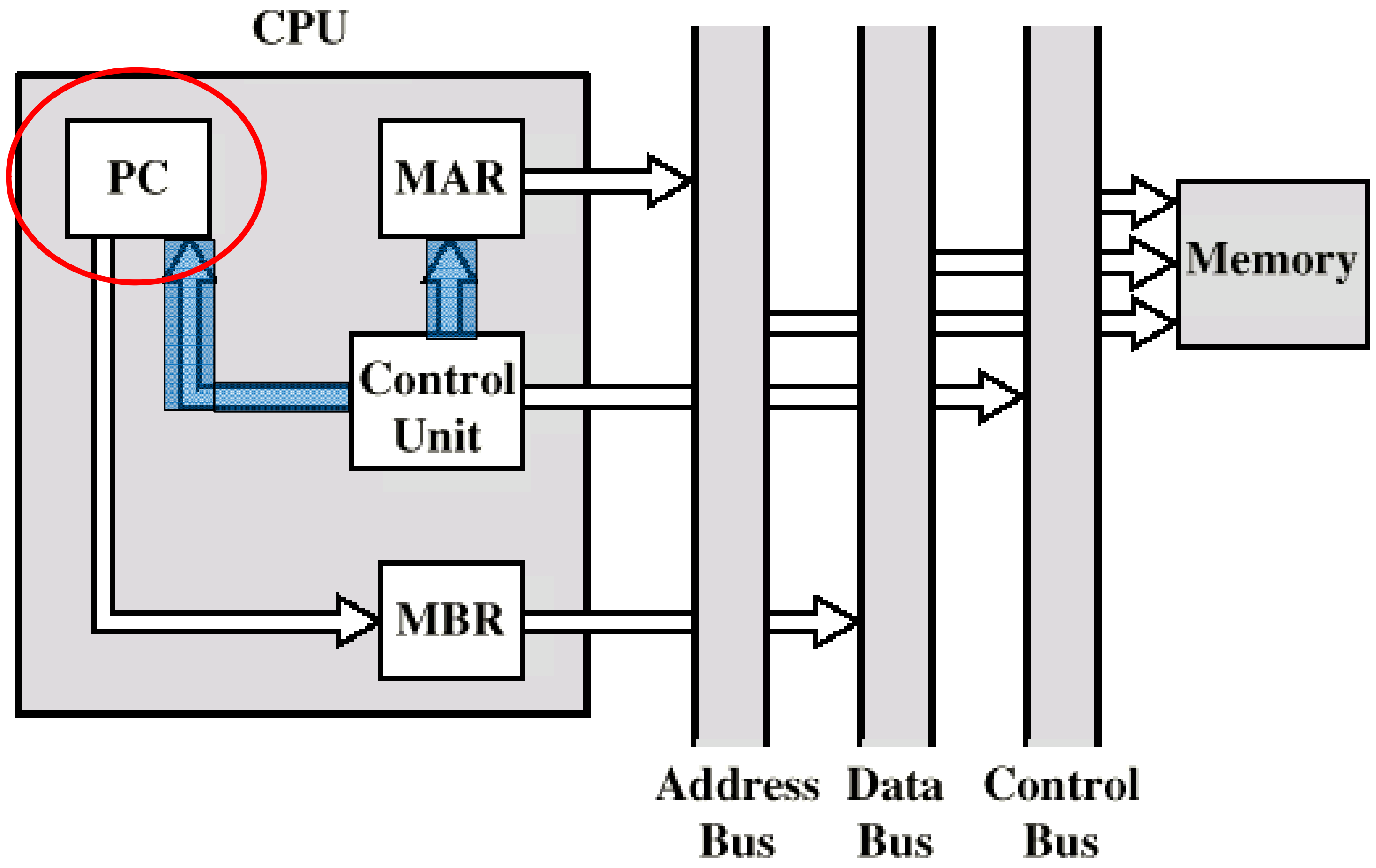
Data Flow (Execute)

- May take many forms
- Depends on instruction being executed
- May include
  - —Memory read/write
  - —Input/Output
  - —Register transfers
  - —ALU operations

Data Flow (Interrupt)

- Current PC saved to allow resumption after interrupt
- Contents of *PC copied to MBR*
- *Special memory location* (e.g. *stack pointer*) loaded to MAR
- MBR written to *memory*
- PC loaded with address of *interrupt handling routine*
- Next instruction (first of interrupt handler) can be fetched

Data Flow (Interrupt Diagram)



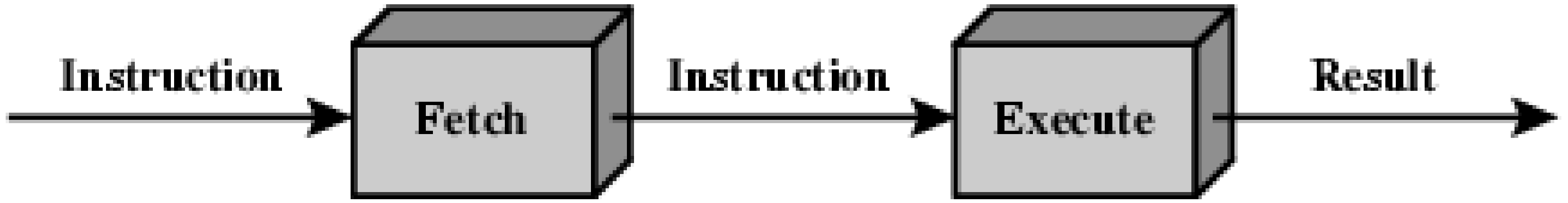
12.4 Instruction pipelining

- Pipelining strategy
- Pipeline performance
- Dealing with branches

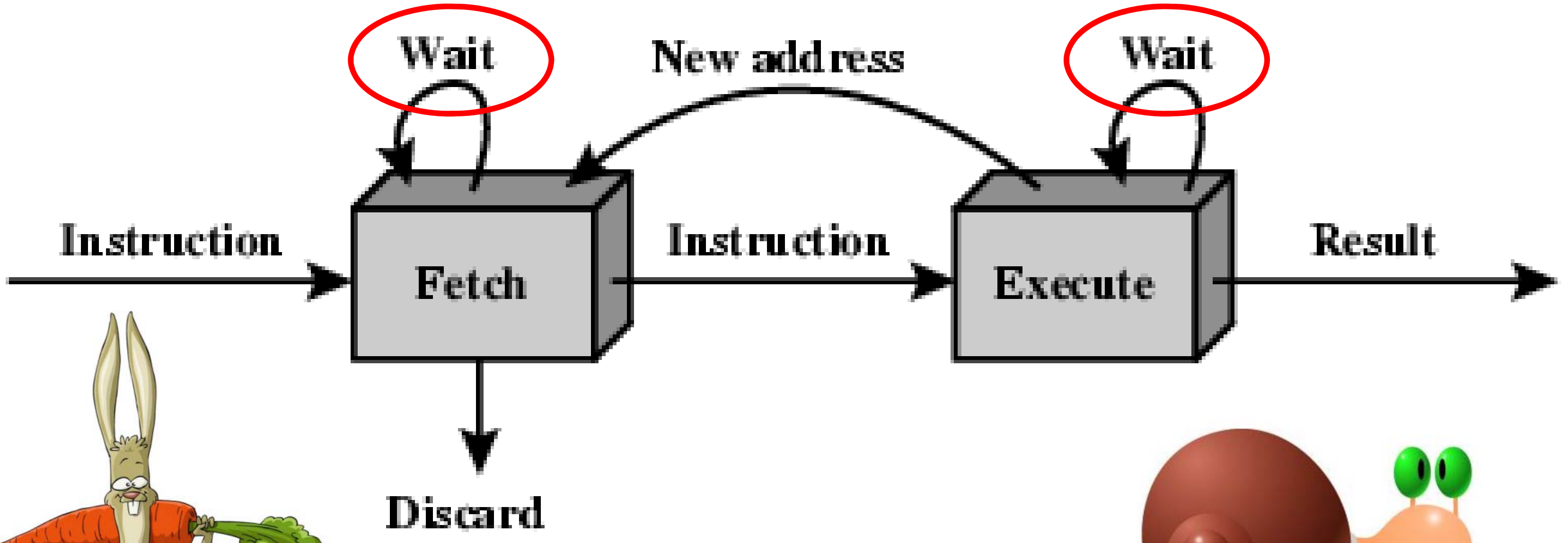
Pipelining strategy

- Similar to assembly line
- Stages work simultaneously

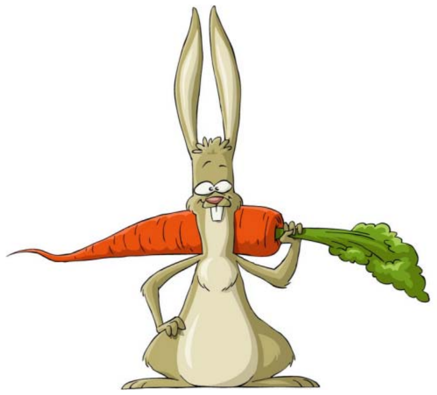
Two Stage Instruction Pipeline

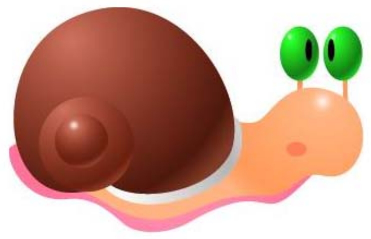


(a) Simplified view



(b) Expanded view

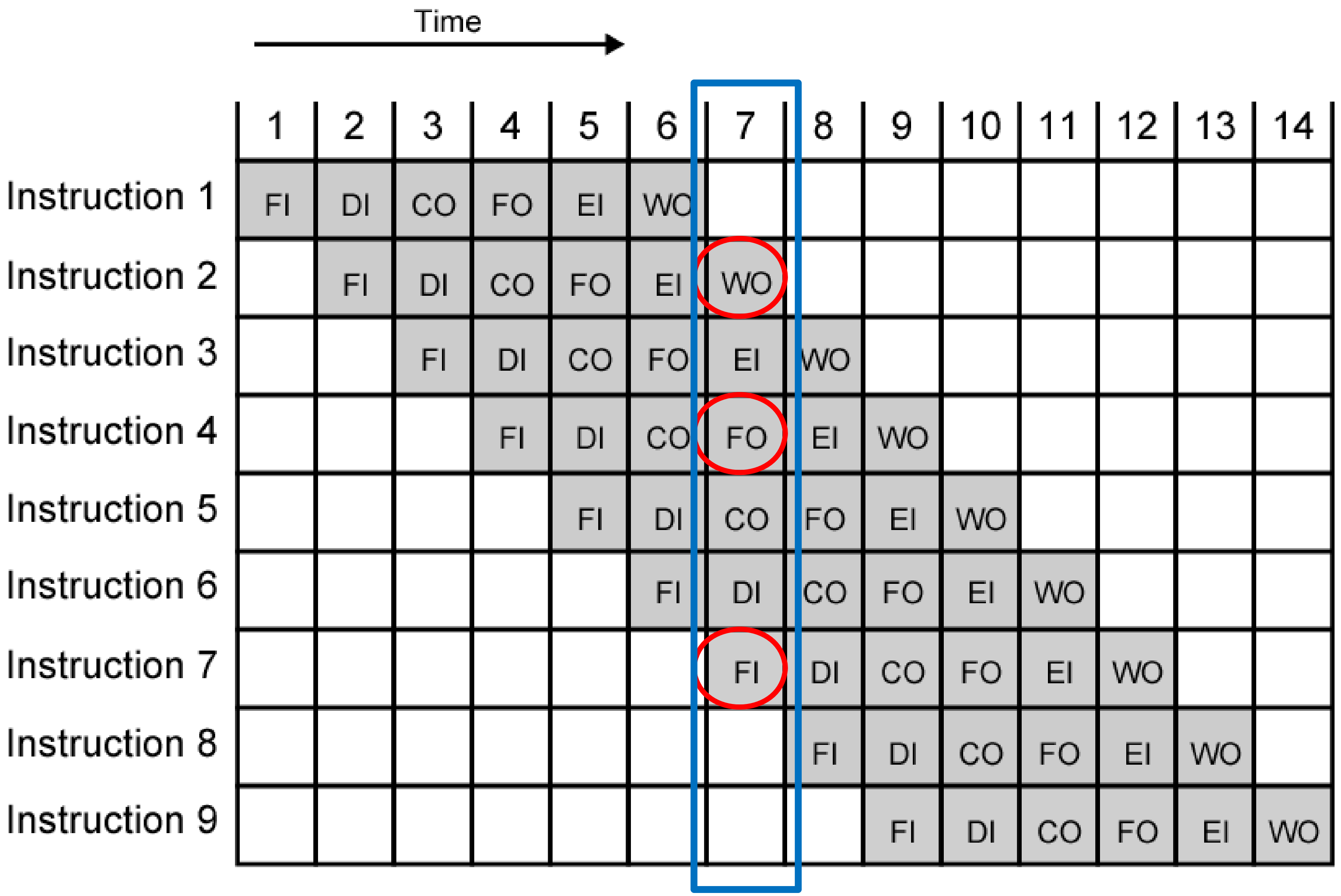




Decomposition of instruction processing

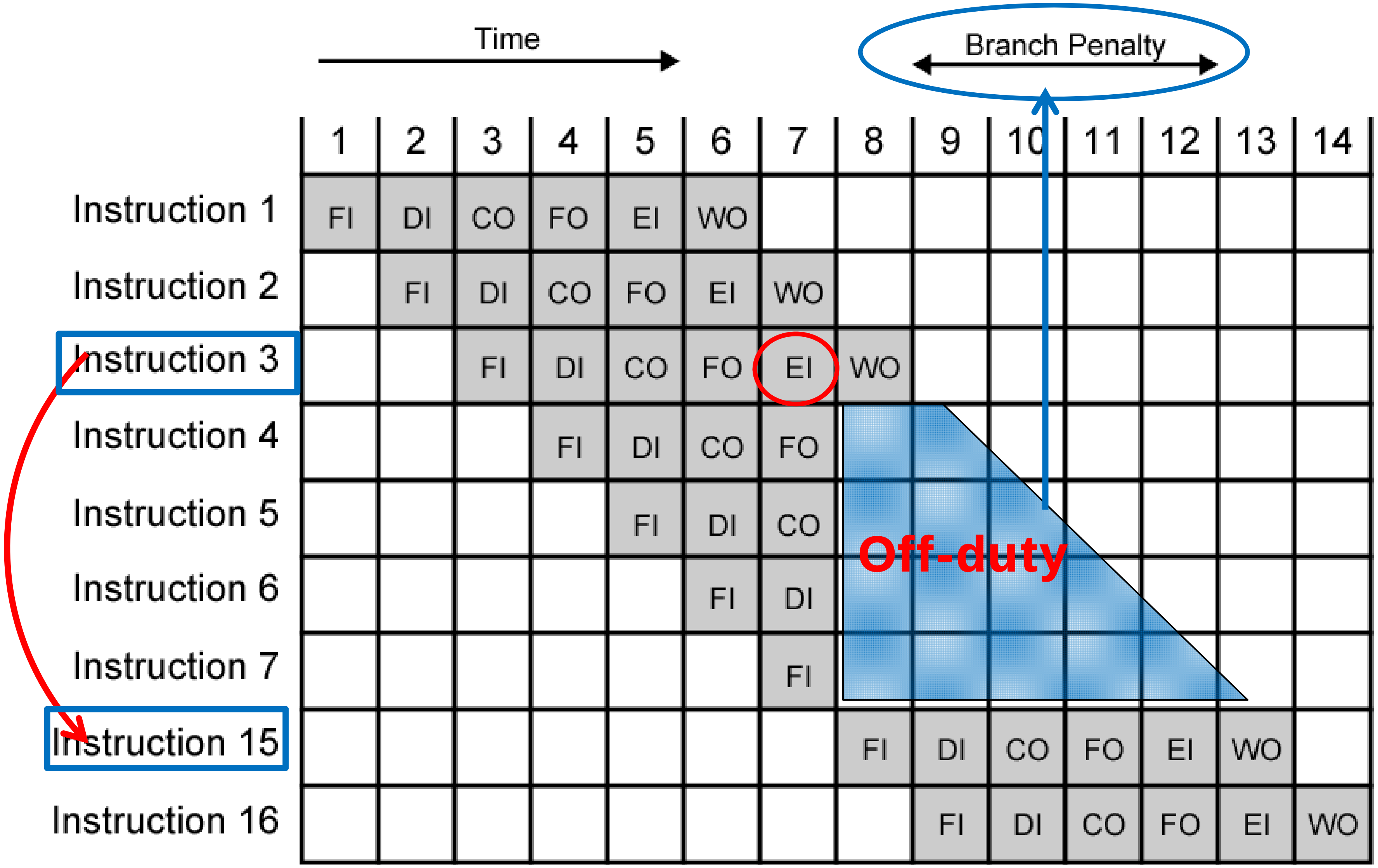
- Fetch instruction (FI)
- Decode instruction (DI)
- Calculate operands (CO)
- Fetch operands (FO)
- Execute instructions (EI)
- Write operand (WO)

Timing Diagram for Instruction Pipeline Operation



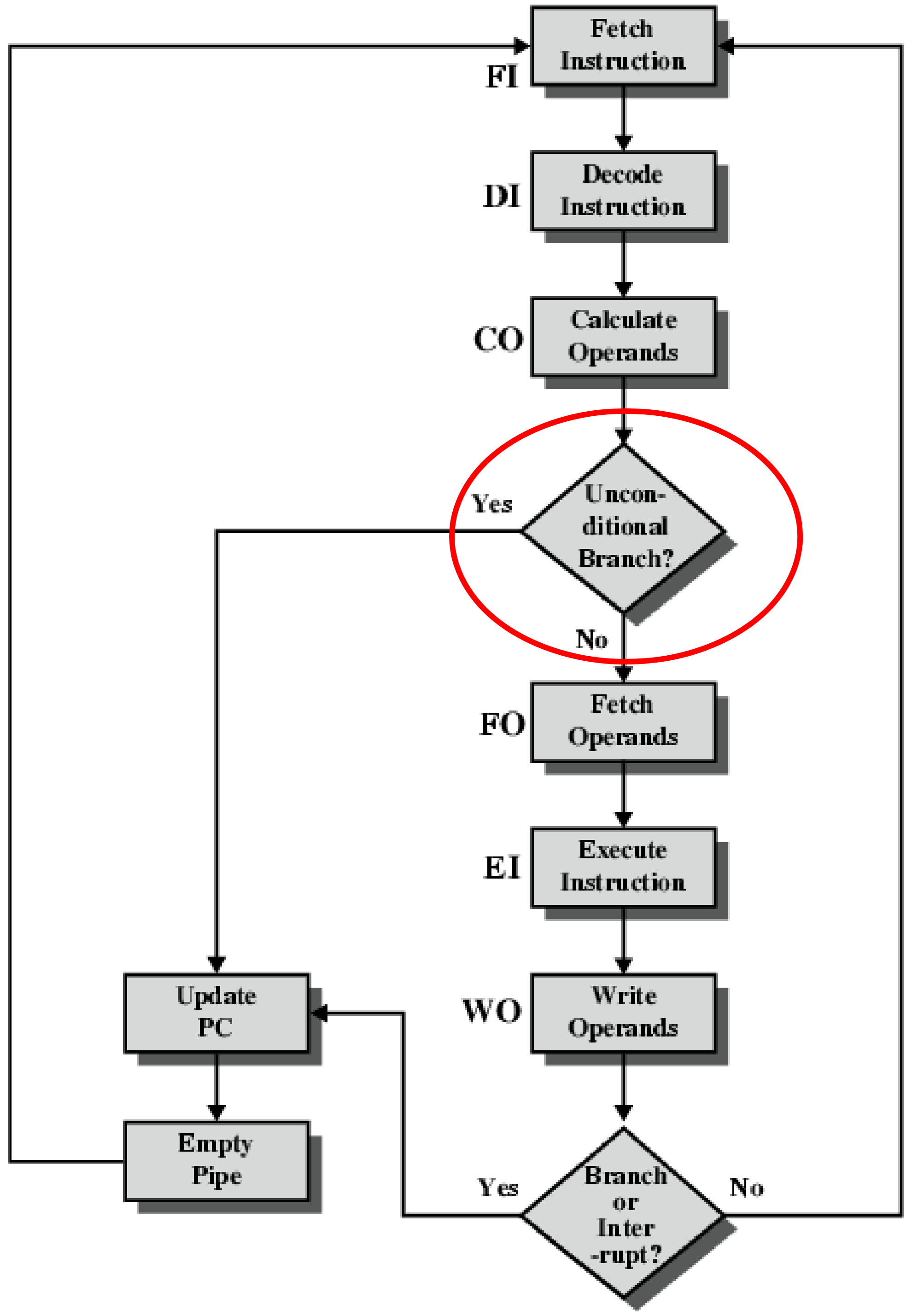
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO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

The Effect of a Conditional Branch on Instruction Pipeline Operation

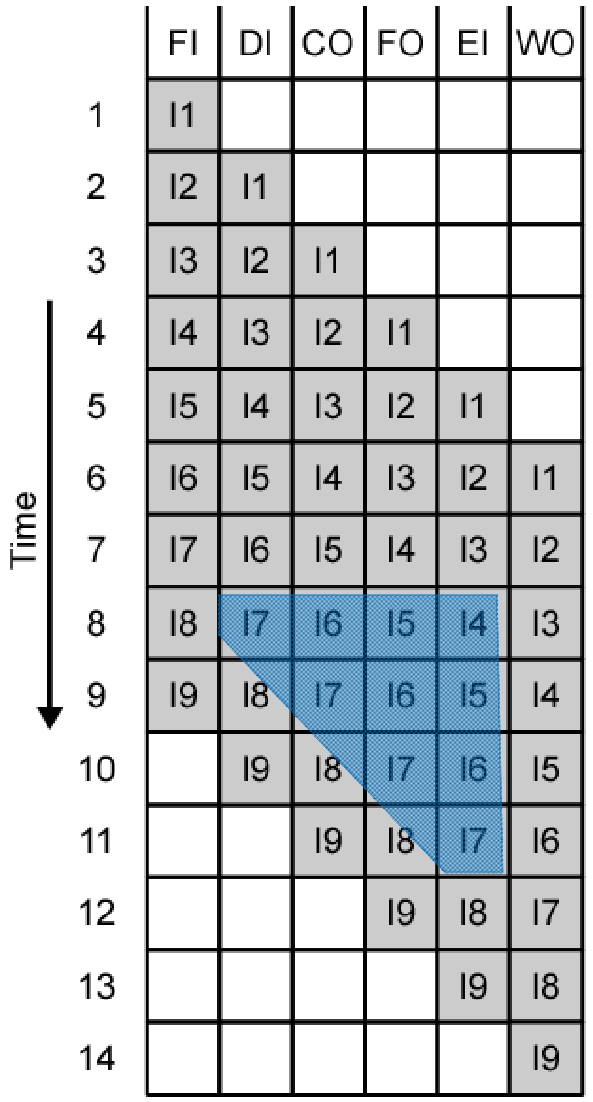


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

Six Stage Instruction Pipeline

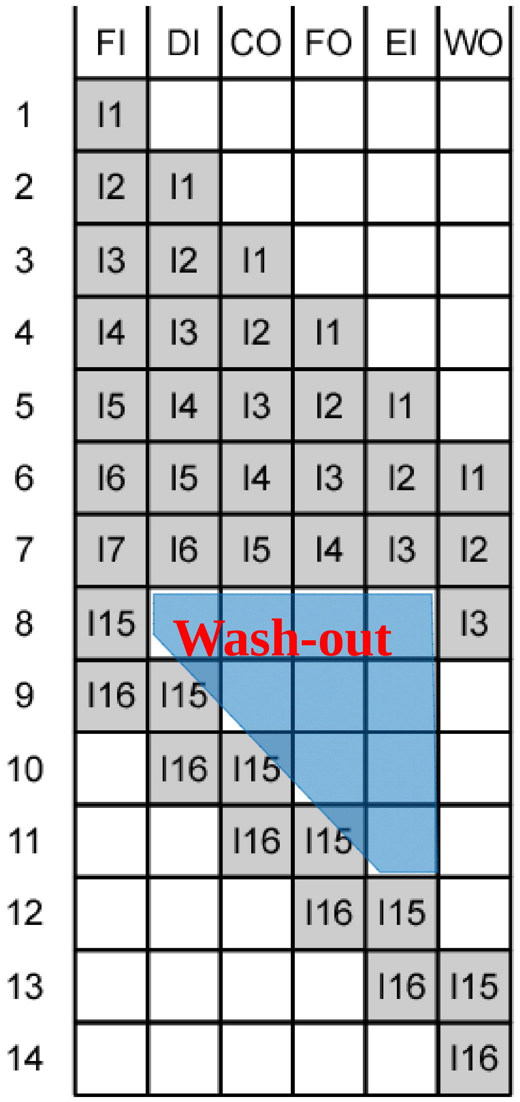


Alternative Pipeline Depiction



	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

(a) No branches



	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

(b) With conditional branch

Pipeline Performance

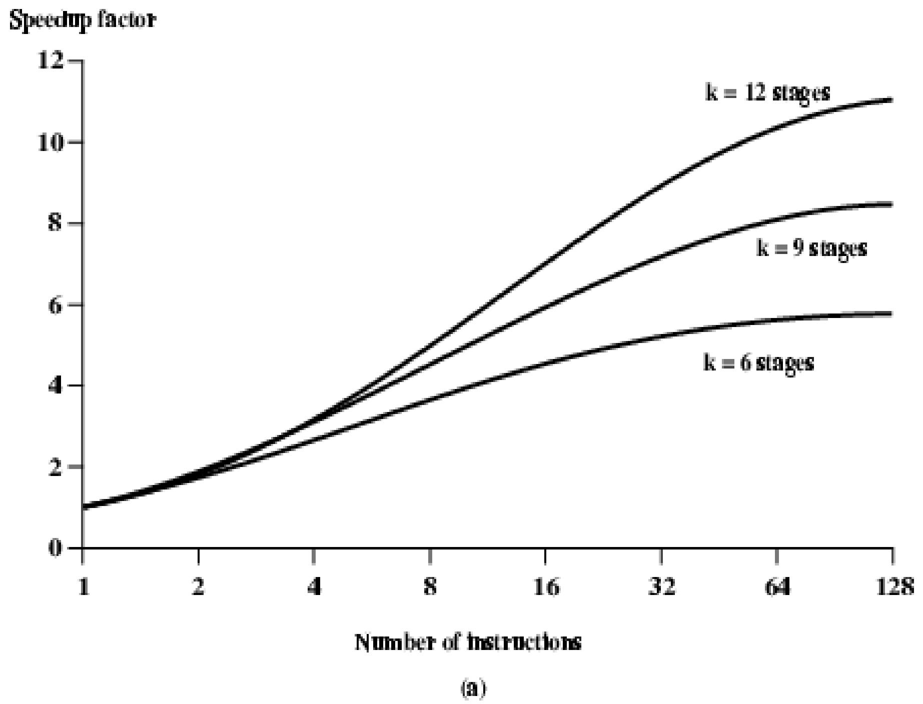
- *n* Instructions, *k* stages, no branches

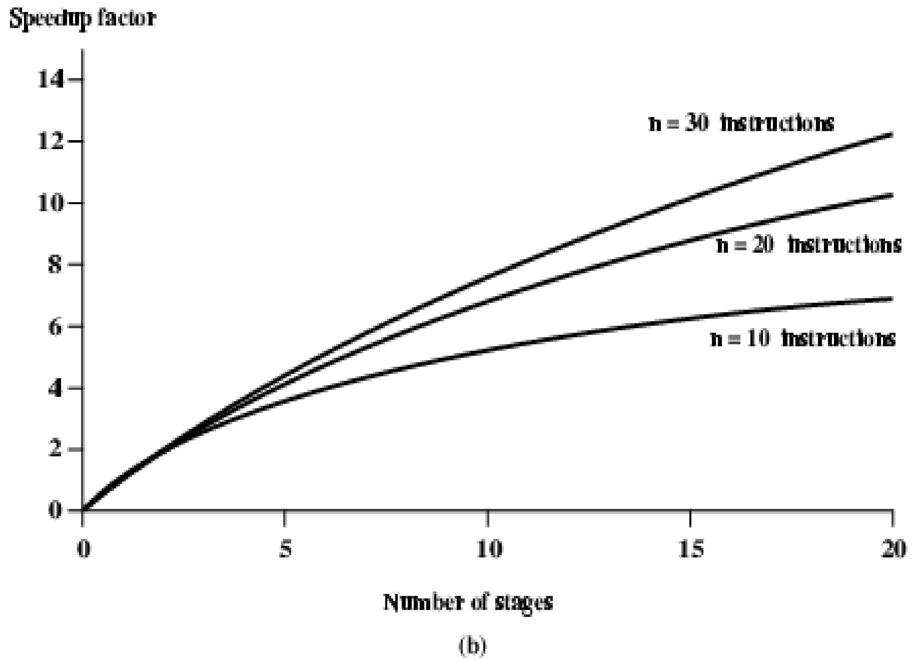
$$T_k = [k + (n - 1)]\tau$$

- Speedup factor

$$S_k = \frac{nk\tau}{[k + (n - 1)]\tau} = \frac{nk}{k + (n-1)}$$

- *The more stages, the better?*





Dealing with Branches

- Multiple Streams
- Prefetch Branch Target
- Loop buffer
- Branch prediction
- Delayed branching

Multiple Streams

- Have two pipelines
- *Prefetch each branch* into a separate pipeline
- Use appropriate pipeline

- Leads to bus & register *contention*
- Multiple branches lead to *further pipelines* being needed

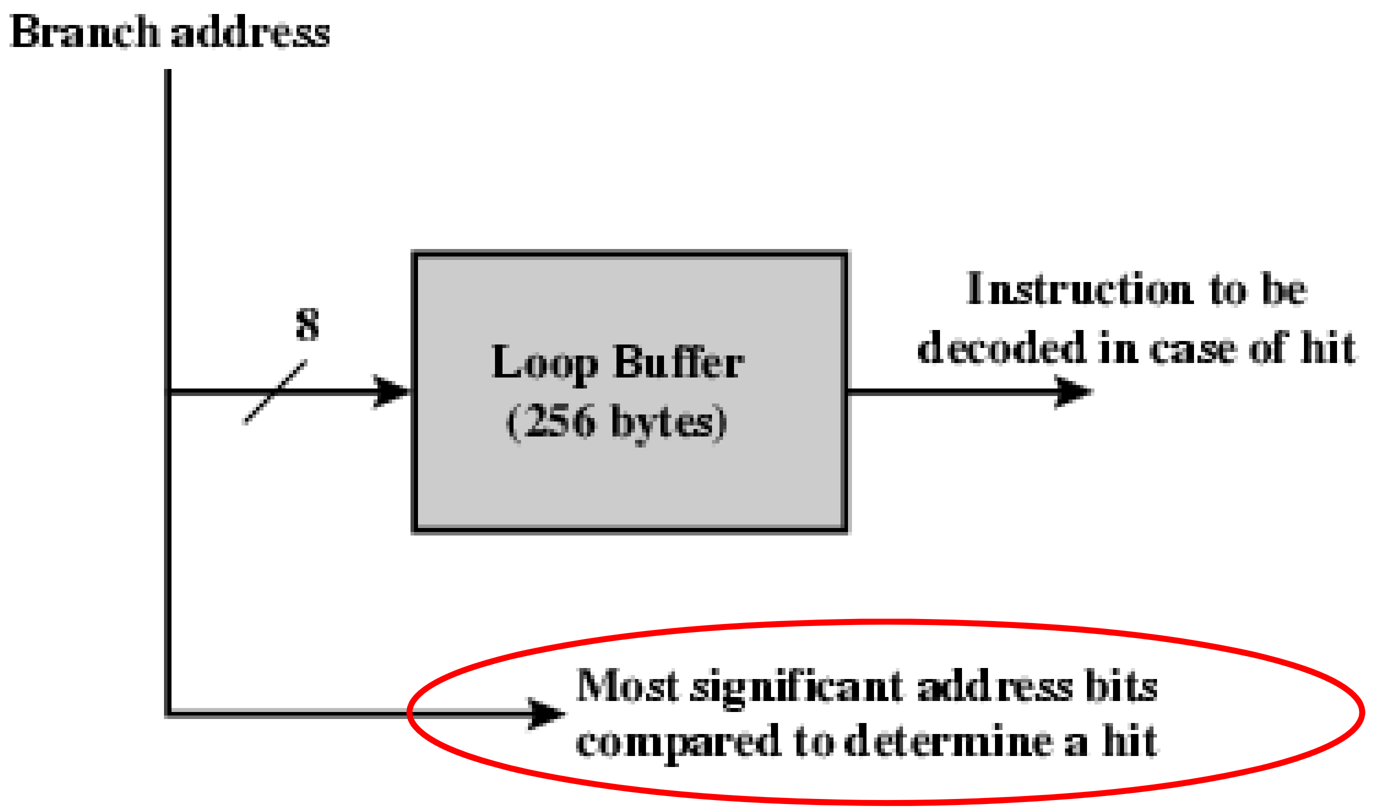
Prefetch Branch Target

- Target of branch is *prefetched* in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91

Loop Buffer

- A small, Very fast memory
- Containing the *n most recently fetched* instructions
- Maintained by *fetch stage* of pipeline
- Check buffer before fetching from memory
- Very good for *small loops* or jumps

Loop Buffer Diagram


Branch address
8
Loop Buffer
(256 bytes)
Instruction to be decoded in case of hit
Most significant address bits compared to determine a hit

Branch Prediction

- Static prediction
  - Not related to branch history
- Dynamic prediction
  - Related to branch history

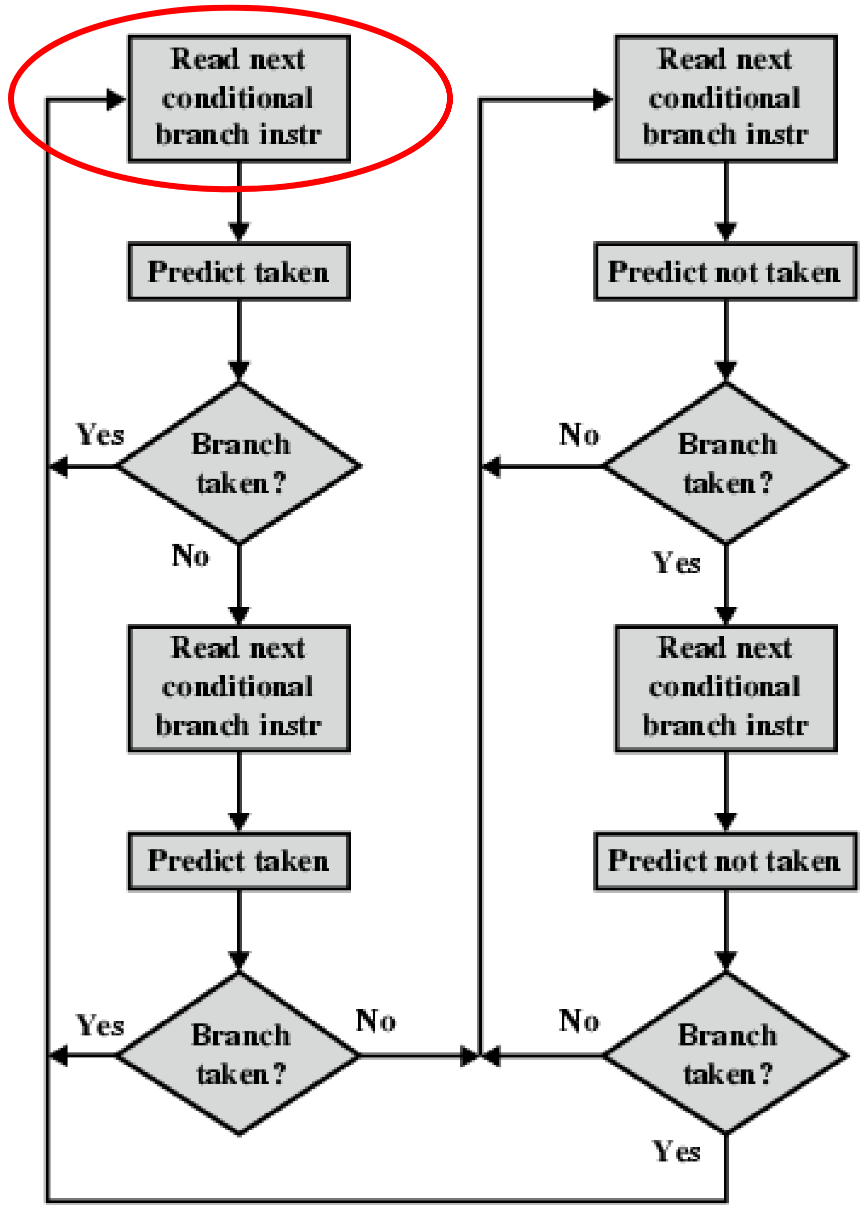
Static prediction

- Predict never taken
  - —Assume that jump will not happen
  - —*Always fetch next instruction*
  - —68020 & VAX 11/780
- Predict always taken
  - —Assume that jump will happen
  - —*Always fetch branch instruction*
  - —May cause page fault
- Predict by Opcode
  - —Some instructions are more likely to result in a jump than others
  - —Can get up to 75% success

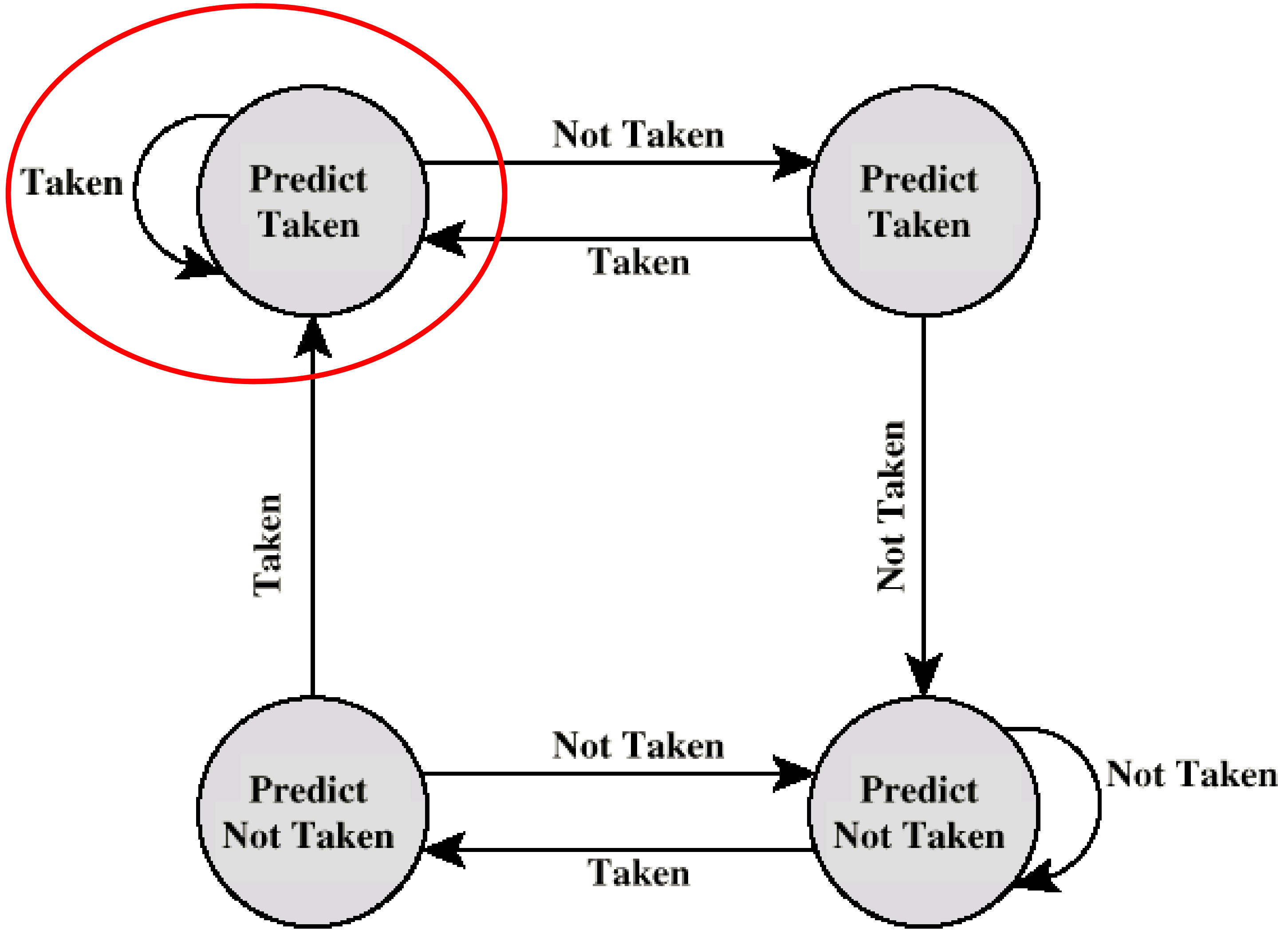
Dynamic Branch Prediction

- Taken/Not taken switch
  - Based on previous history (1 or more bits)
  - Good for loops
  - More than 80% accuracy
- Delayed Branch
  - Do not take jump until you have to
  - Rearrange instructions

Branch Prediction Flowchart


Read next conditional branch instr
Predict taken
Branch taken?
Yes
No
Read next conditional branch instr
Predict taken
Branch taken?
Yes
No
Read next conditional branch instr
Predict not taken
Branch taken?
No
Yes
Read next conditional branch instr
Predict not taken
Branch taken?
No
Yes

Branch Prediction State Diagram



Branch target buffer

- Address of branch instruction
- History bits
- Information about the target instruction (target address or target instruction)

Homework

- Reading paper: Lilja, D., “Reducing the Branch Penalty in Pipelined Processors.” Computer, July 1988.
- Key Terms
- Problems: 7,8,11