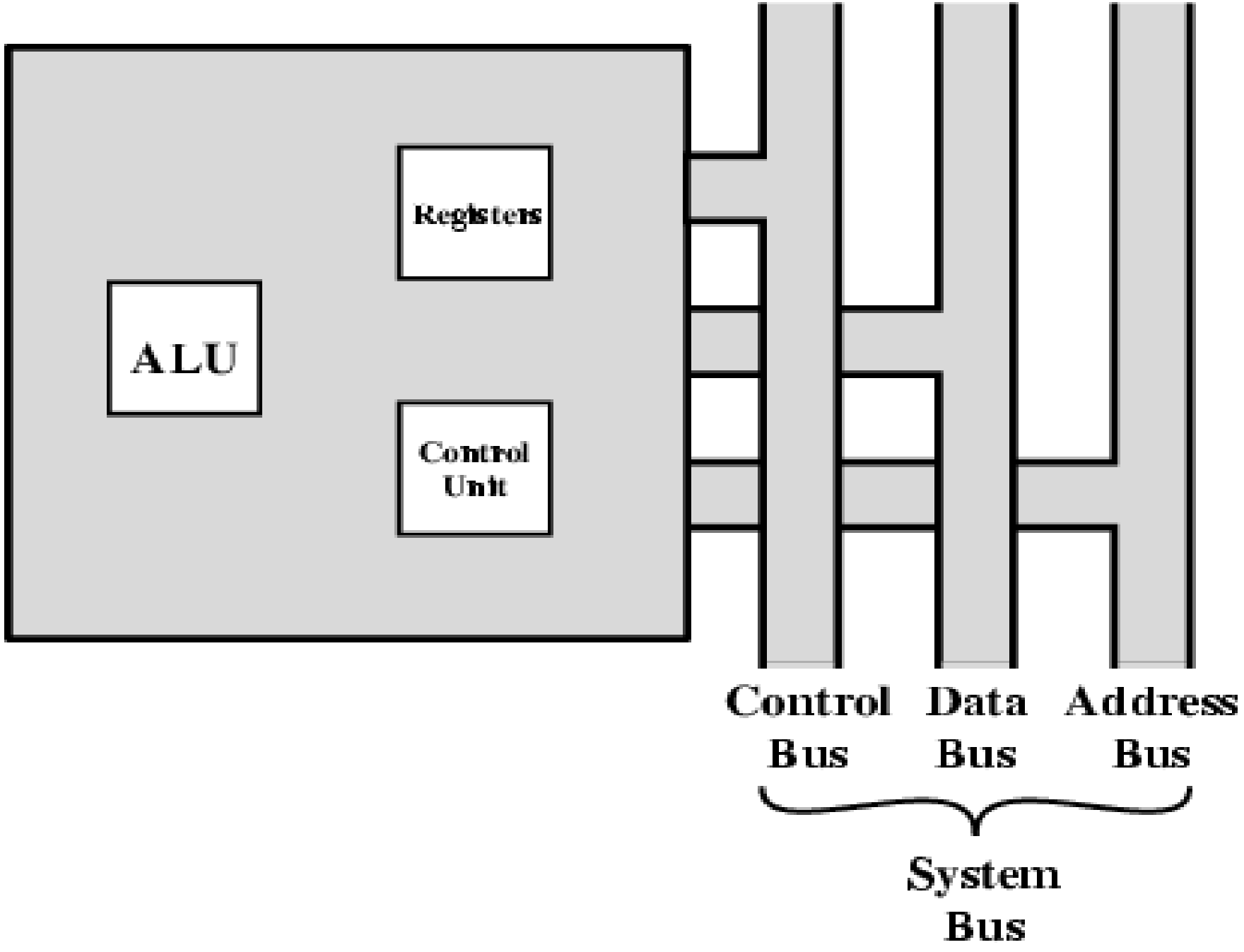
CPU Structure and Function

Chapter 12,
William Stallings
Computer Organization and Architecture
7th Edition

CPU Function

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

CPU With Systems Bus



Registers

- CPU must have some working space (temporary storage) - 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 (AC)
- Addressing: segment (cf. virtual memory), stack (points to top of stack, cf. implicit addressing)

General Purpose Registers (2)

- Make them general purpose
  - Increased flexibility and programmer options
  - Increased instruction size & complexity, addressing
- Make them specialized
  - Smaller (faster) but more instructions
  - Less flexibility, addresses implicit in opcode

How Many GP Registers?

- Between 8 - 32
- Less = more memory references
- More takes up processor real estate
- See also RISC

How big?

- Large enough to hold full address
- Large enough to hold full data types
- But often possible to combine two data registers or two address registers by using more complex addressing (e.g., page and offset)

Condition Code Registers – Flags

- Sets of individual bits, flags
  - e.g., result of last operation was zero
- Can be read by programs
  - e.g., Jump if zero – simplifies branch taking
- Can not (usually) be set by programs

Control & Status Registers

- Program Counter (PC)
- Instruction Register (IR)
- Memory Address Register (MAR) – connects to address bus
- Memory Buffer Register (MBR) – connects to data bus, feeds other registers

Program Status Word

- A set of bits
- Condition Codes:
  - Sign (of last result)
  - Zero (last result)
  - Carry (multiword arithmetic)
  - Equal (two latest results)
  - Overflow
- Interrupts enabled/disabled
- Supervisor/user mode

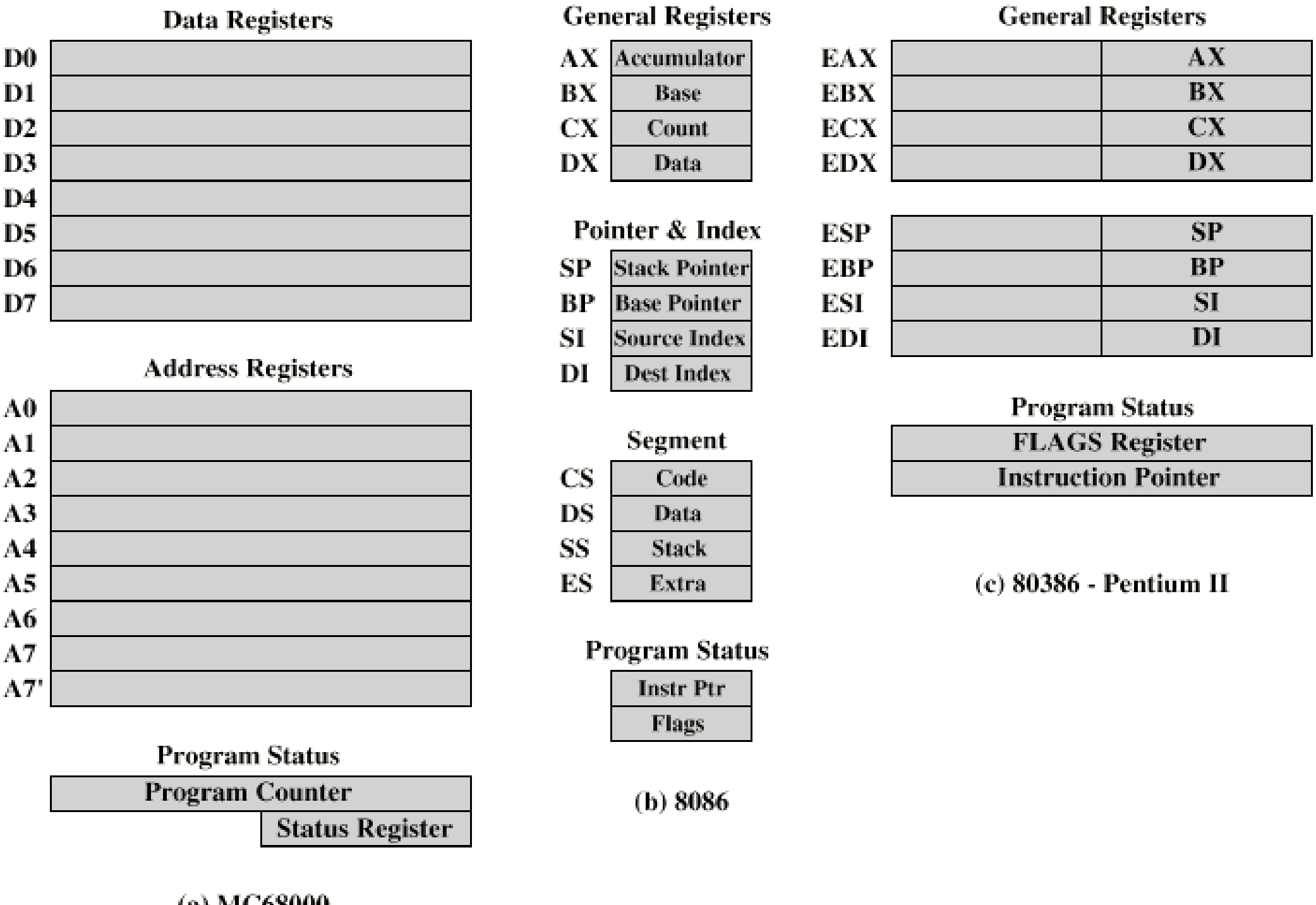
Supervisor Mode

- 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 OS)
  - Interrupt Vectors (see OS)

- N.B. CPU design and operating system design are closely linked


Data Registers
D0
D1
D2
D3
D4
D5
D6
D7
Address Registers
A0
A1
A2
A3
A4
A5
A6
A7
A7'
Program Status
Program Counter
Status Register
(a) MC68000
General Registers
AX
Accumulator
BX
Base
CX
Count
DX
Data
Pointer & Index
SP
Stack Pointer
BP
Base Pointer
SI
Source Index
DI
Dest Index
Segment
CS
Code
DS
Data
SS
Stack
ES
Extra
Program Status
Instr Ptr
Flags
(b) 8086
General Registers
EAX
AX
EBX
BX
ECX
CX
EDX
DX
ESP
SP
EBP
BP
ESI
SI
EDI
DI
Program Status
FLAGS Register
Instruction Pointer
(c) 80386 - Pentium II

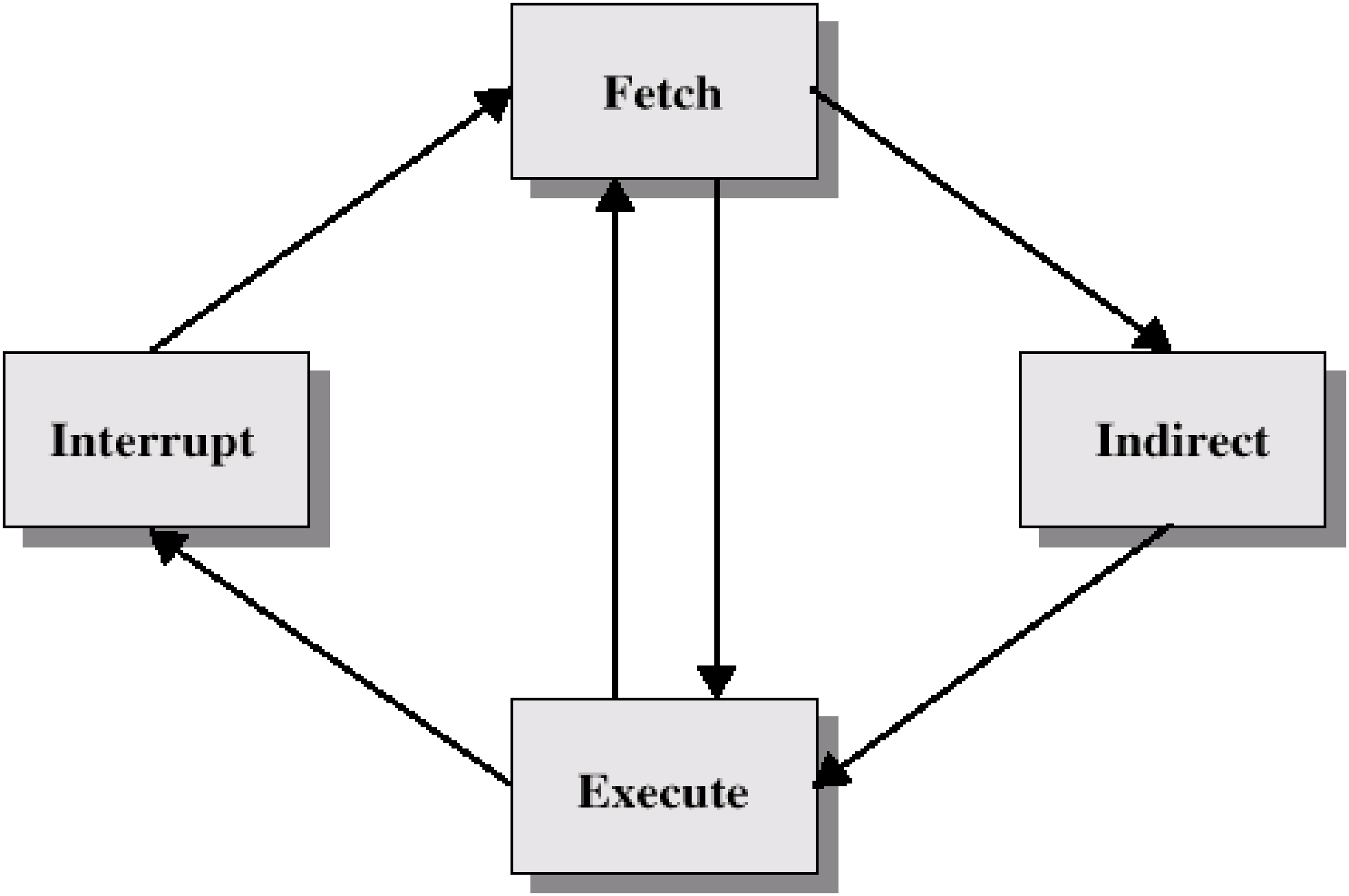
MC68000 and Intel registers

- Motorola:
  - Largely general purpose registers – explicit addressing
  - Data registers also for indexing
  - A7 and A7’ for user and kernel stacks
- Intel
  - Largely specific purpose registers – implicit addressing
  - Segment, Pointer & Index, Data/General purpose
  - Pentium II – backward compatibility

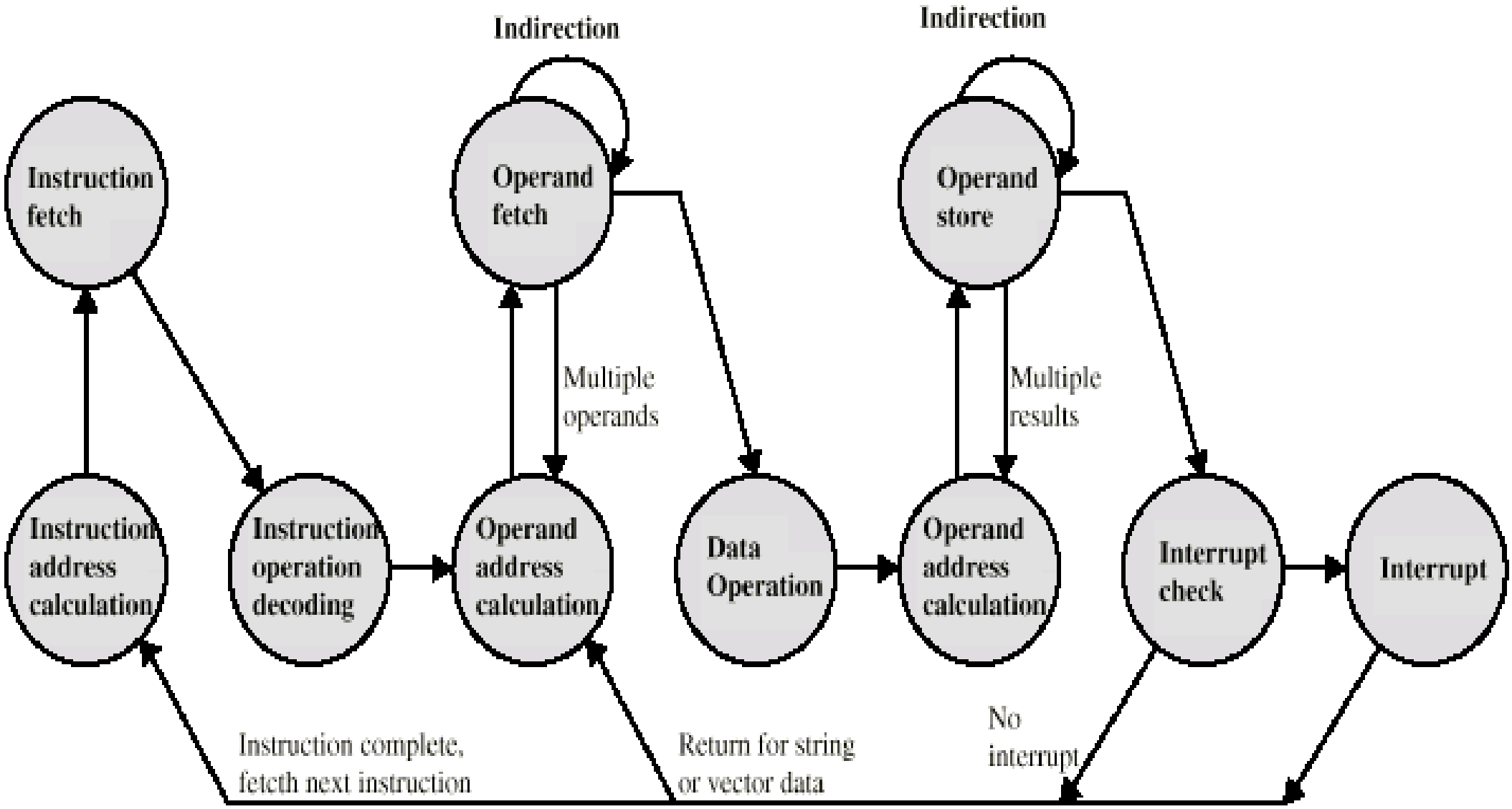
Indirect Cycle

- Same address can refer to different arguments (by changing the content of the location the address is pointing to)
- Indirect addressing requires more memory accesses to fetch operands
- Can be thought of as additional instruction subcycle

Instruction Cycle with Indirect



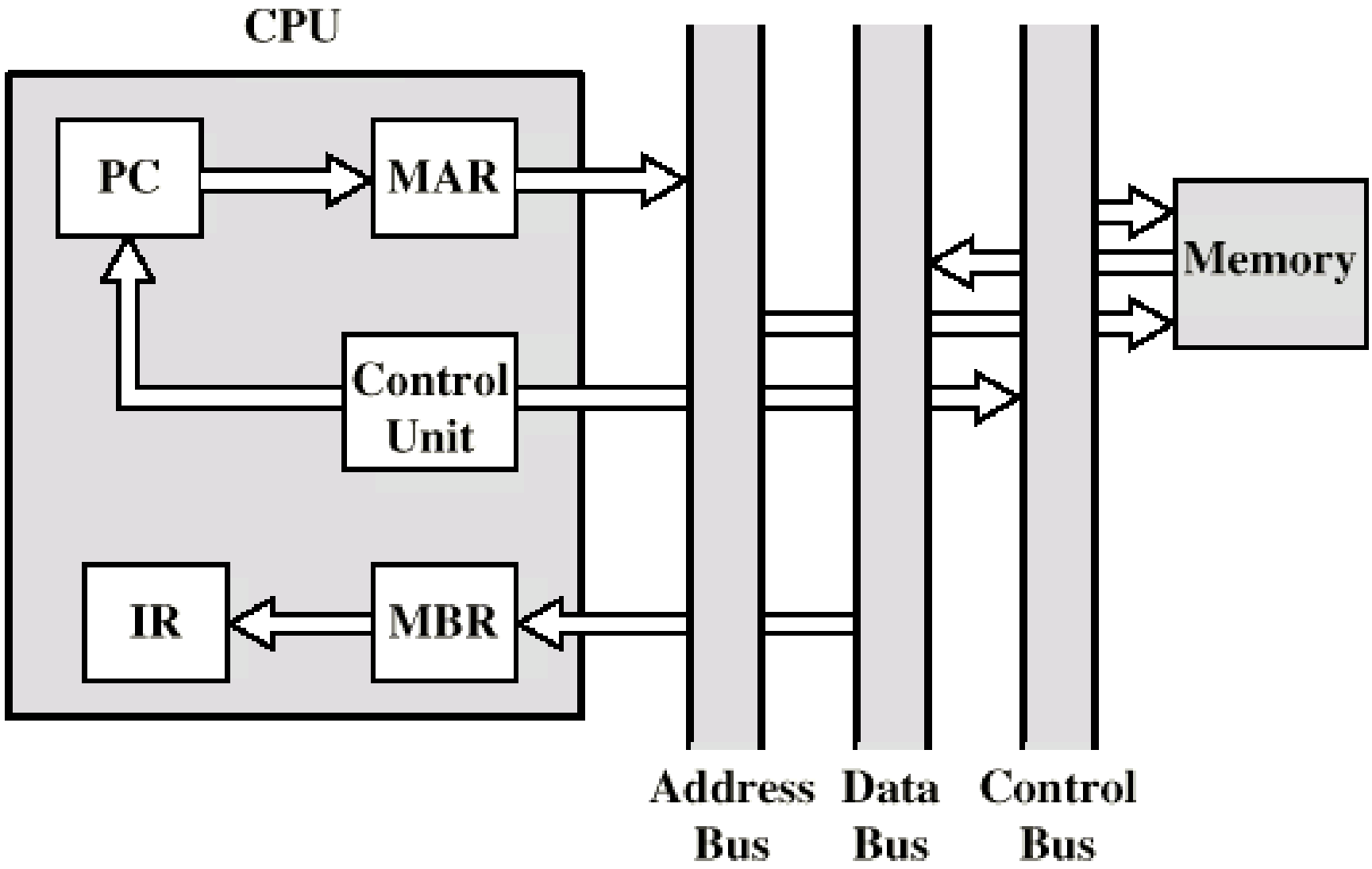
Instruction Cycle State Diagram



Data Flow (Instruction 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 (Fetch Diagram)

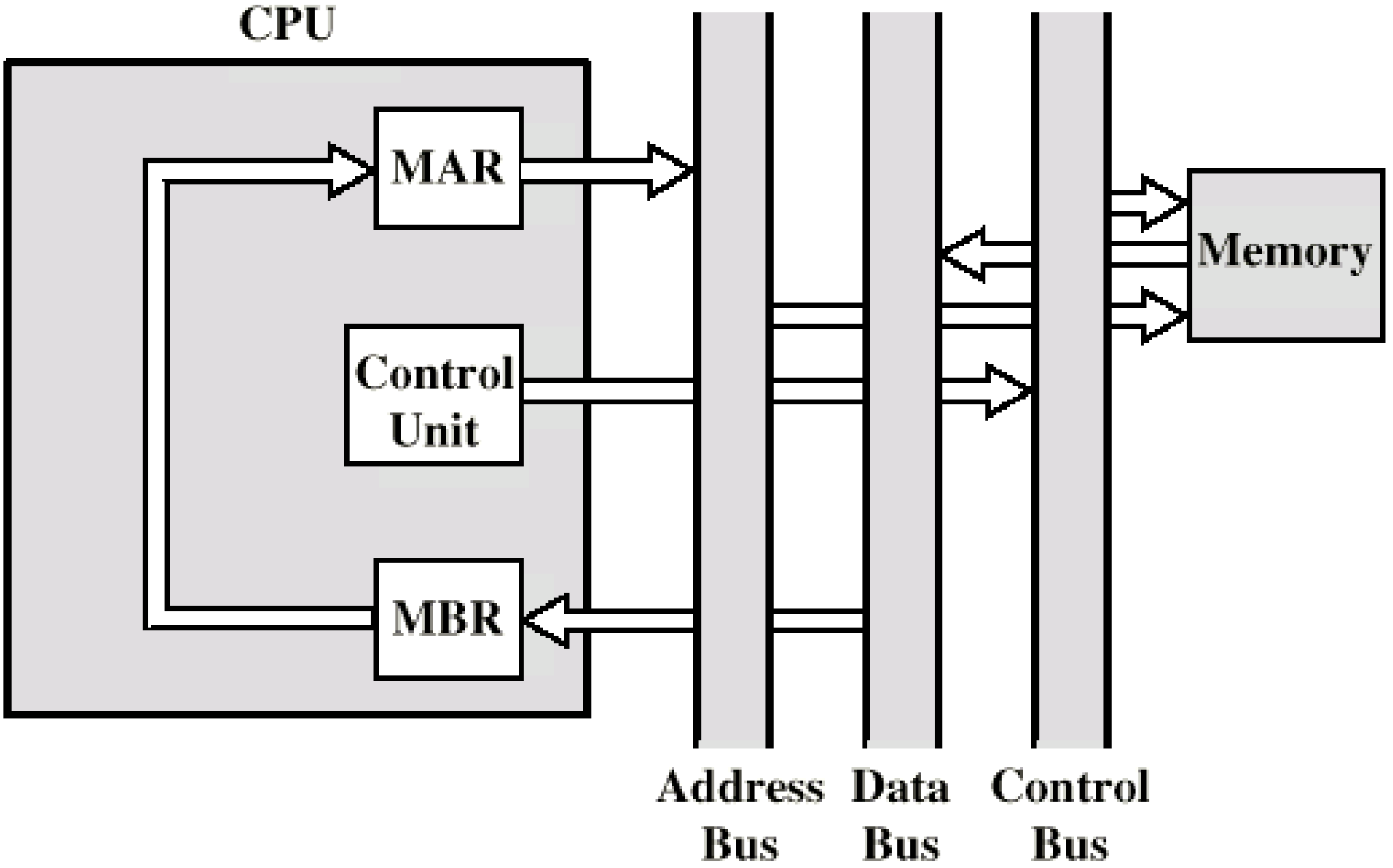


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Data Flow (Data Fetch)

- IR is examined
- If indirect addressing, indirect cycle is performed
  - Rightmost n bits of MBR (address part of instruction) transferred to MAR
  - Control unit requests memory read
  - Result (address of operand) moved to MBR

Data Flow (Indirect Diagram)



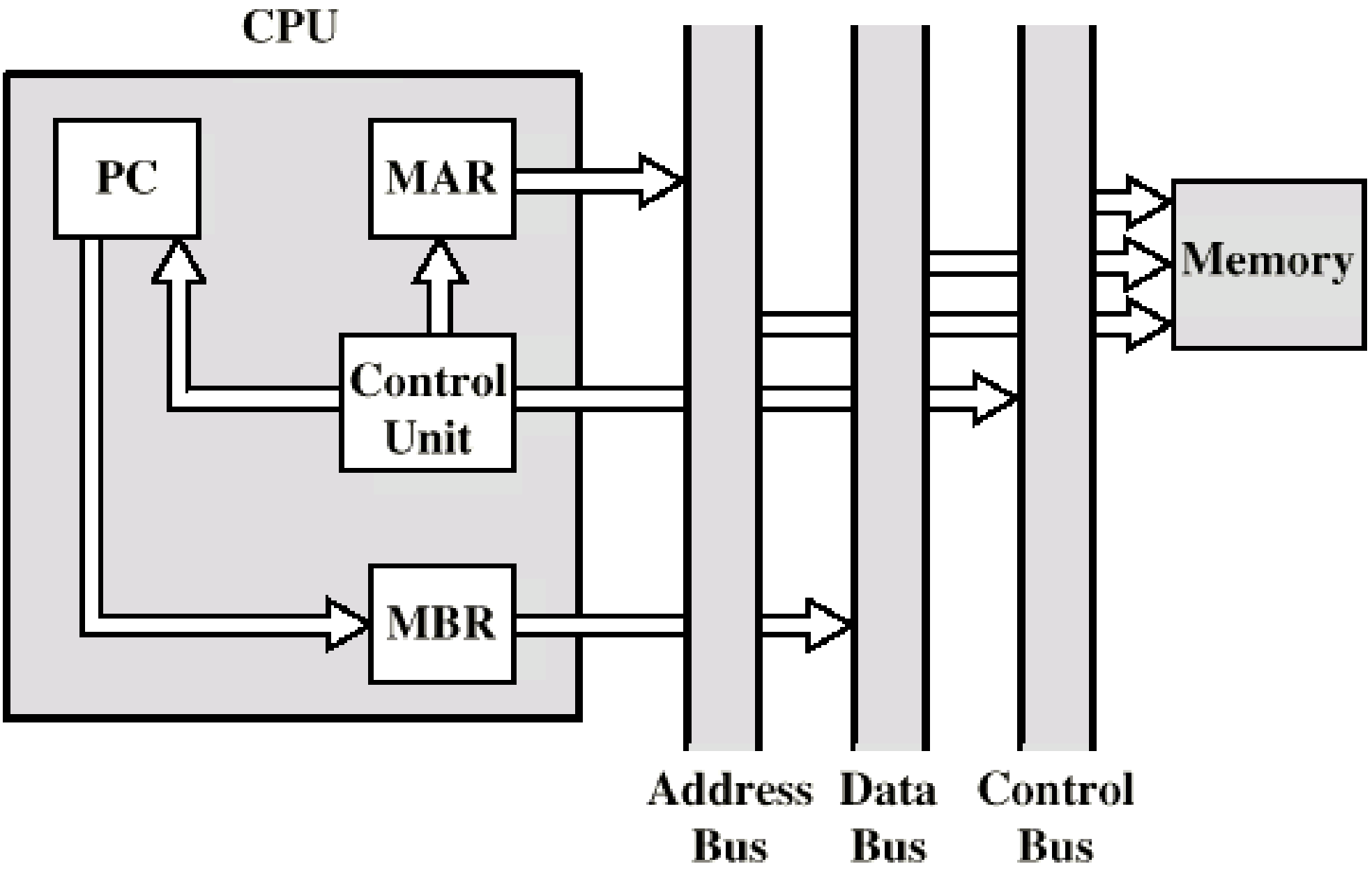
Data Flow (Execute)

- 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 according to content of MAR
- PC loaded with address of interrupt handling routine
- Next instruction (first of interrupt handler) can be fetched

Data Flow (Interrupt Diagram)



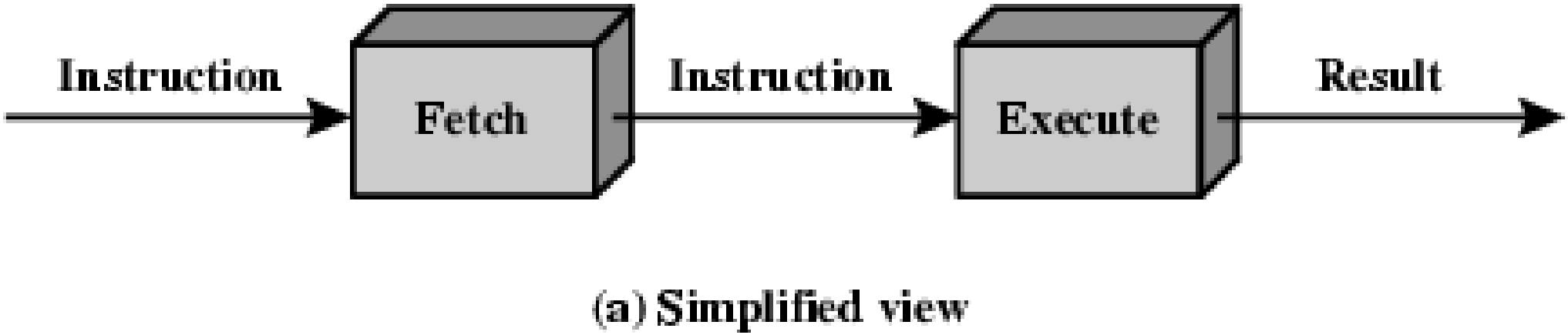
Prefetch

- Fetch involves accessing main memory
- Execution of ALU operations do not access main memory
- Can fetch next instruction during execution of current instruction, cf. assembly line
- Called instruction prefetch

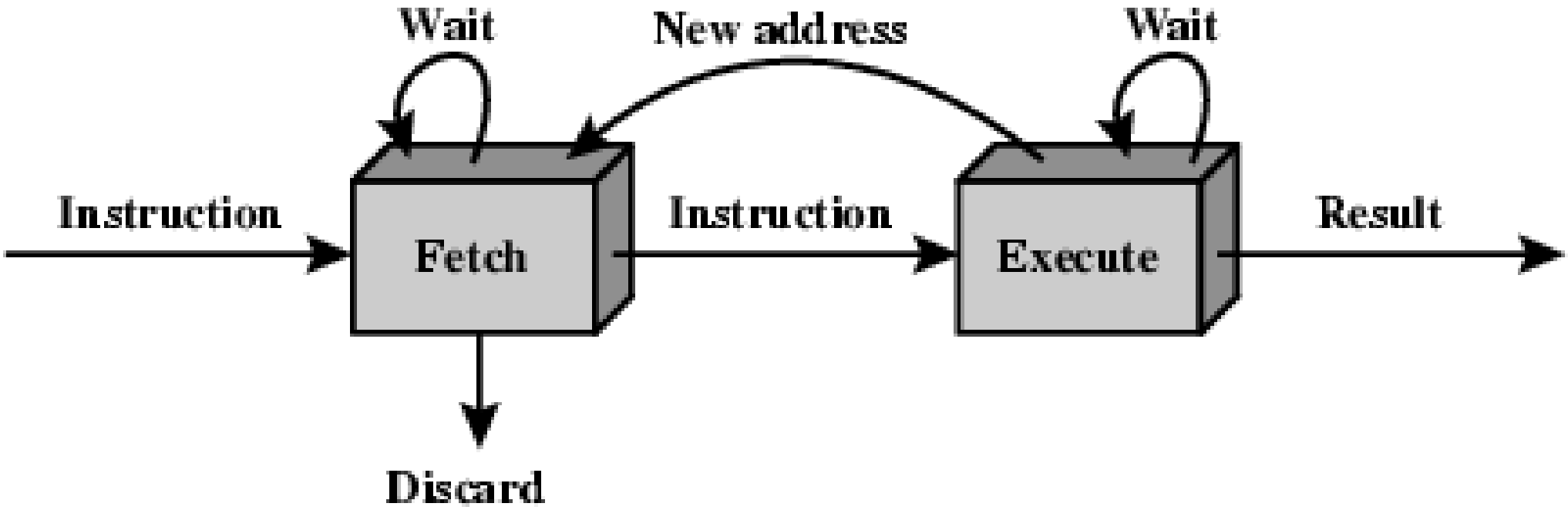
Improved Performance

- But not doubled:
  - Fetch usually shorter than execution (cf. reading and storing operands)
    - Prefetch more than one instruction?
  - Any jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performance

Two Stage Instruction Pipeline



(a) Simplified view



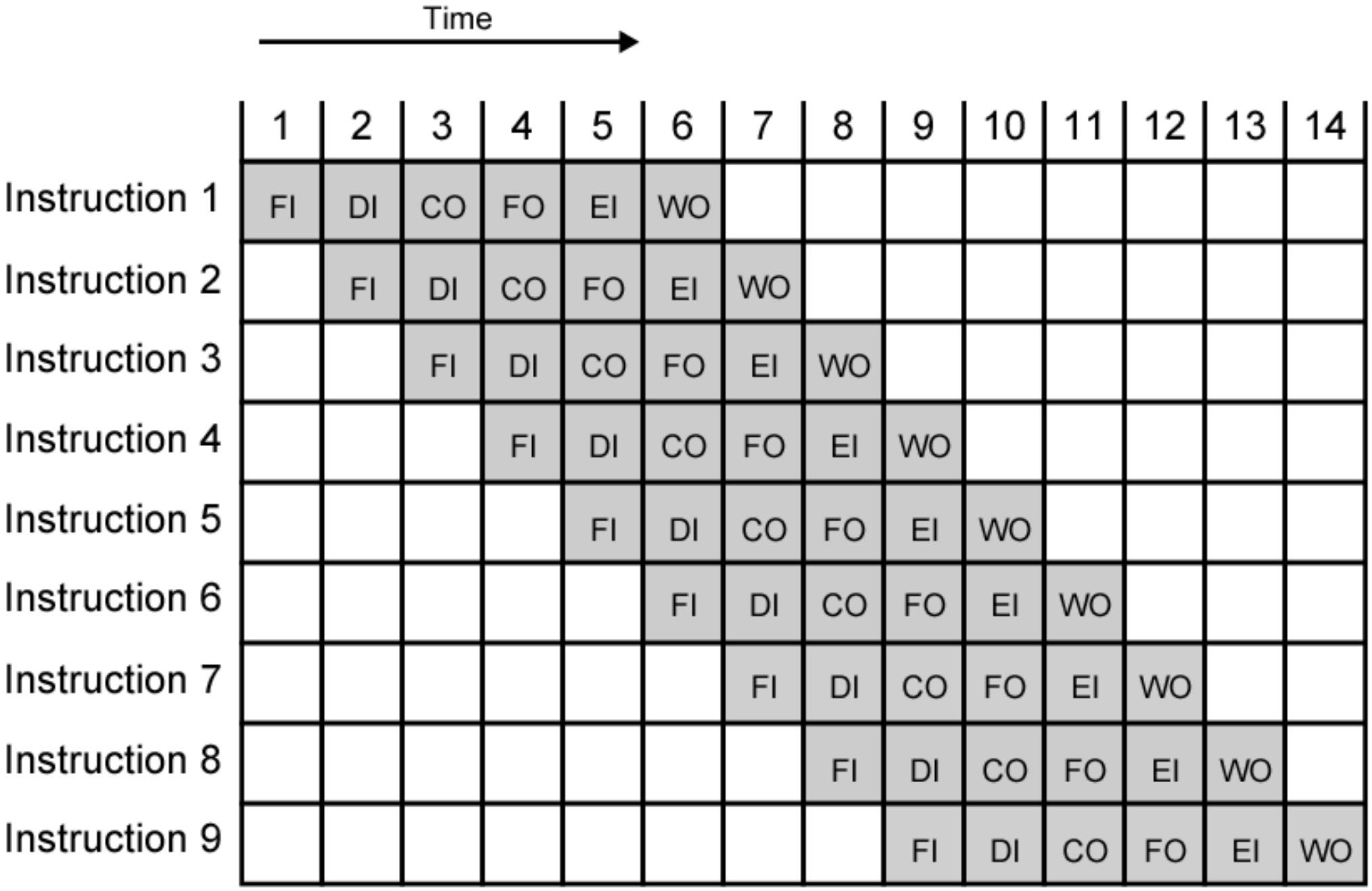
(b) Expanded view

Pipelining (six stages)

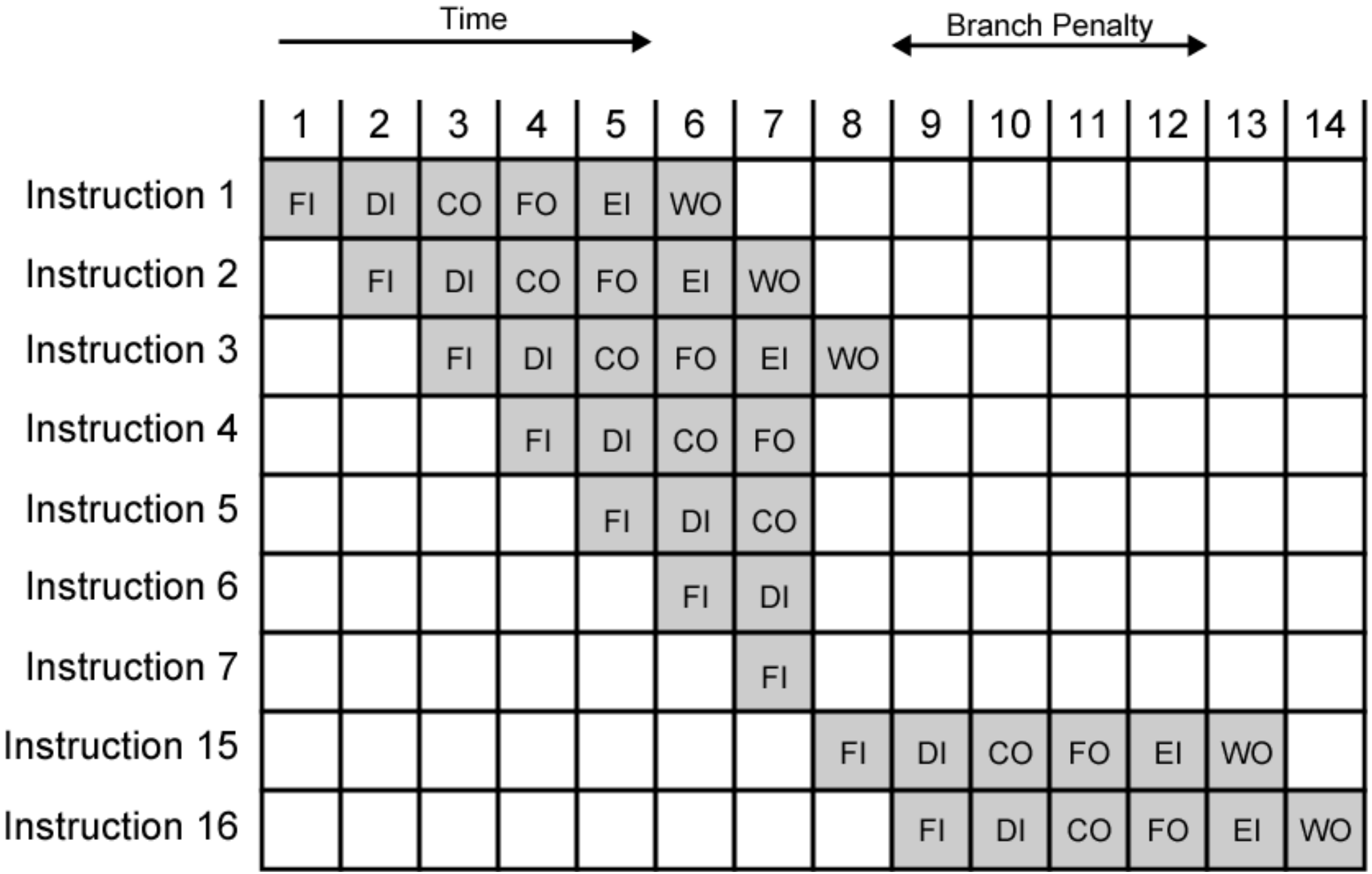
1. Fetch instruction
2. Decode instruction
3. Calculate operands (i.e., EAs)
4. Fetch operands
5. Execute instructions
6. Write result

- Overlap these operations

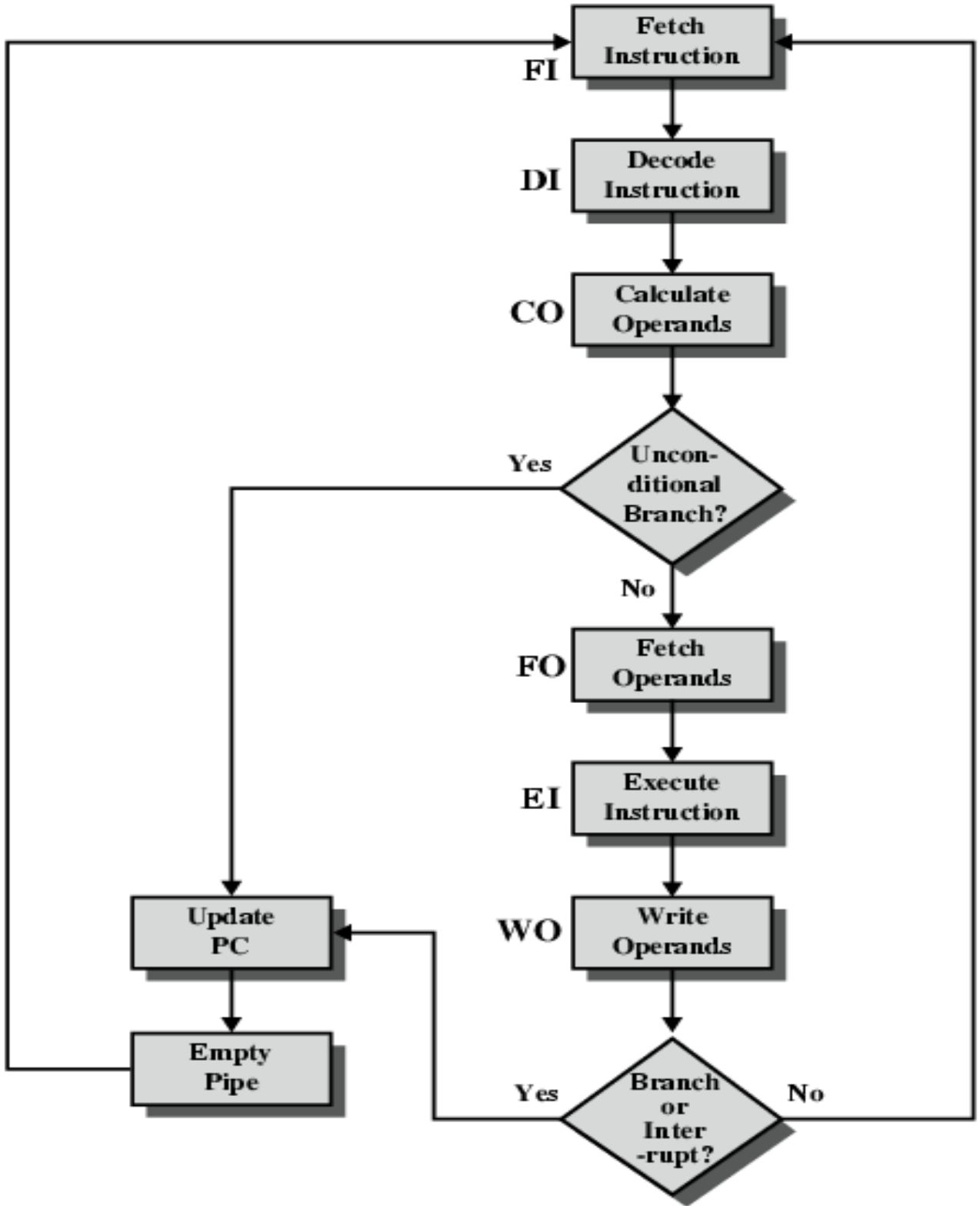
Timing Diagram for Instruction Pipeline Operation (assuming independence)



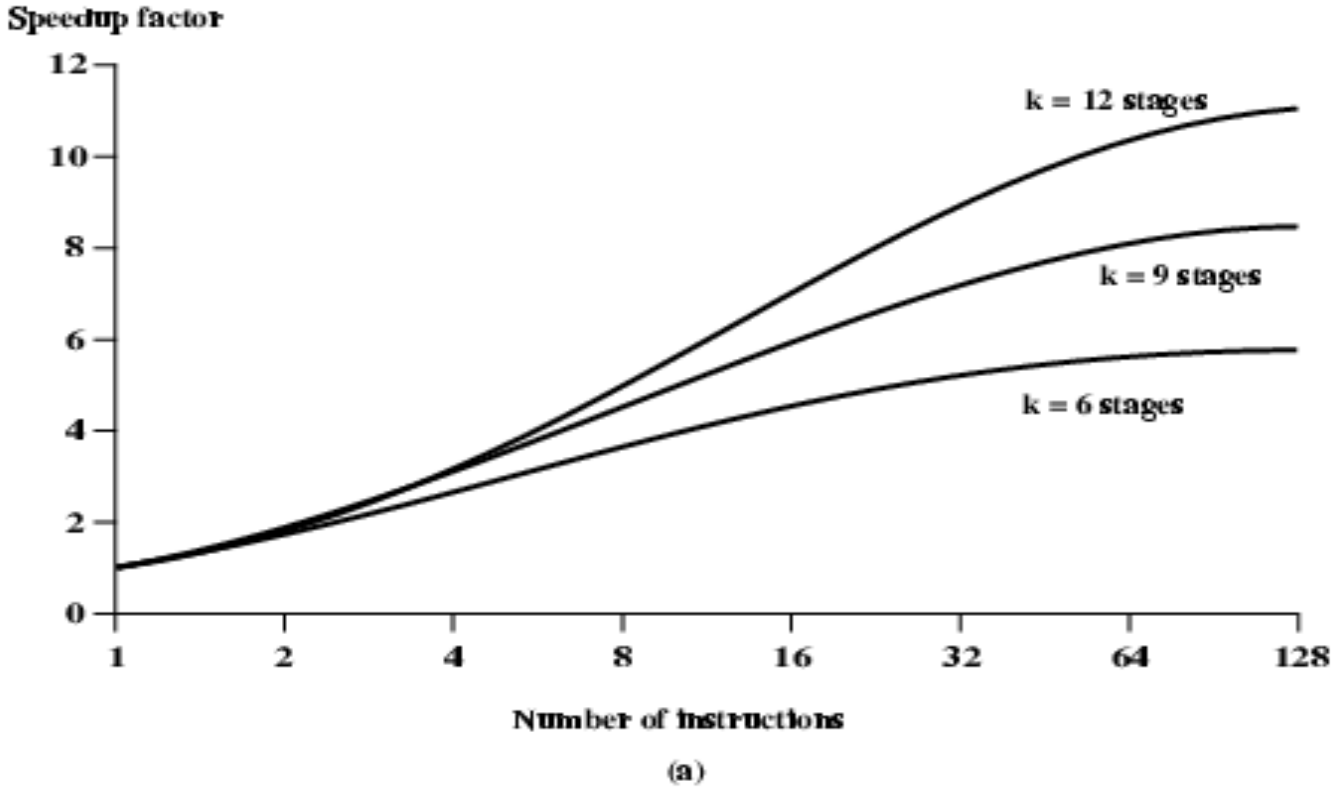
The Effect of a Conditional Branch/Interrupt on Instruction Pipeline Operation



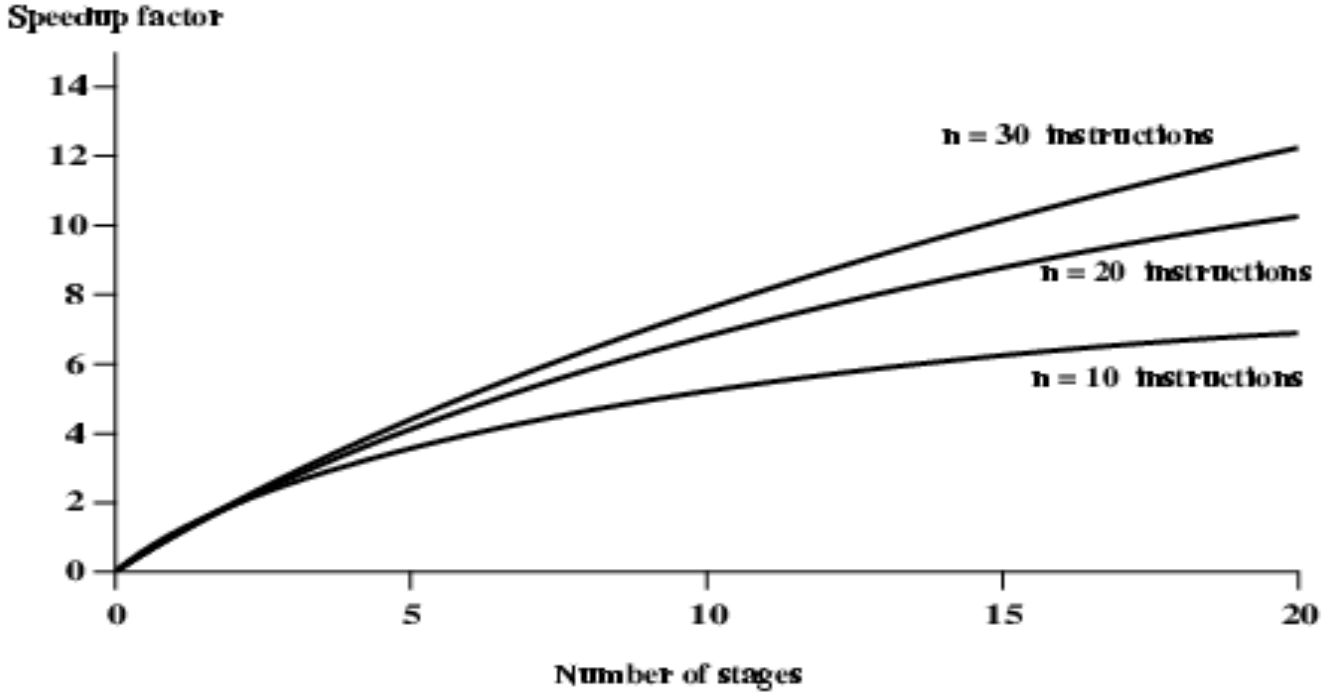
Six Stage Instruction Pipeline



Speedup Factors with Instruction Pipelining: nk/(n+k-1) (ideally)



(a)



(b)

Dealing with Branches

1. Prefetch Branch Target
2. Loop buffer
3. Branch prediction
4. Delayed branching (see RISC)

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
- cf. cache

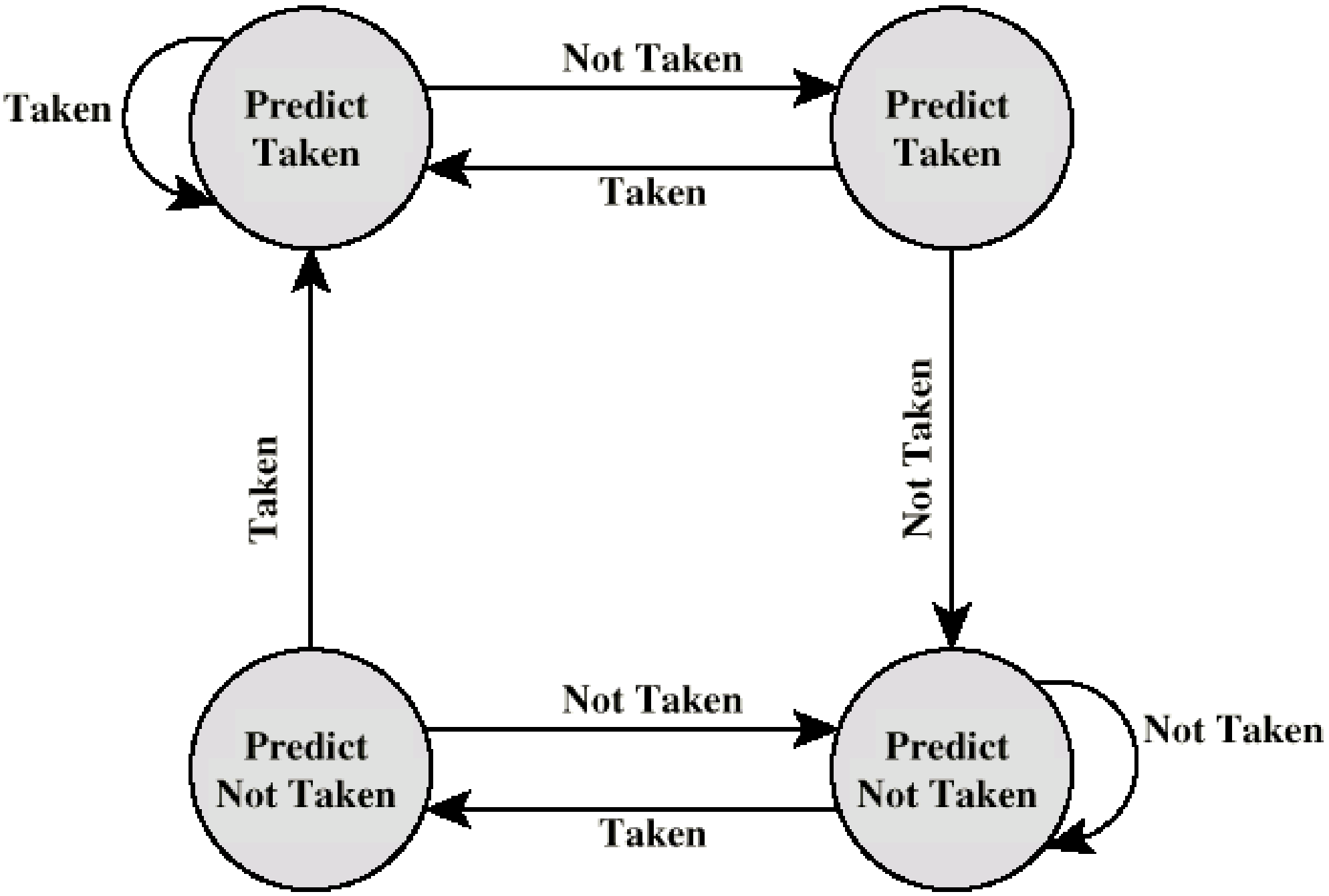
Branch Prediction (1)

- Predict never taken
  - Assume that jump will not happen
  - Always (almost) fetch next instruction
  - VAX will not prefetch after branch if a page fault would result (OS v CPU design)
- Predict always taken
  - Assume that jump will happen (at least 50%)
  - Always fetch target instruction

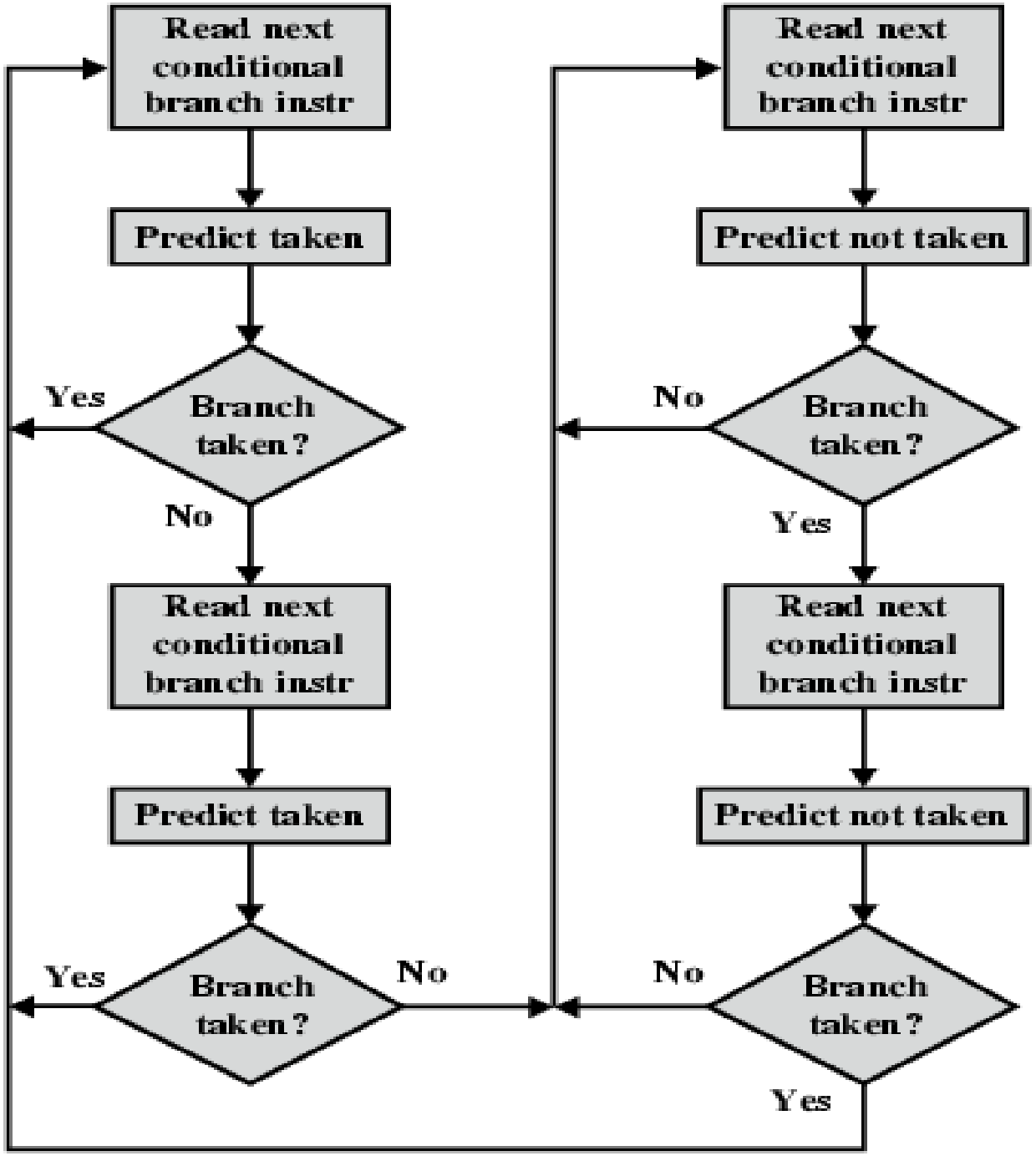
Branch Prediction (2)

- 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
- Delayed branch – rearrange instructions (see RISC)

Branch Prediction State Diagram (two bits)



Branch Prediction Flowchart



Intel 80486 Pipelining

1. Fetch
   - 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 (variable size)
   - Independent of other stages to keep buffers full
1. Decode stage 1
   - Opcode & address-mode info
   - At most first 3 bytes of instruction needed for this
   - Can direct D2 stage to get rest of instruction
1. Decode stage 2
   - Expand opcode into control signals
   - Computation of complex addressing modes
1. Execute
   - ALU operations, cache access, register update
1. Writeback
   - Update registers & flags
   - Results sent to cache

Pentium 4 Registers

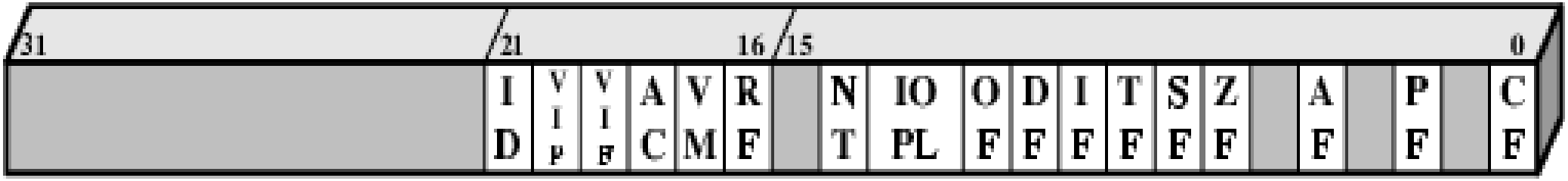
(a) Integer Unit

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	48	Points to instruction interrupted by exception
Data Pointer	1	48	Points to operand interrupted by exception

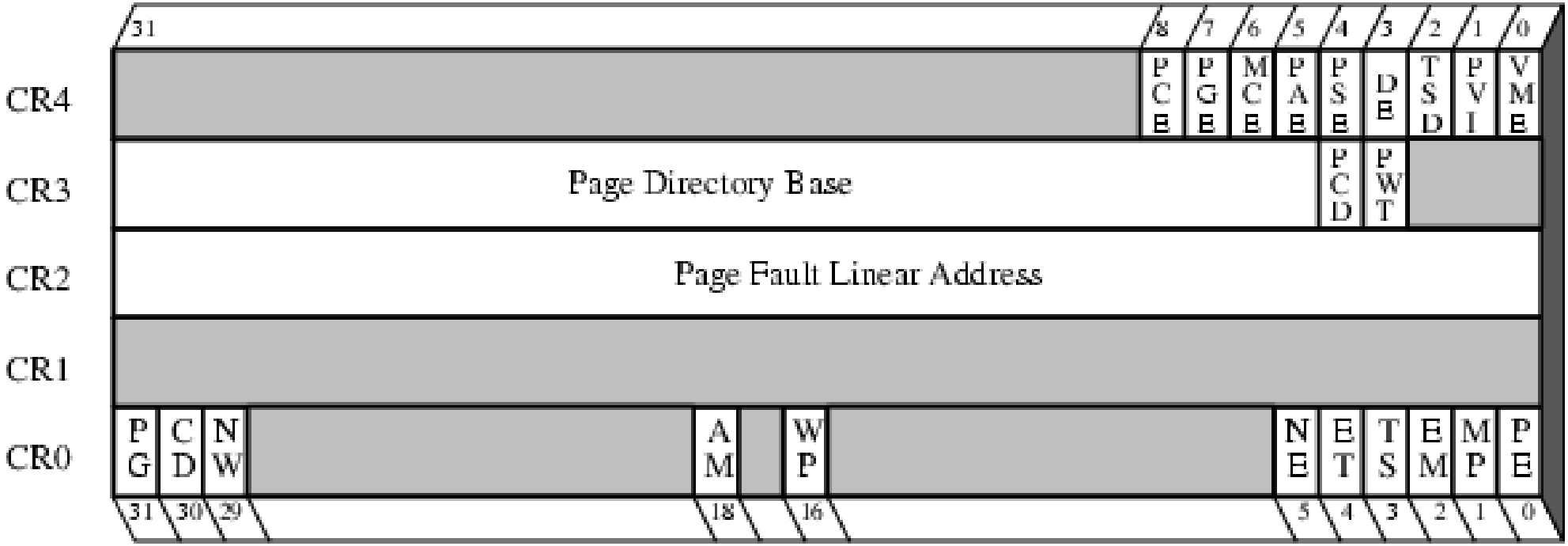
EFLAGS Register



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	PG	=	Paging
PGE	=	Page Global Enable	CD	=	Cache Disable
MCE	=	Machine Check Enable	NW	=	Not Write Through
PAE	=	Physical Address Extension	AM	=	Alignment Mask
PSE	=	Page Size Extensions	WP	=	Write Protect
DE	=	Debug Extensions	NE	=	Numeric Error
TSD	=	Time Stamp Disable	ET	=	Extension Type
PVI	=	Protected Mode Virtual Interrupt	TS	=	Task Switched
VME	=	Virtual 8086 Mode Extensions	EM	=	Emulation
PCD	=	Page-level Cache Disable	MP	=	Monitor Coprocessor
PWT	=	Page-level Writes Transparent	PE	=	Protection Enable

Pentium Interrupt Processing

- Interrupts (hardware): (non-)maskable
- Exceptions (software): processor detected (error) or programmed (exception)
- Interrupt vector table
  - Each interrupt type assigned a number
  - Index to vector table
  - 256 * 32 bit interrupt vectors (address of ISR)
- 5 priority classes: 1. exception by previous instruction 2. external interrupt, 3.-5. faults from fetching, decoding or executing instruction