

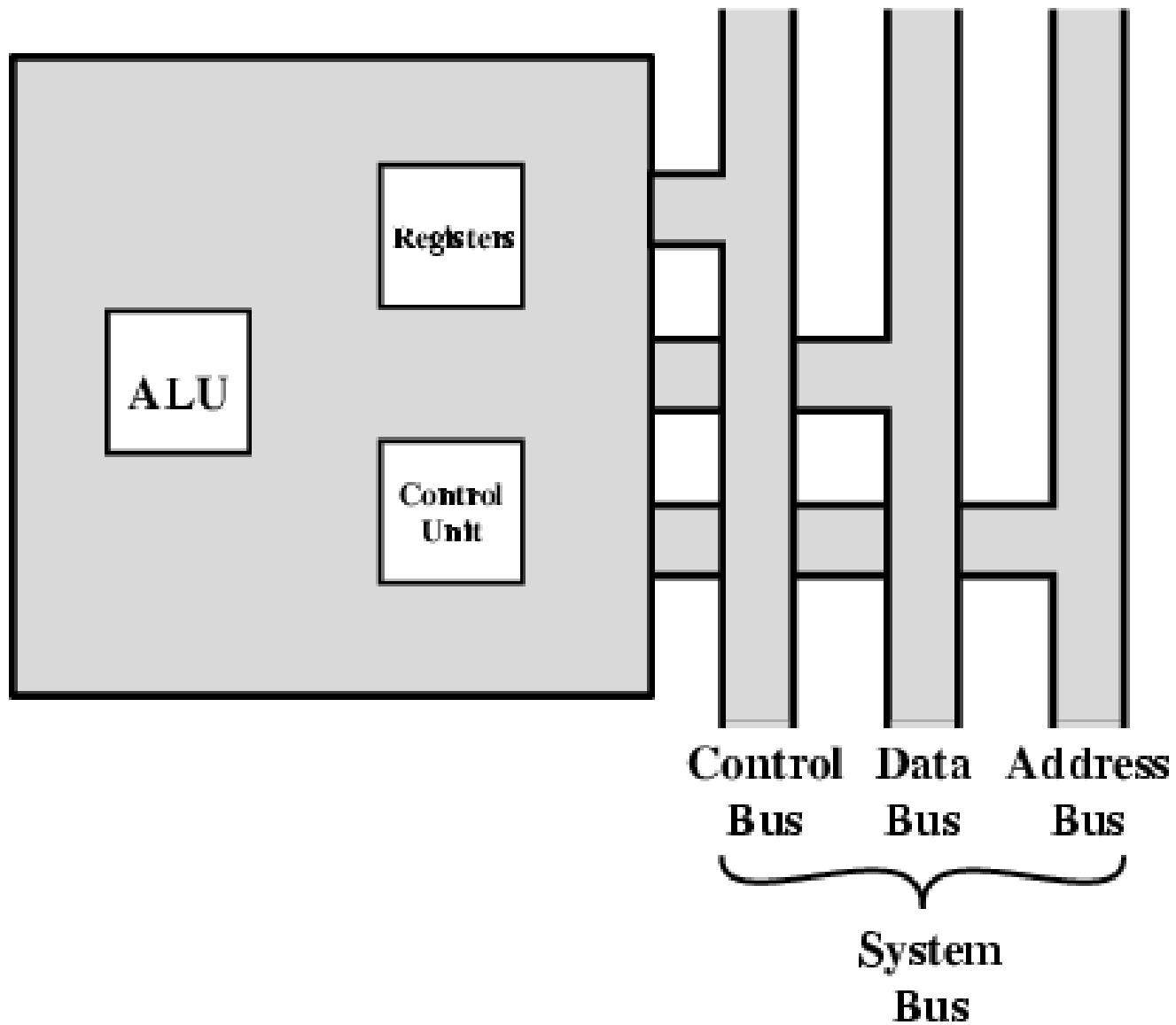
William Stallings
Computer Organization
and Architecture
8th Edition

Chapter 12
Processor Structure and
Function

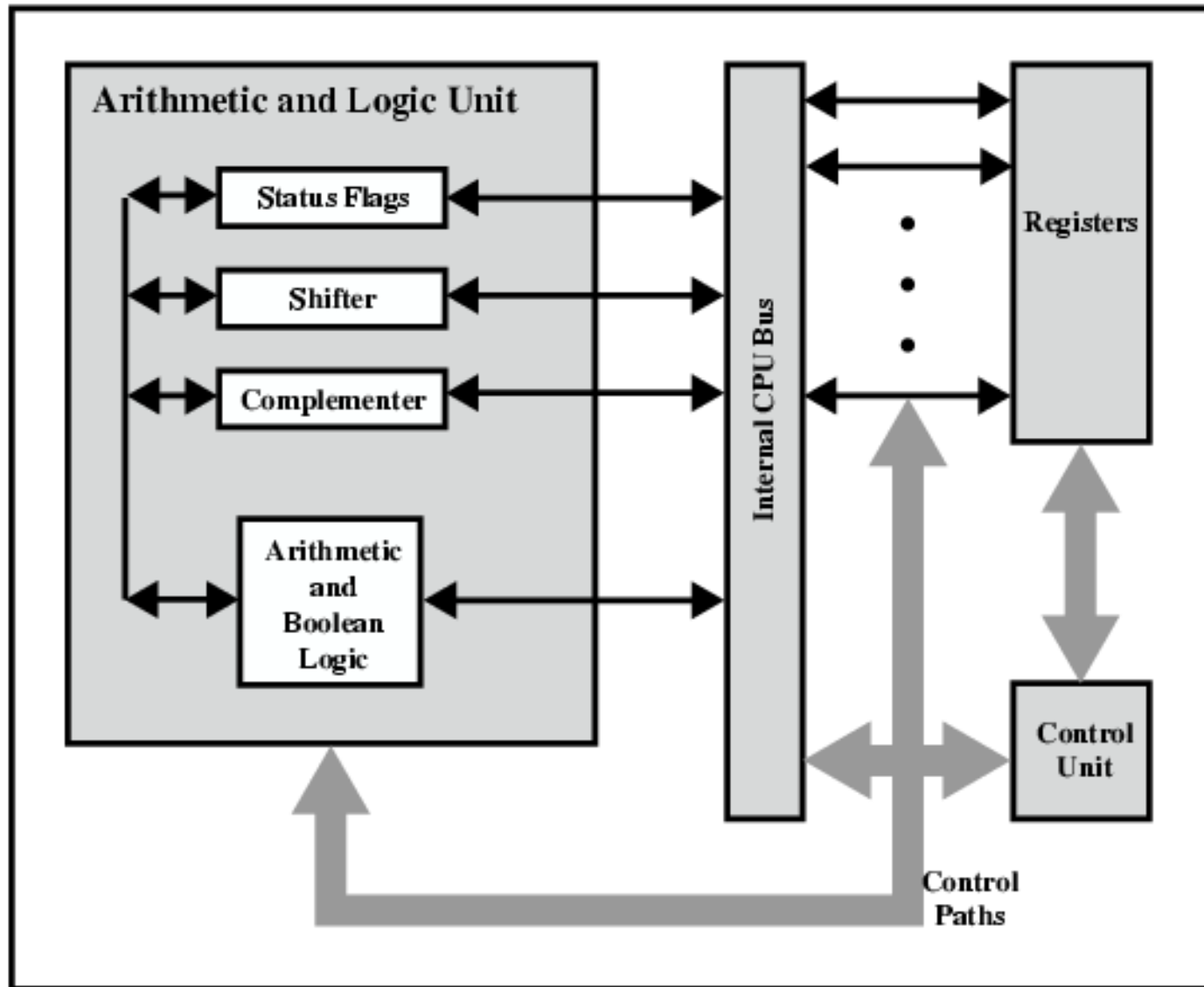
CPU Structure

- CPU must:
 - Fetch instructions
 - Interpret instructions
 - Fetch data
 - Process data
 - Write data

CPU With Systems Bus



CPU Internal Structure



Registers

- CPU must have some working space (temporary storage)
- Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

User Visible Registers

- General Purpose
- Data
- Address
- Condition Codes

General Purpose Registers (1)

- May be true general purpose
- May be restricted
- May be used for data or addressing
- Data
 - Accumulator
- Addressing
 - Segment

General Purpose Registers (2)

- Make them general purpose
 - Increase flexibility and programmer options
 - Increase instruction size & complexity
- Make them specialized
 - Smaller (faster) instructions
 - Less flexibility

How Many GP Registers?

- Between 8 - 32
- Fewer = 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
 - `double int a;`
 - `long int a;`

Condition Code Registers

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

Control & Status Registers

- Program Counter
- Instruction Decoding Register
- Memory Address Register
- Memory Buffer Register

- Revision: what do these all do?

Program Status Word

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

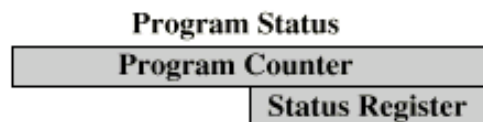
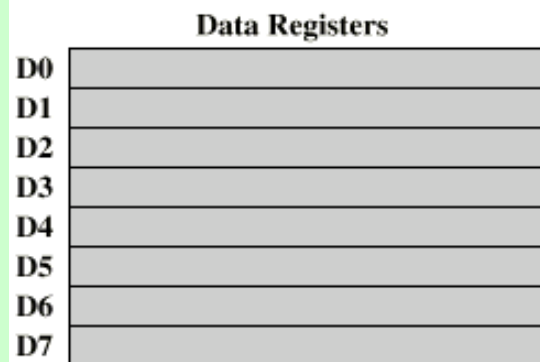
Supervisor Mode

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

Other Registers

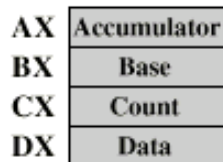
- May have registers pointing to:
 - Process control blocks (see O/S)
 - Interrupt Vectors (see O/S)
- N.B. CPU design and operating system design are closely linked

Example Register Organizations

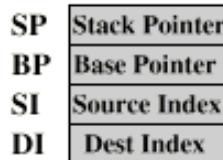


(a) MC68000

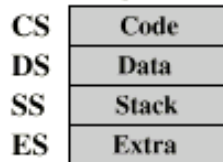
General Registers



Pointer & Index



Segment

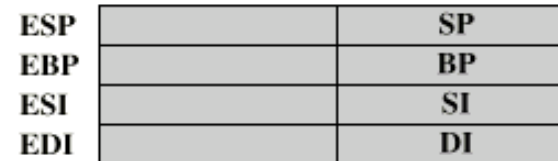
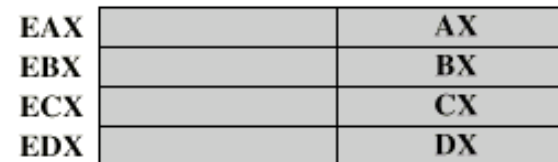


Program Status



(b) 8086

General Registers



Program Status



(c) 80386 - Pentium II

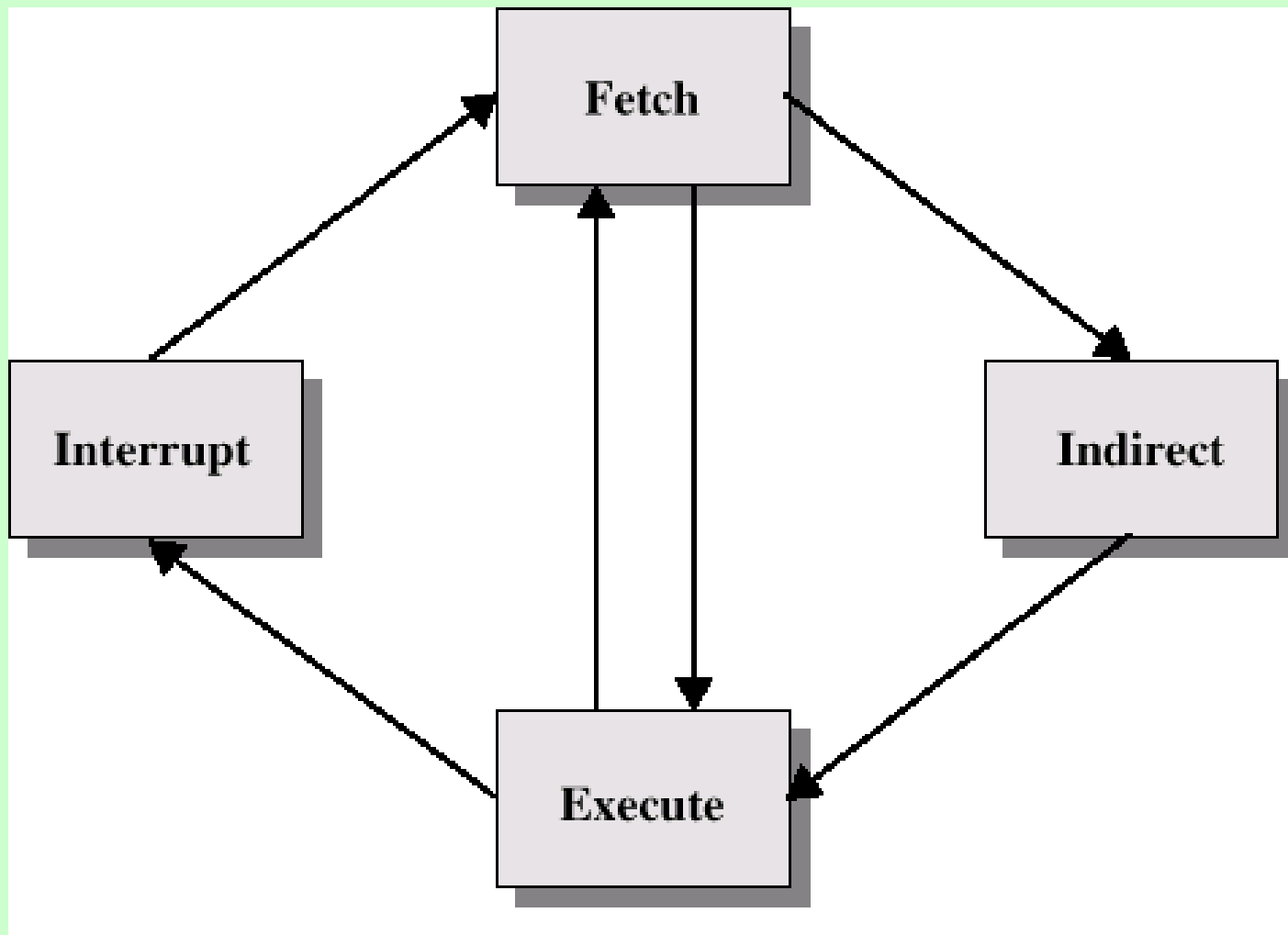
Instruction Cycle

- Revision
- Stallings Chapter 3

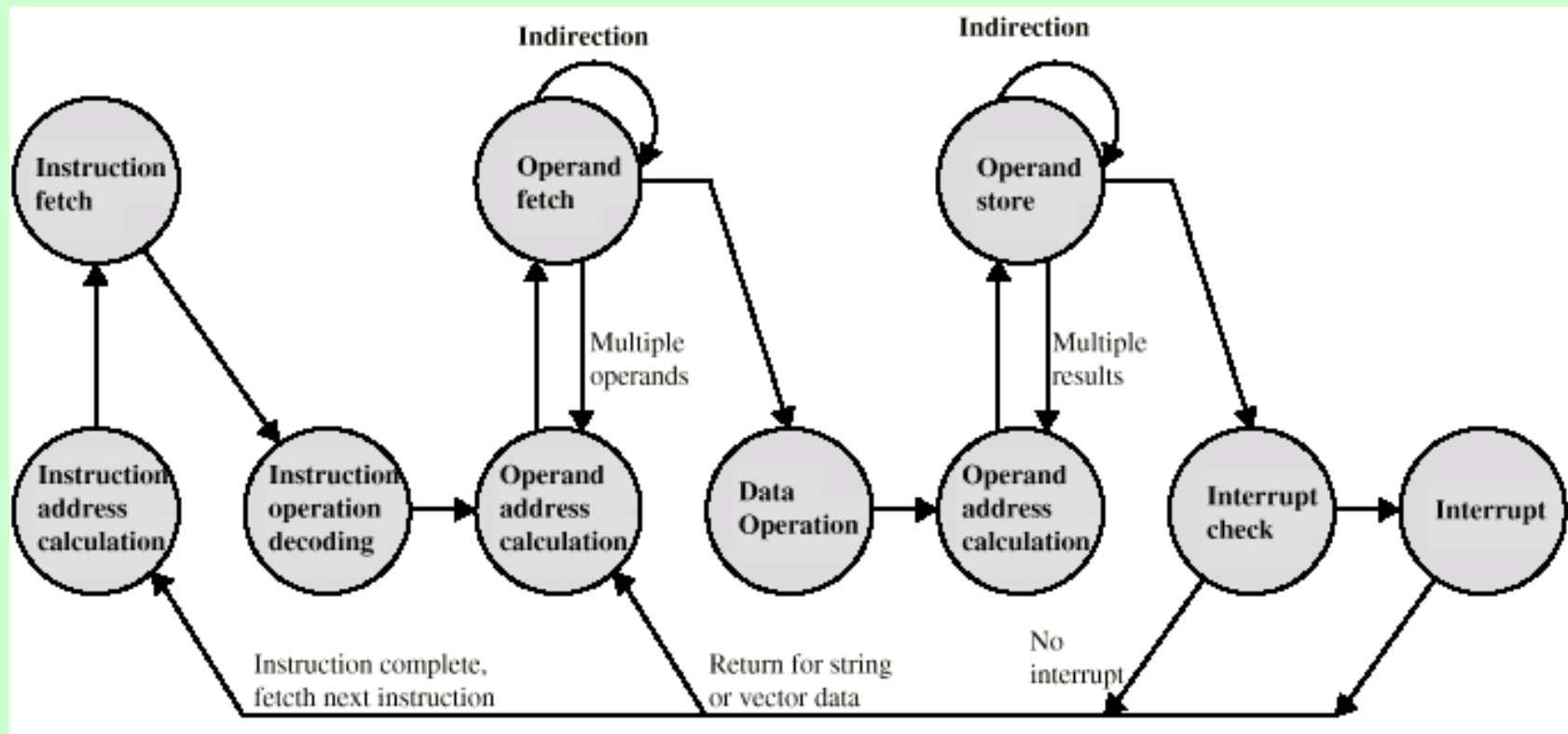
Indirect Cycle

- 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



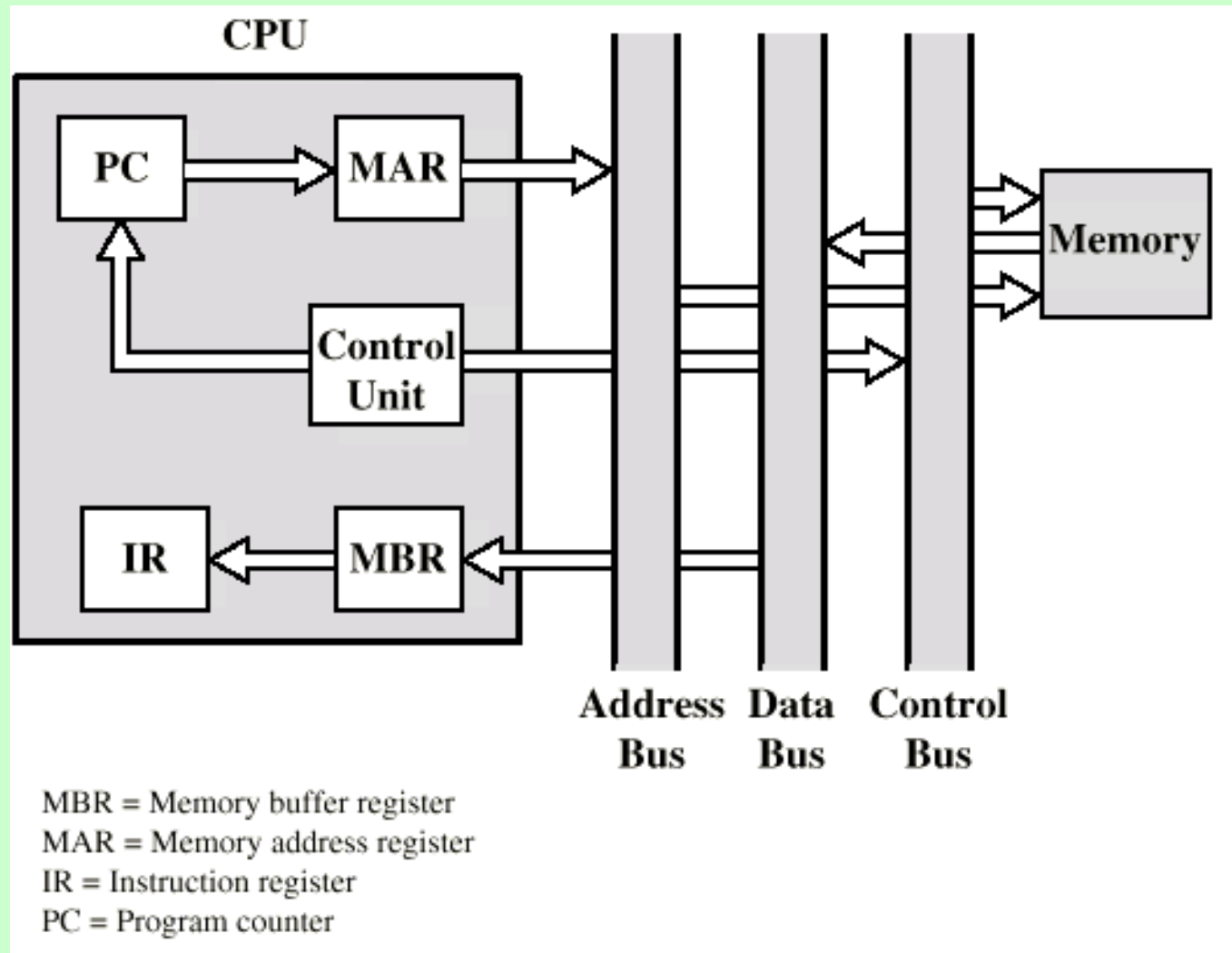
Data Flow (Instruction Fetch)

- Depends on CPU design
- In general:
- Fetch
 - 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

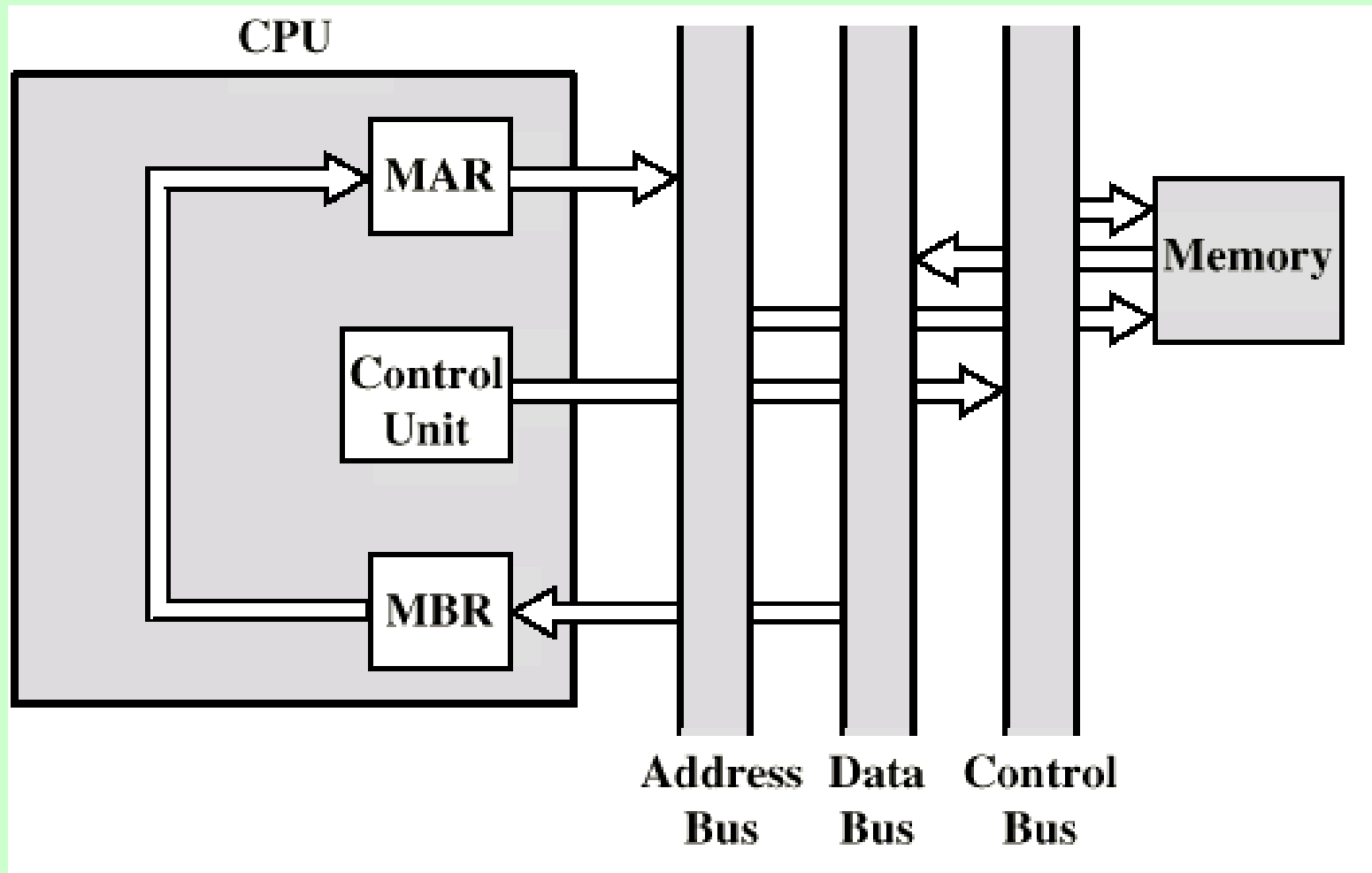
Data Flow (Data Fetch)

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

Data Flow (Fetch Diagram)



Data Flow (Indirect Diagram)



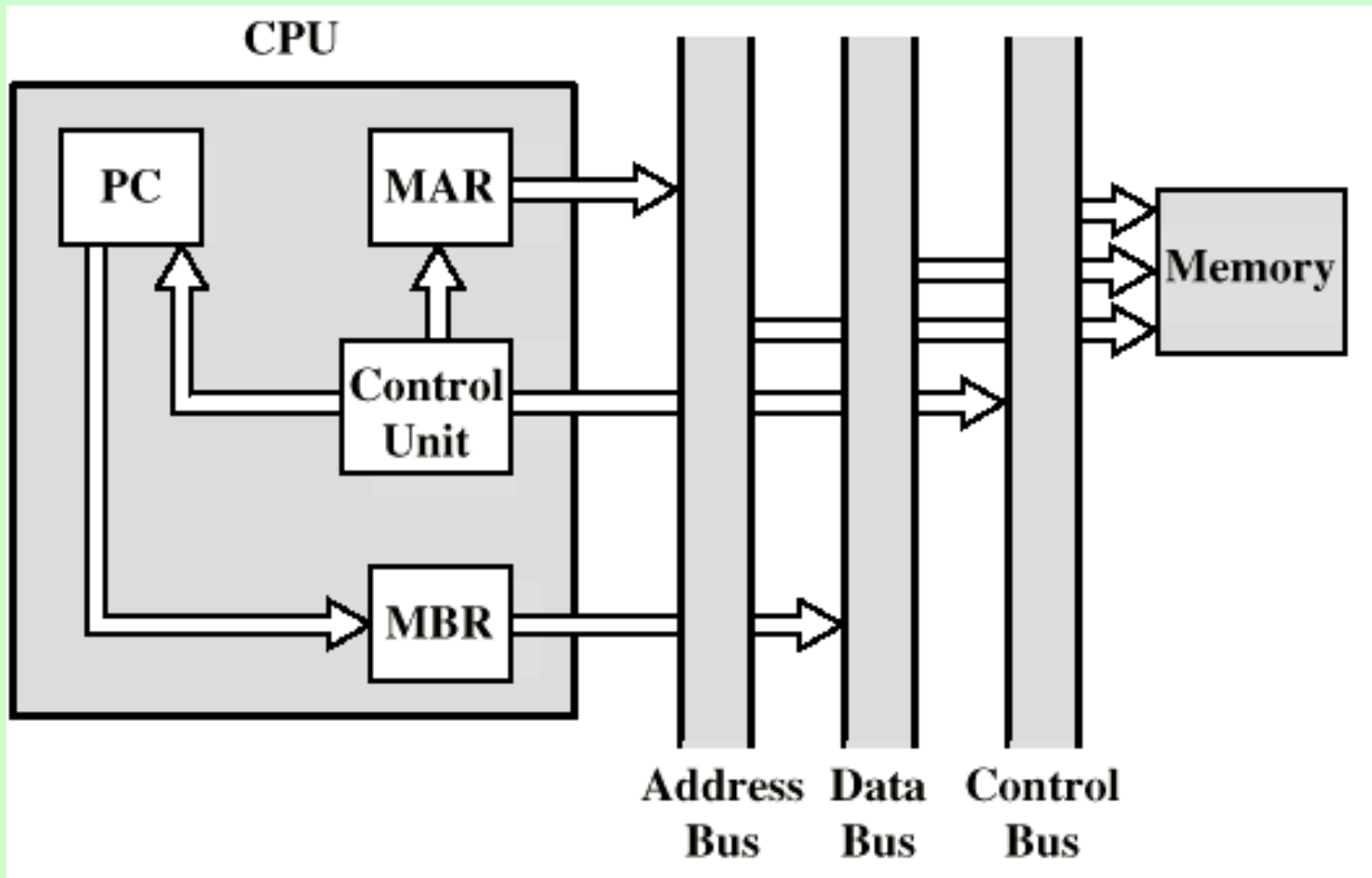
Data Flow (Execute)

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

Data Flow (Interrupt)

- Simple
- Predictable
- 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)



Prefetch

- Fetch accessing main memory
- Execution usually does not access main memory
- Can fetch next instruction during execution of current instruction
- Called instruction prefetch

Improved Performance

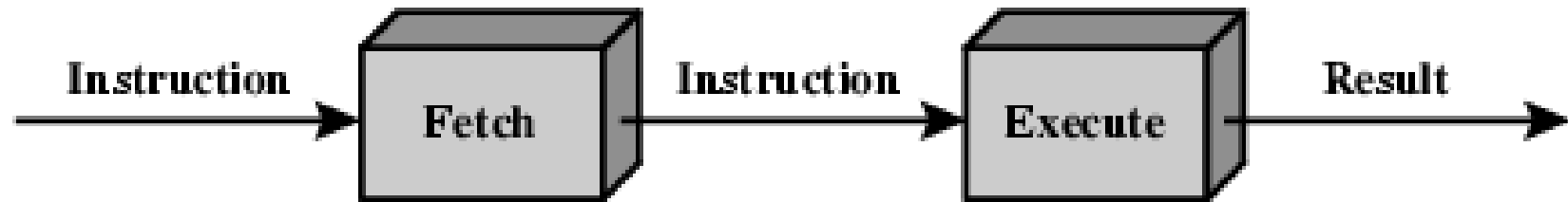
- But not doubled:
 - Fetch usually shorter than execution
 - Prefetch more than one instruction?
 - Any jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performance

Pipelining

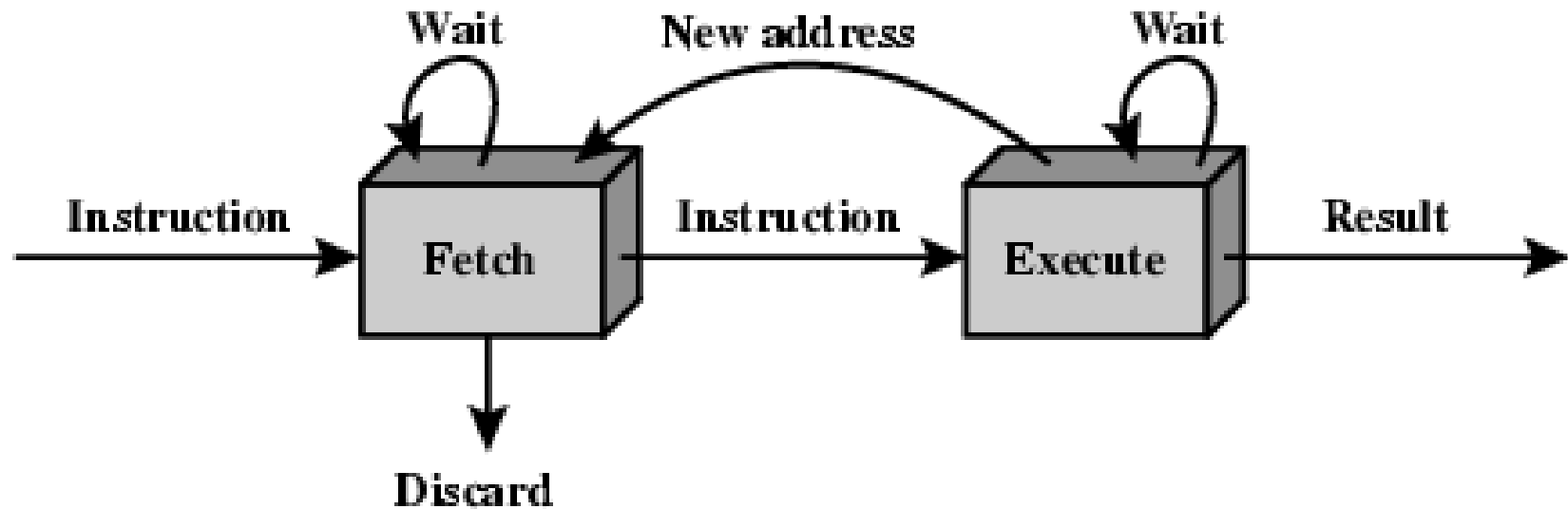
- Fetch instruction
- Decode instruction
- Calculate operands (i.e. EAs)
- Fetch operands
- Execute instructions
- Write result

- Overlap these operations

Two Stage Instruction Pipeline



(a) Simplified view



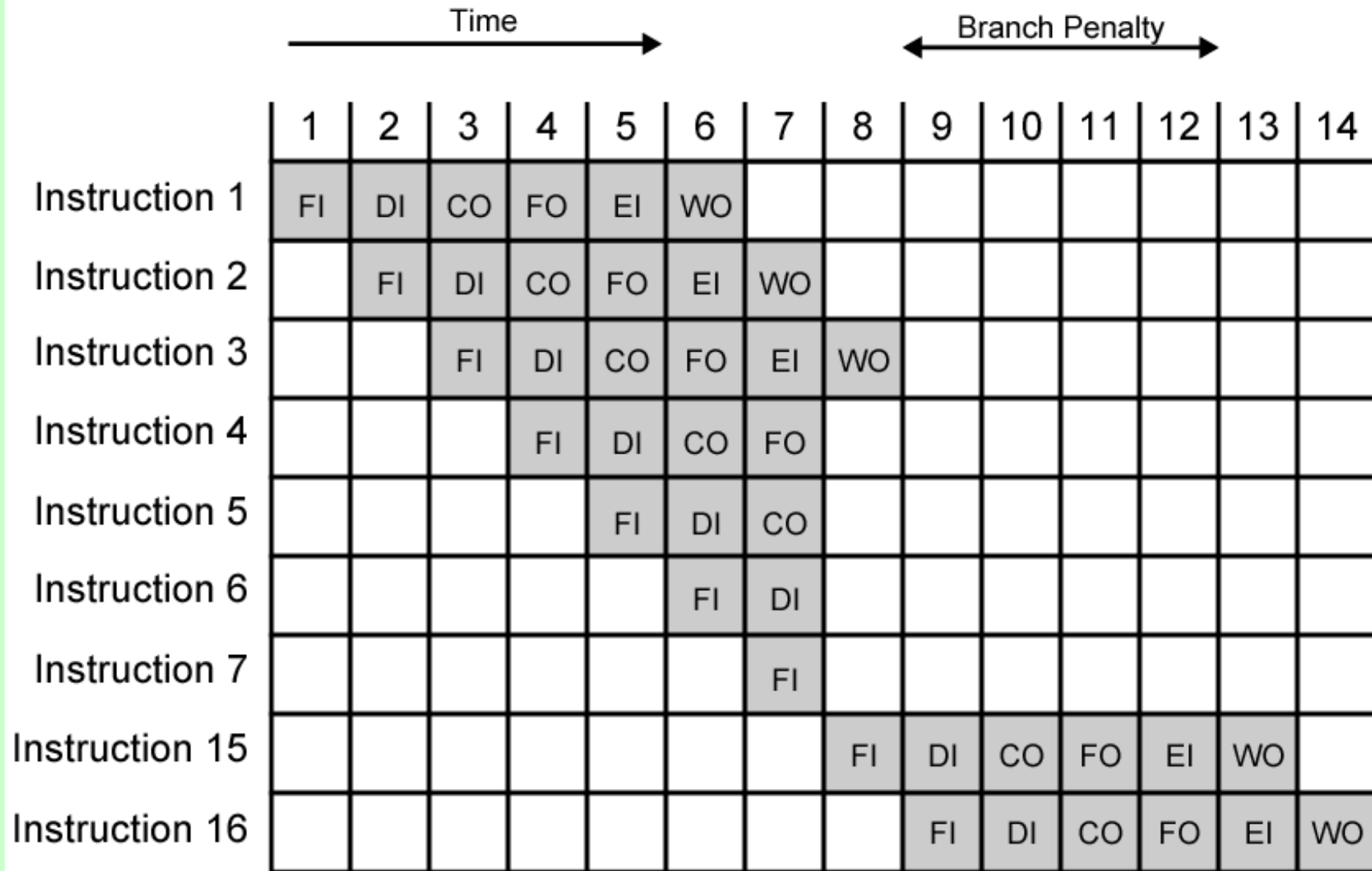
(b) Expanded view

Timing Diagram for Instruction Pipeline Operation

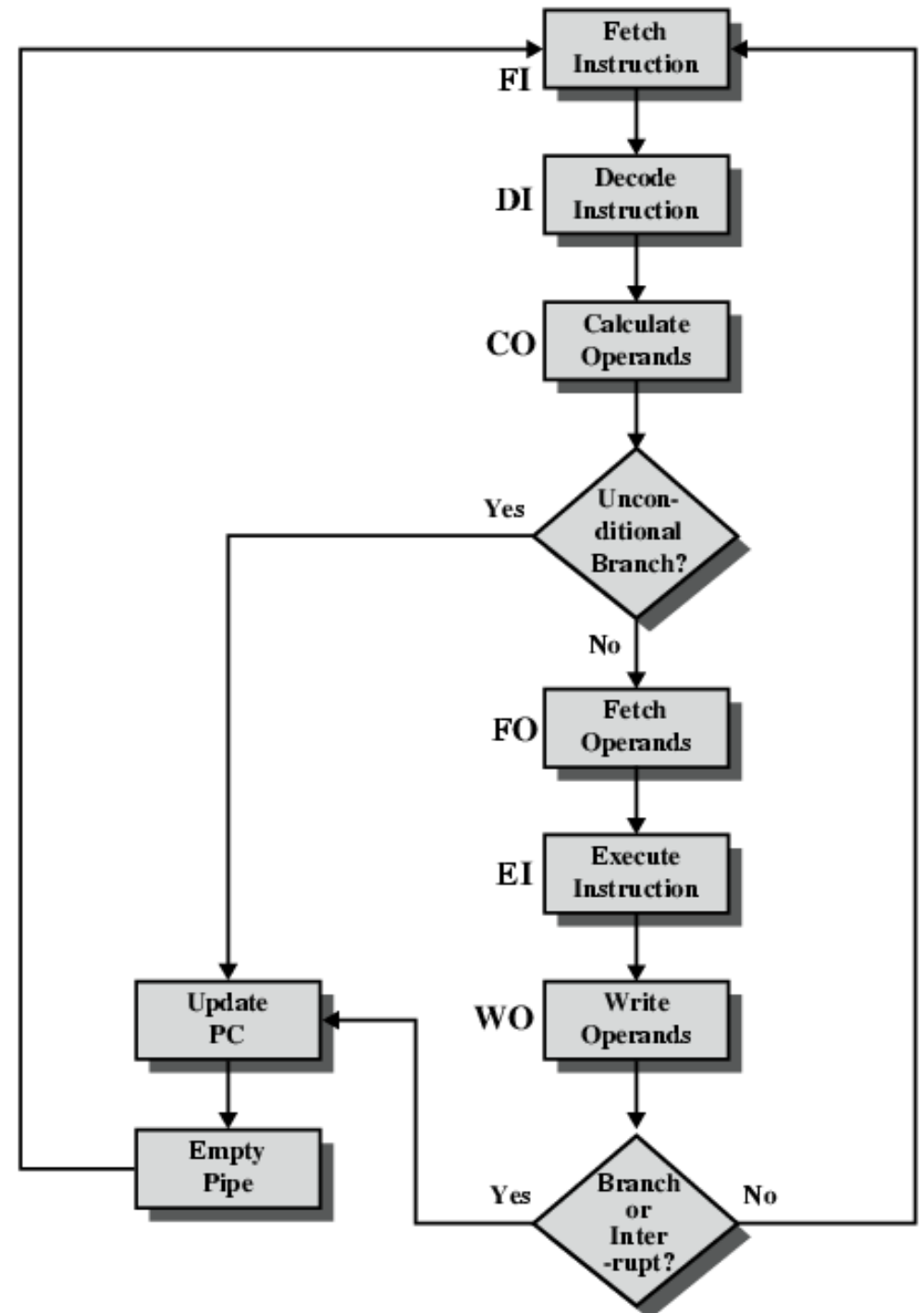
Time →

	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



Six Stage Instruction Pipeline



Alternative Pipeline Depiction

Time
↓

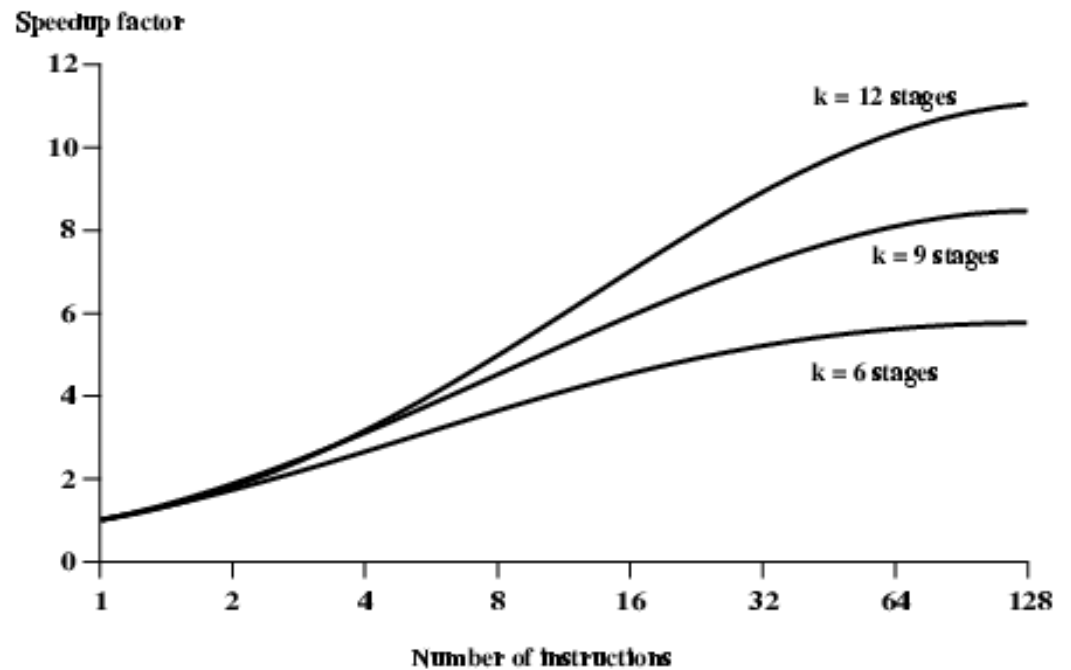
	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches

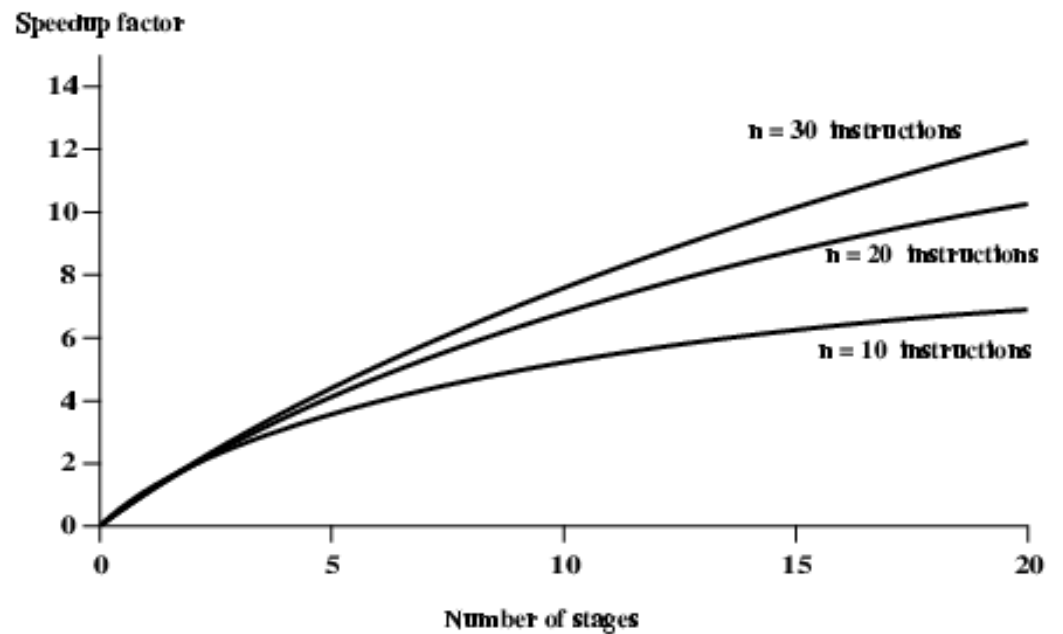
	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

Speedup Factors with Instruction Pipelining



(a)



(b)

Pipeline Hazards

- Pipeline, or some portion of pipeline, must stall
- Also called *pipeline bubble*
- Types of hazards
 - Resource
 - Data
 - Control

Resource Hazards

- Two (or more) instructions in pipeline need same resource
- Executed in serial rather than parallel for part of pipeline
- Also called *structural hazard*
- E.g. Assume simplified five-stage pipeline
 - Each stage takes one clock cycle
- Ideal case is new instruction enters pipeline each clock cycle
- Assume main memory has single port
- Assume instruction fetches and data reads and writes performed one at a time
- Ignore the cache
- Operand read or write cannot be performed in parallel with instruction fetch
- Fetch instruction stage must idle for one cycle fetching I3

- E.g. multiple instructions ready to enter execute instruction phase
- Single ALU

- One solution: increase available resources
 - Multiple main memory ports
 - Multiple ALUs

Data Hazards

- Conflict in access of an operand location
- Two instructions to be executed in sequence
- Both access a particular memory or register operand
- If in strict sequence, no problem occurs
- If in a pipeline, operand value could be updated so as to produce different result from strict sequential execution
- E.g. x86 machine instruction sequence:
 - `ADD EAX, EBX` /* $EAX = EAX + EBX$
 - `SUB ECX, EAX` /* $ECX = ECX - EAX$
- ADD instruction does not update EAX until end of stage 5, at clock cycle 5
- SUB instruction needs value at beginning of its stage 2, at clock cycle 4
- Pipeline must stall for two clocks cycles
- Without special hardware and specific avoidance algorithms, results in inefficient pipeline usage

Data Hazard Diagram

		Clock cycle									
		1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX		FI	DI	FO	EI	WO					
SUB ECX, EAX			FI	DI	Idle		FO	EI	WO		
	13			FI			DI	FO	EI	WO	
	14						FI	DI	FO	EI	WO

Types of Data Hazard

- Read after write (RAW), or true dependency
 - An instruction modifies a register or memory location
 - Succeeding instruction reads data in that location
 - Hazard if read takes place before write complete
- Write after read (WAR), or antidependency
 - An instruction reads a register or memory location
 - Succeeding instruction writes to location
 - Hazard if write completes before read takes place
- Write after write (WAW), or output dependency
 - Two instructions both write to same location
 - Hazard if writes take place in reverse of order intended sequence
- Previous example is RAW hazard
- See also Chapter 14

Resource Hazard Diagram

		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucción	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			FI	DI	FO	EI	WO		
	I4				FI	DI	FO	EI	WO	

(a) Five-stage pipeline, ideal case

		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucción	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			Idle	FI	DI	FO	EI	WO	
	I4					FI	DI	FO	EI	WO

(b) I1 source operand in memory

Control Hazard

Control Hazard

- Also known as *branch hazard*
- Pipeline makes wrong decision on branch prediction
- Brings instructions into pipeline that must subsequently be discarded
- 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

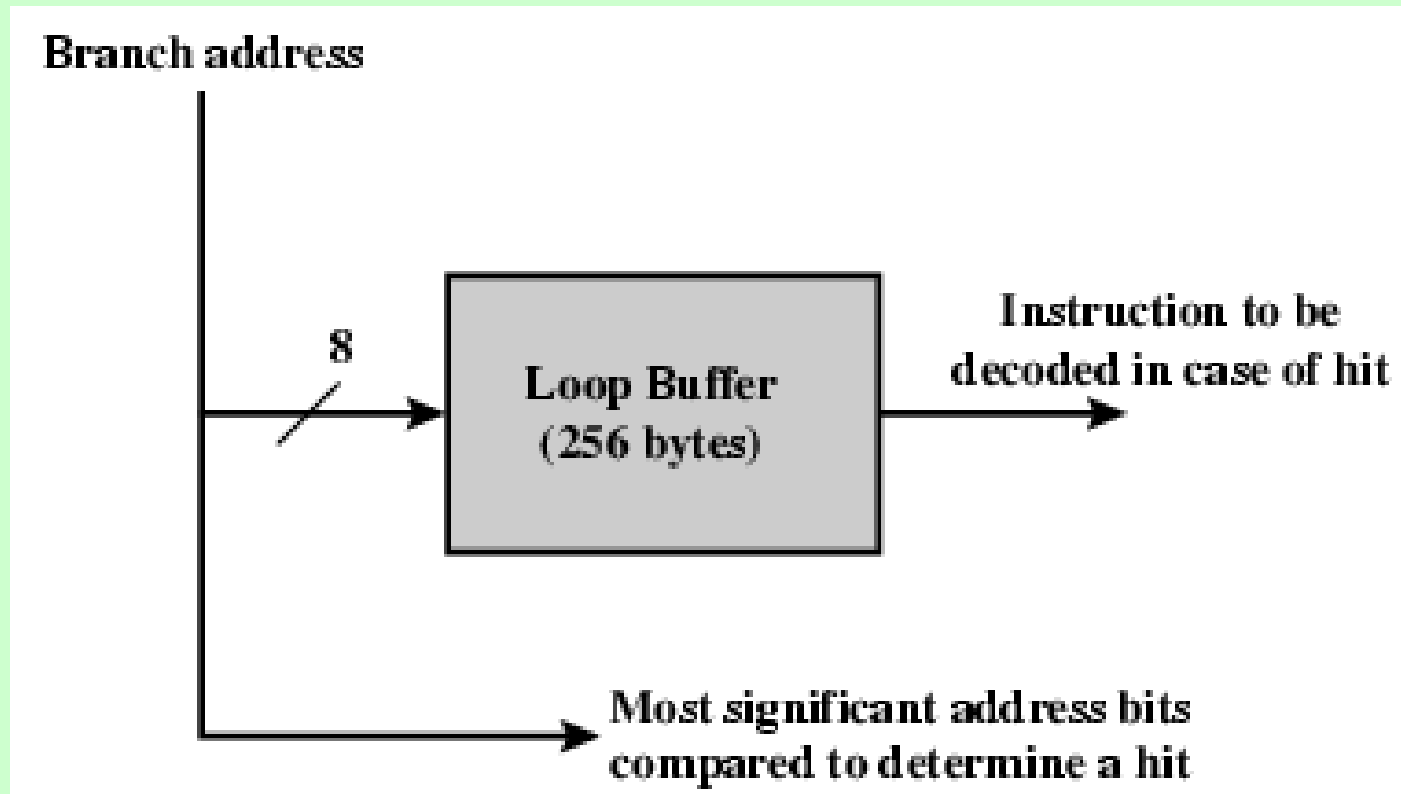
Prefetch Branch Target

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

Loop Buffer

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- c.f. cache
- Used by CRAY-1

Loop Buffer Diagram



Branch Prediction (1)

- Predict never taken
 - Assume that jump will not happen
 - Always fetch next instruction
 - 68020 & VAX 11/780
 - VAX will not prefetch after branch if a page fault would result (O/S v CPU design)
- Predict always taken
 - Assume that jump will happen
 - Always fetch target instruction

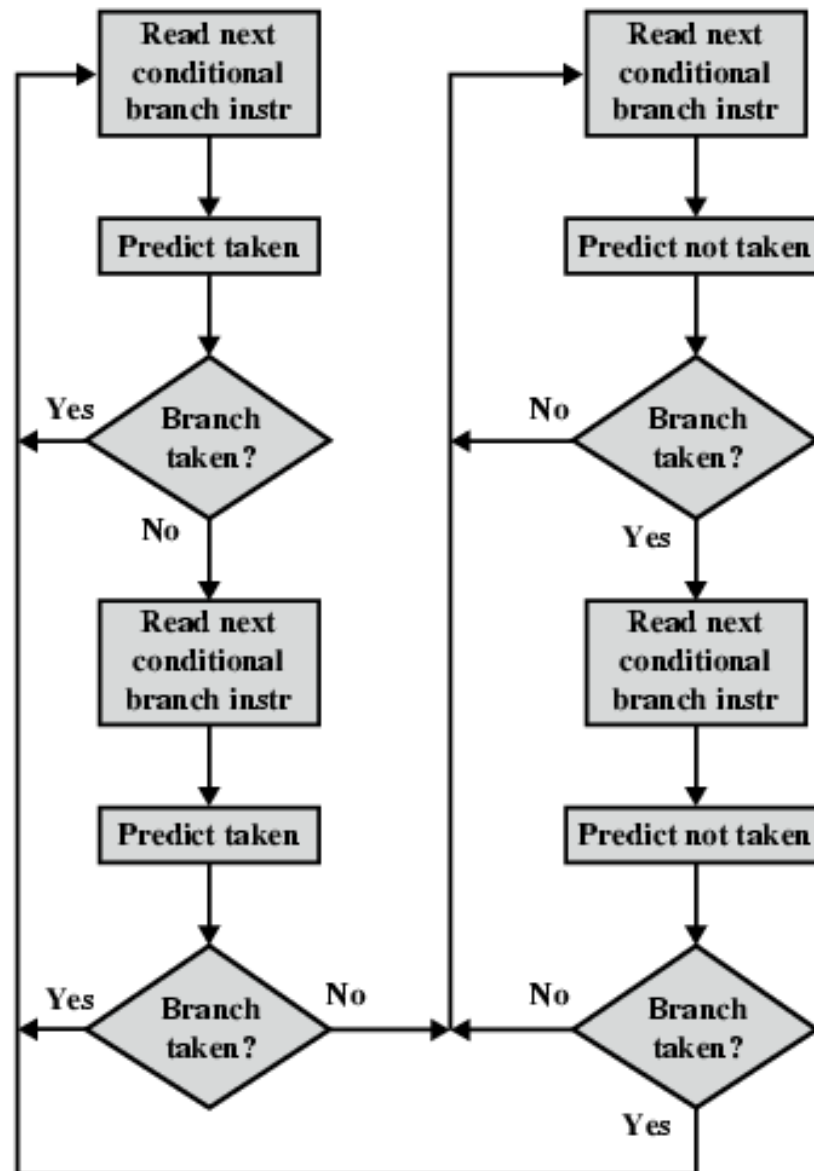
Branch Prediction (2)

- Predict by Opcode
 - Some instructions are more likely to result in a jump than others
 - Can get up to 75% success
- Taken/Not taken switch
 - Based on previous history
 - Good for loops
 - Refined by two-level or correlation-based branch history
- Correlation-based
 - In loop-closing branches, history is good predictor
 - In more complex structures, branch direction correlates with that of related branches
 - Use recent branch history as well

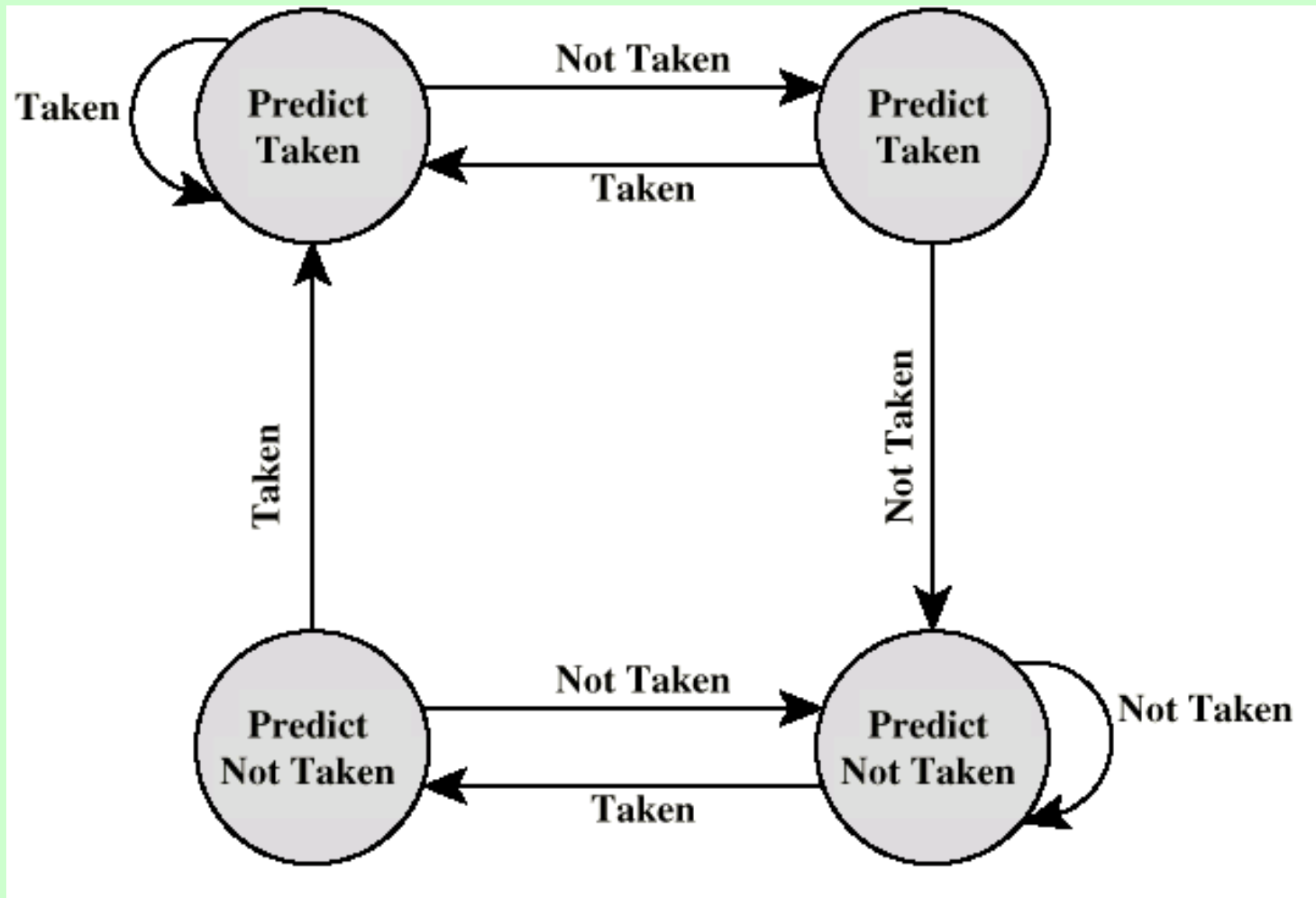
Branch Prediction (3)

- Delayed Branch
 - Do not take jump until you have to
 - Rearrange instructions

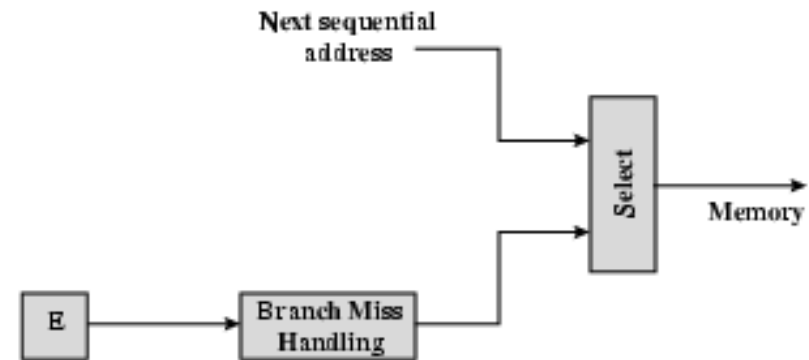
Branch Prediction Flowchart



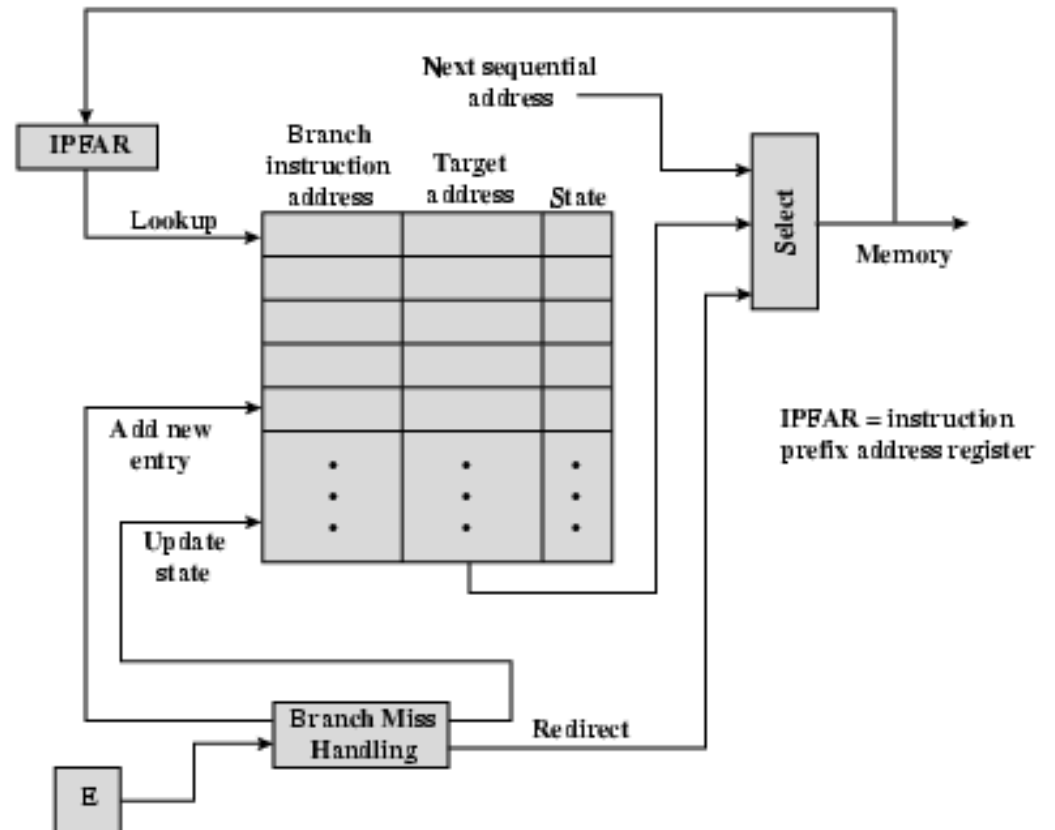
Branch Prediction State Diagram



Dealing With Branches



(a) Predict never taken strategy

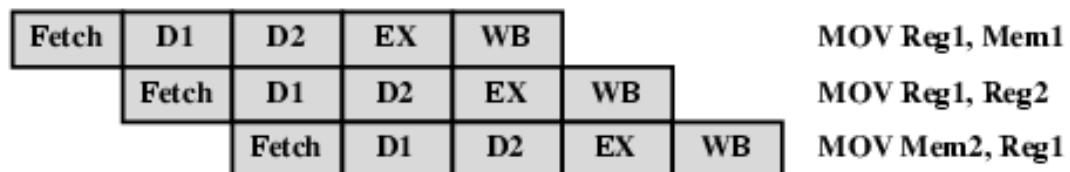


(b) Branch history table strategy

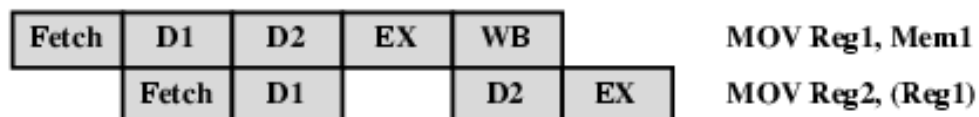
Intel 80486 Pipelining

- Fetch
 - From cache or external memory
 - Put in one of two 16-byte prefetch buffers
 - Fill buffer with new data as soon as old data consumed
 - Average 5 instructions fetched per load
 - Independent of other stages to keep buffers full
- Decode stage 1
 - Opcode & address-mode info
 - At most first 3 bytes of instruction
 - Can direct D2 stage to get rest of instruction
- Decode stage 2
 - Expand opcode into control signals
 - Computation of complex address modes
- Execute
 - ALU operations, cache access, register update
- Writeback
 - Update registers & flags
 - Results sent to cache & bus interface write buffers

80486 Instruction Pipeline Examples



(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay



(c) Branch Instruction Timing

Pentium 4 Registers

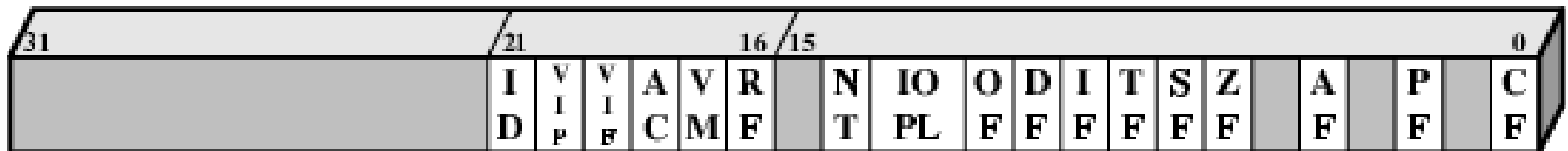
(a) Integer Unit

Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
Flags	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

(b) Floating-Point Unit

Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
Control	1	16	Control bits
Status	1	16	Status bits
Tag Word	1	16	Specifies contents of numeric registers
Instruction Pointer	1	+8	Points to instruction interrupted by exception
Data Pointer	1	+8	Points to operand interrupted by exception

EFLAGS Register



ID = Identification flag

VIP = Virtual interrupt pending

VIF = Virtual interrupt flag

AC = Alignment check

VM = Virtual 8086 mode

RF = Resume flag

NT = Nested task flag

IOPL = I/O privilege level

OF = Overflow flag

DF = Direction flag

IF = Interrupt enable flag

TF = Trap flag

SF = Sign flag

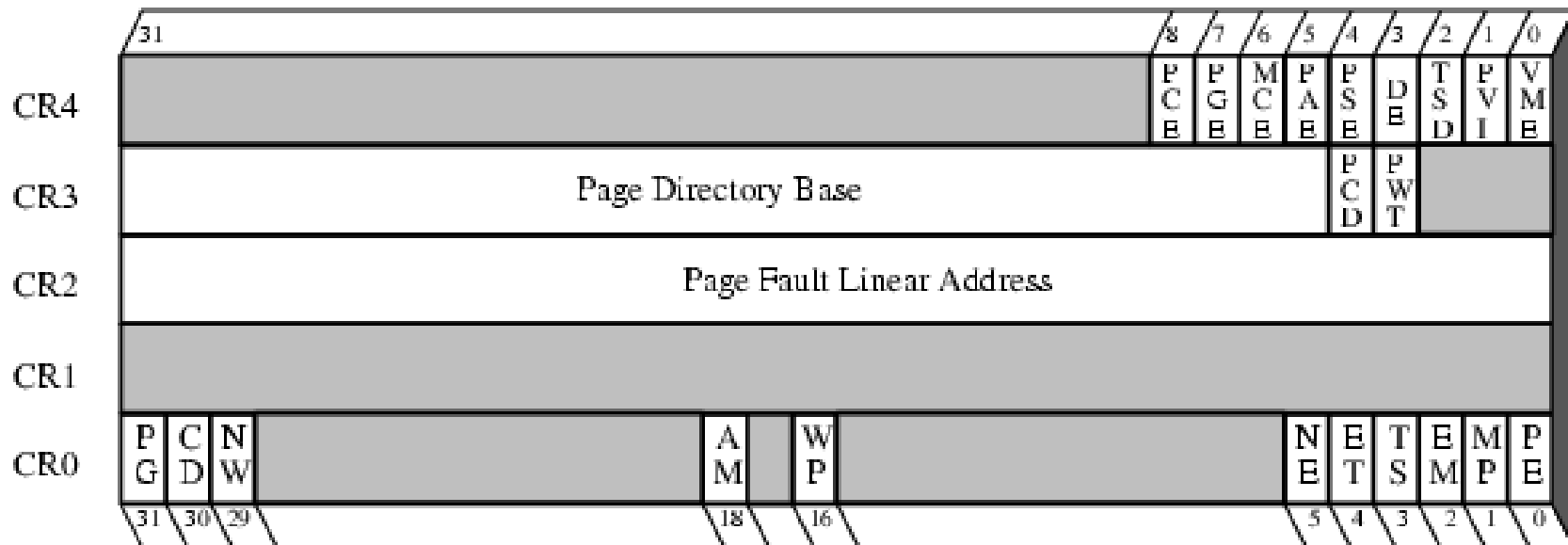
ZF = Zero flag

AF = Auxiliary carry flag

PF = Parity flag

CF = Carry flag

Control Registers



PCE = Performance Counter Enable

PGE = Page Global Enable

MCE = Machine Check Enable

PAE = Physical Address Extension

PSE = Page Size Extensions

DE = Debug Extensions

TSD = Time Stamp Disable

PVI = Protected Mode Virtual Interrupt

VME = Virtual 8086 Mode Extensions

PCD = Page-level Cache Disable

PWT = Page-level Writes Transparent

PG = Paging

CD = Cache Disable

NW = Not Write Through

AM = Alignment Mask

WP = Write Protect

NE = Numeric Error

ET = Extension Type

TS = Task Switched

EM = Emulation

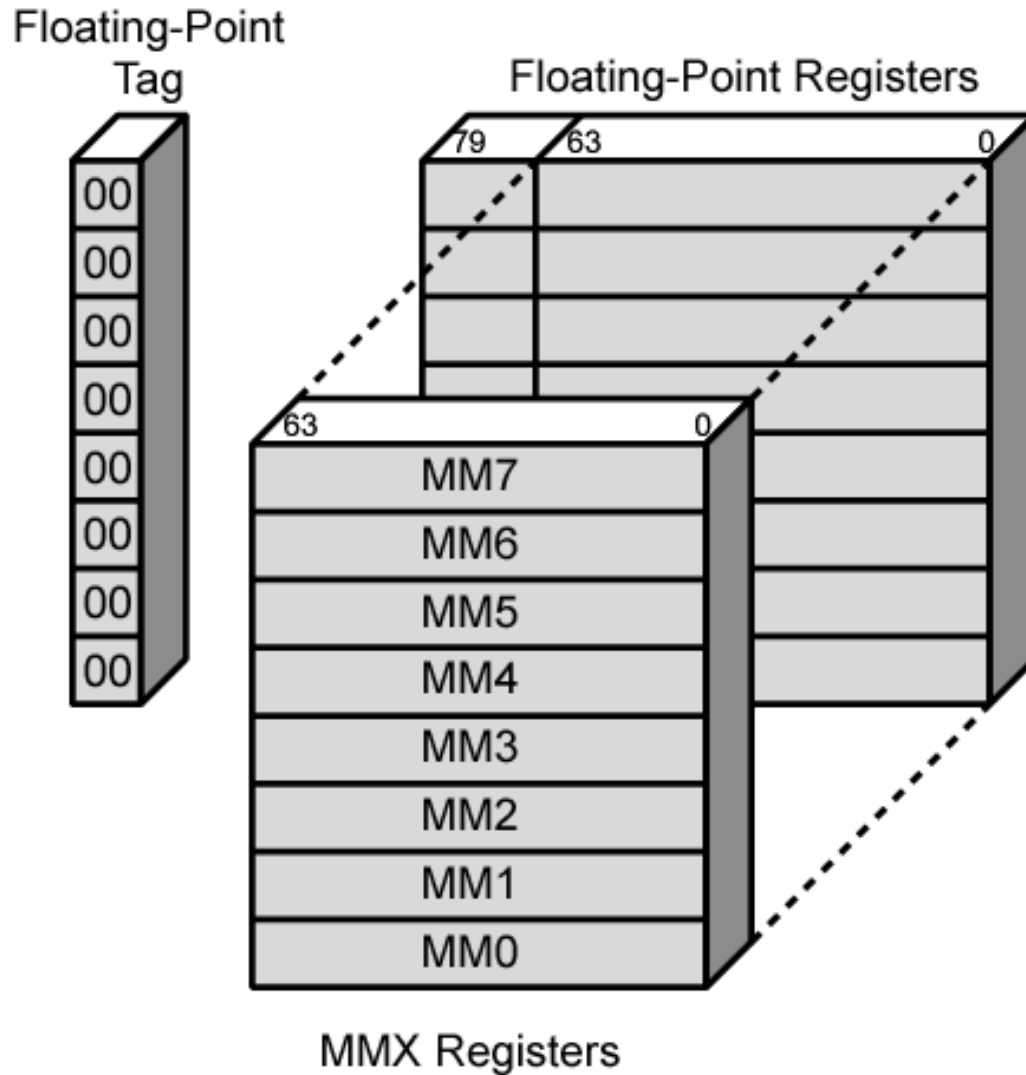
MP = Monitor Coprocessor

PE = Protection Enable

MMX Register Mapping

- MMX uses several 64 bit data types
- Use 3 bit register address fields
 - 8 registers
- No MMX specific registers
 - Aliasing to lower 64 bits of existing floating point registers

Mapping of MMX Registers to Floating-Point Registers



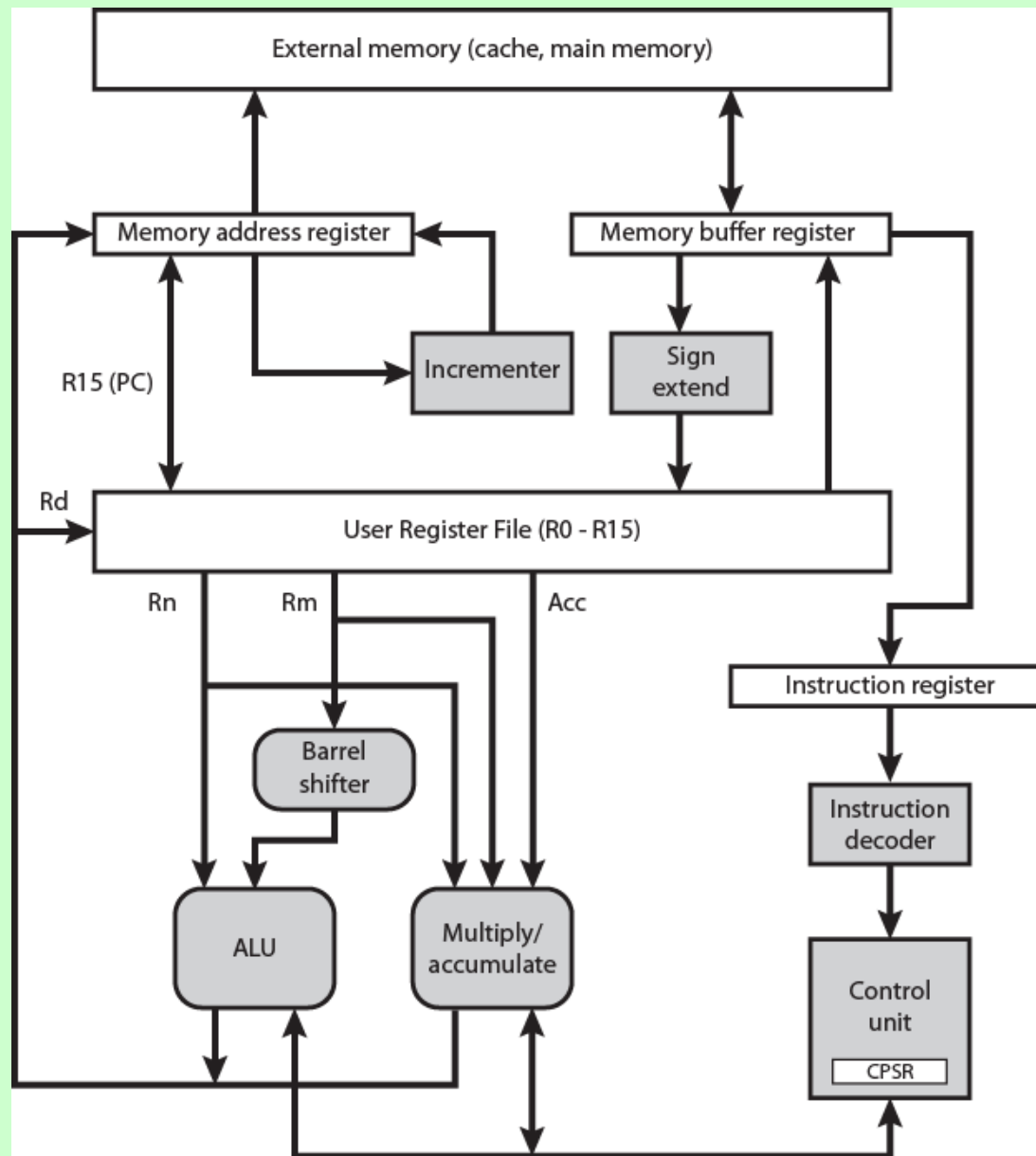
Pentium Interrupt Processing

- Interrupts
 - Maskable
 - Nonmaskable
- Exceptions
 - Processor detected
 - Programmed
- Interrupt vector table
 - Each interrupt type assigned a number
 - Index to vector table
 - 256 * 32 bit interrupt vectors
- 5 priority classes

ARM Attributes

- RISC
- Moderate array of uniform registers
 - More than most CISC, less than many RISC
- Load/store model
 - Operations perform on operands in registers only
- Uniform fixed-length instruction
 - 32 bits standard set 16 bits Thumb
- Shift or rotation can preprocess source registers
 - Separate ALU and shifter units
- Small number of addressing modes
 - All load/store addressees from registers and instruction fields
 - No indirect or indexed addressing involving values in memory
- Auto-increment and auto-decrement addressing
 - Improve loops
- Conditional execution of instructions minimizes conditional branches

Simplified ARM Organization



ARM Processor Organization

- Many variations depending on ARM version
- Data exchanged between processor and memory through data bus
- Data item (load/store) or instruction (fetch)
- Instructions go through decoder before execution
- Pipeline and control signal generation in control unit
- Data goes to register file
 - Set of 32 bit registers
 - Byte & halfword twos complement data sign extended
- Typically two source and one result register
- Rotation or shift before ALU

ARM Processor Modes

- User
- Privileged
 - 6 modes
 - OS can tailor systems software use
 - Some registers dedicated to each privileged mode
 - Swifter context changes
- Exception
 - 5 of privileged modes
 - Entered on given exceptions
 - Substitute some registers for user registers
 - Avoid corruption

Privileged Modes

- System Mode
 - Not exception
 - Uses same registers as User mode
 - Can be interrupted by...
- Supervisor mode
 - OS
 - Software interrupt used to invoke operating system services
- Abort mode
 - memory faults
- Undefined mode
 - Attempt instruction that is not supported by integer core coprocessors
- Fast interrupt mode
 - Interrupt signal from designated fast interrupt source
 - Fast interrupt cannot be interrupted
 - May interrupt normal interrupt
- Interrupt mode
- Interrupt signal from any other interrupt source

ARM Register Organization Table

	Modes					
		Privileged modes				
		Exception modes				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13 (SP)	R13 (SP)	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14 (LR)	R14 (LR)	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM Register Organization

- 37 x 32-bit registers
- 31 general-purpose registers
 - Some have special purposes
 - E.g. program counters
- Six program status registers
- Registers in partially overlapping banks
 - Processor mode determines bank
- 16 numbered registers and one or two program status registers visible

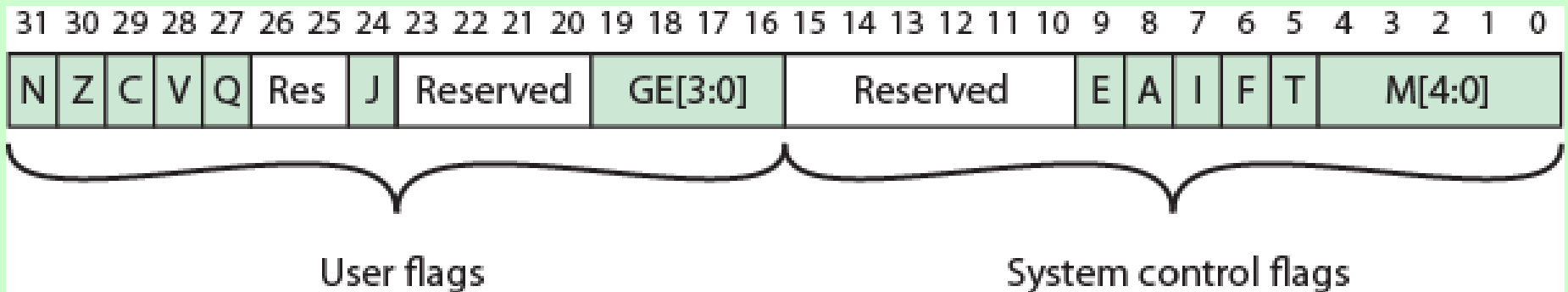
General Register Usage

- R13 normally stack pointer (SP)
 - Each exception mode has its own R13
- R14 link register (LR)
 - Subroutine and exception mode return address
- R15 program counter

CPSR

- CPSR process status register
 - Exception modes have dedicated SPSR
- 16 msb are user flags
 - Condition codes (N,Z,C,V)
 - Q – overflow or saturation in some SMID instructions
 - J – Jazelle (8 bit) instructions
 - GEE[3:0] SMID use [19:16] as greater than or equal flag
- 16 lsb system flags for privilege modes
 - E – endian
 - Interrupt disable
 - T – Normal or Thumb instruction
 - Mode

ARM CPSR and SPSR



ARM Interrupt (Exception) Processing

- More than one exception allowed
- Seven types
- Execution forced from exception vectors
- Multiple exceptions handled in priority order
- Processor halts execution after current instruction
- Processor state preserved in SPSR for exception
 - Address of instruction about to execute put in link register
 - Return by moving SPSR to CPSR and R14 to PC

Foreground Reading

- Processor examples
- Stallings Chapter 12
- Manufacturer web sites & specs