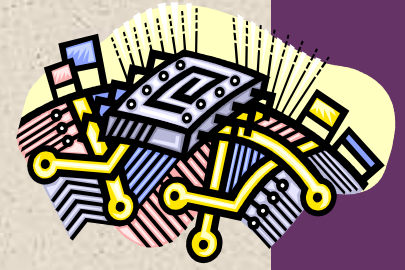




Processor Organization



Processor Requirements:

- **Fetch instruction**
 - The processor reads an instruction from memory (register, cache, main memory)
- **Interpret instruction**
 - The instruction is decoded to determine what action is required
- **Fetch data**
 - The execution of an instruction may require reading data from memory or an I/O module
- **Process data**
 - The execution of an instruction may require performing some arithmetic or logical operation on data
- **Write data**
 - The results of an execution may require writing data to memory or an I/O module
- In order to do these things the processor needs to store some data temporarily and therefore needs a small internal memory

CPU With the System Bus

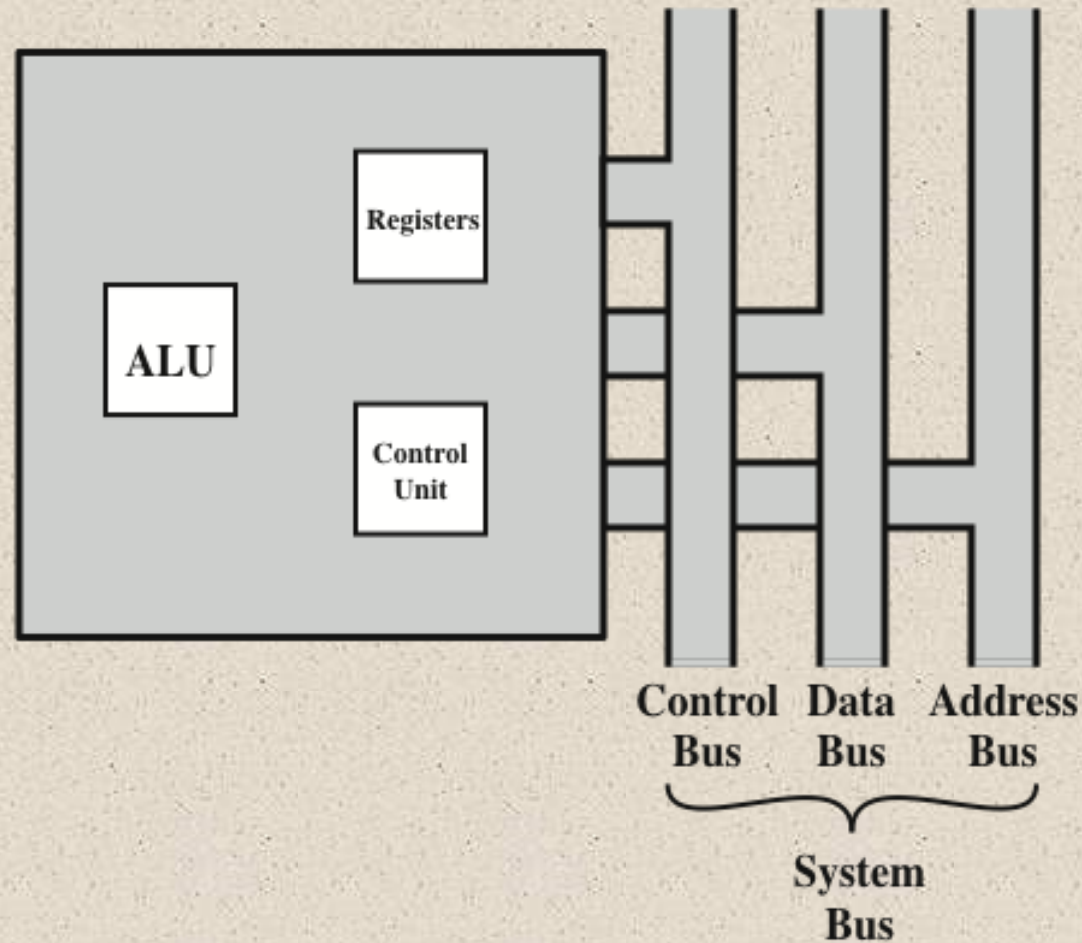


Figure 14.1 The CPU with the System Bus

CPU Internal Structure

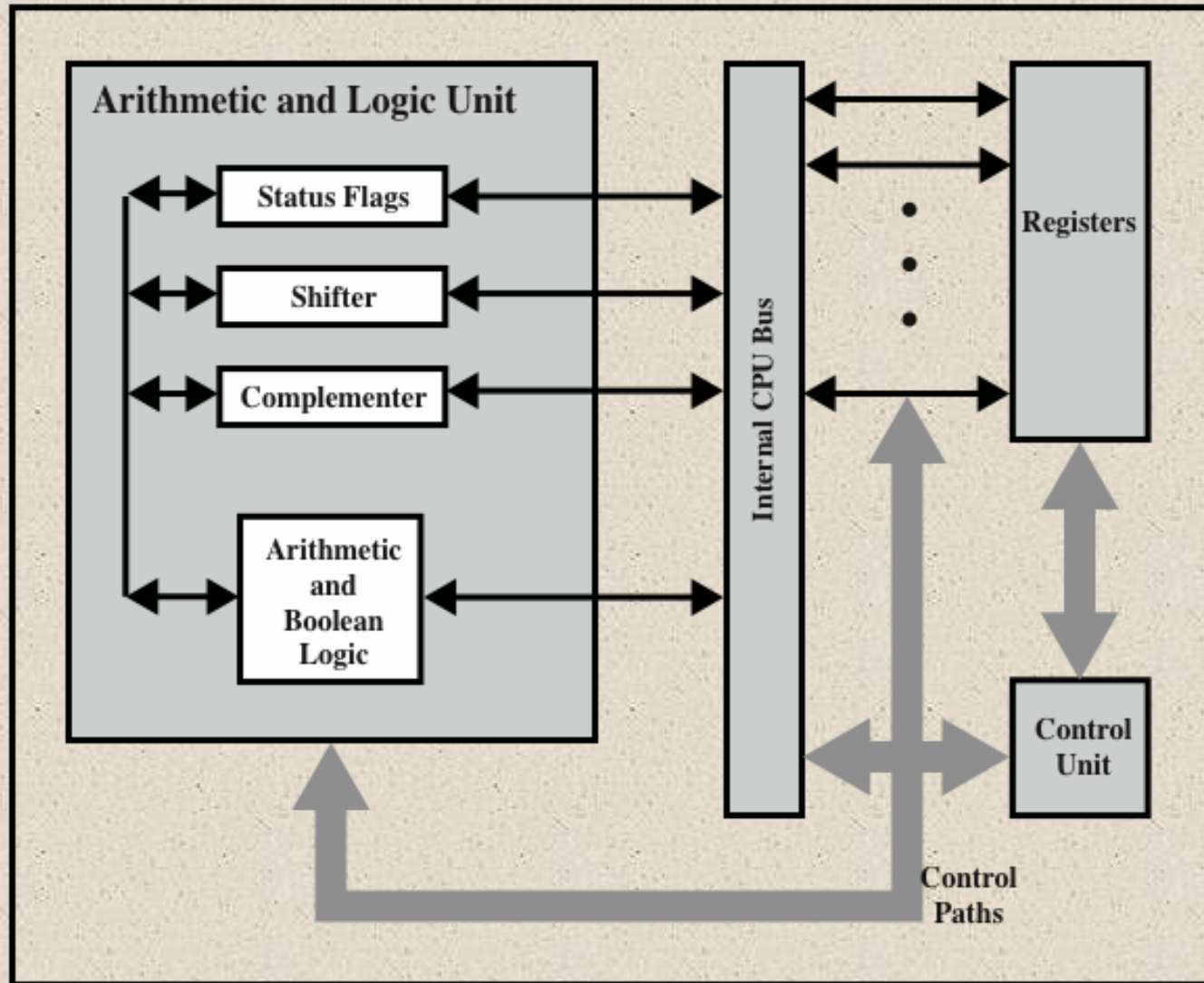


Figure 14.2 Internal Structure of the CPU



Register Organization



- Within the processor there is a set of registers that function as a level of memory above main memory and cache in the hierarchy
- The registers in the processor perform two roles:

User-Visible Registers

- Enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers

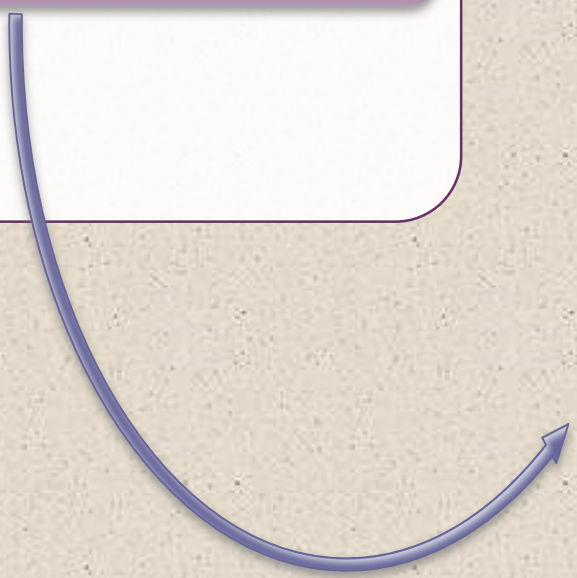
Control and Status Registers

- Used by the control unit to control the operation of the processor and by privileged operating system programs to control the execution of programs

User-Visible Registers

Categories:

Referenced by means of
the machine language
that the processor
executes



- **General purpose**

- Can be assigned to a variety of functions by the programmer

- **Data**

- May be used only to hold data and cannot be employed in the calculation of an operand address

- **Address**

- May be somewhat general purpose or may be devoted to a particular addressing mode
- Examples: segment pointers, index registers, stack pointer

- **Condition codes**

- Also referred to as *flags*
- Bits set by the processor hardware as the result of operations



Control and Status Registers

Four registers are essential to instruction execution:

- Program counter (PC)
 - Contains the address of an instruction to be fetched
- Instruction register (IR)
 - Contains the instruction most recently fetched
- Memory address register (MAR)
 - Contains the address of a location in memory
- Memory buffer register (MBR)
 - Contains a word of data to be written to memory or the word most recently read

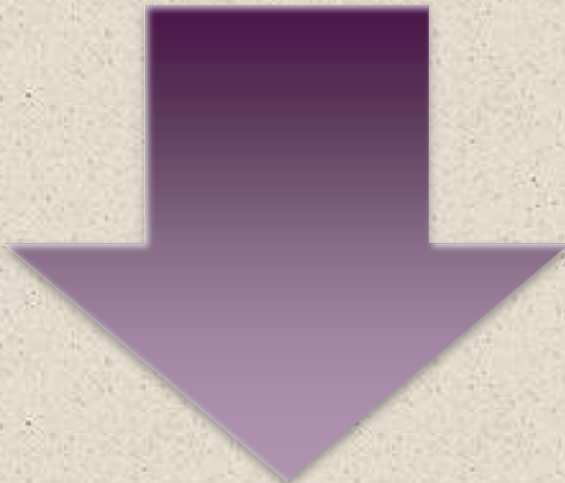




Program Status Word (PSW)



Register or set of registers that contain status information



Common fields or flags include:

- Sign
- Zero
- Carry
- Equal
- Overflow
- Interrupt Enable/Disable
- Supervisor

Data registers	
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

Address registers	
A0	
A1	
A2	
A3	
A4	
A5	
A6	
A7	

Program status	
Program counter	
Status register	

(a) MC68000

General registers

AX	Accumulator
BX	Base
CX	Count
DX	Data

Pointers & index

SP	Stack ptr
BP	Base ptr
SI	Source index
DI	Dest index

Segment

CS	Code
DS	Data
SS	Stack
ES	Extrat

Program status

Flags
Instr ptr

(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 4

Example Microprocessor Register Organizations

Figure 14.3 Example Microprocessor Register Organizations

+ Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack





Addressing Modes

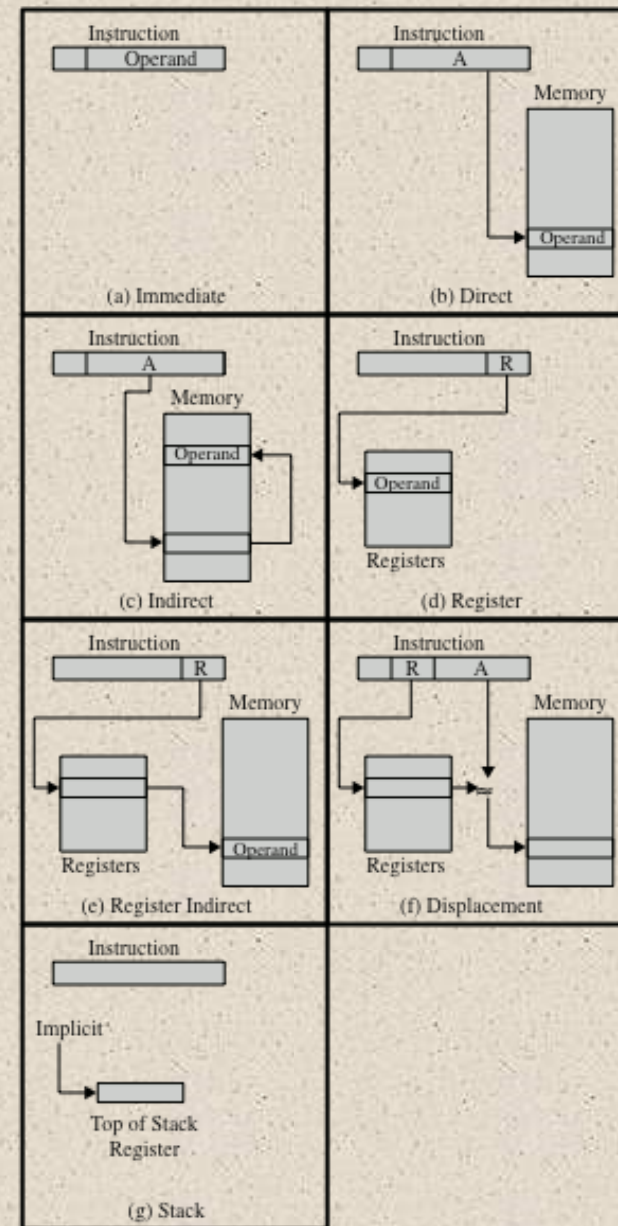


Figure 13.1 Addressing Modes



Basic Addressing Modes



Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Table 13.1 Basic Addressing Modes

Table 13.2

x86 Addressing Modes

Mode	Algorithm
Immediate	$\text{Operand} = A$
Register Operand	$\text{LA} = R$
Displacement	$\text{LA} = (\text{SR}) + A$
Base	$\text{LA} = (\text{SR}) + (B)$
Base with Displacement	$\text{LA} = (\text{SR}) + (B) + A$
Scaled Index with Displacement	$\text{LA} = (\text{SR}) + (I) \times S + A$
Base with Index and Displacement	$\text{LA} = (\text{SR}) + (B) + (I) + A$
Base with Scaled Index and Displacement	$\text{LA} = (\text{SR}) + (I) \times S + (B) + A$
Relative	$\text{LA} = (\text{PC}) + A$

LA = linear address
 (X) = contents of X
 SR = segment register
 PC = program counter
 A = contents of an address field in the instruction
 R = register
 B = base register
 I = index register
 S = scaling factor

Instruction Formats



Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

+ Instruction Length

- Most basic design issue
- Affects, and is affected by:
 - Memory size
 - Memory organization
 - Bus structure
 - Processor complexity
 - Processor speed
- Should be equal to the memory-transfer length or one should be a multiple of the other
- Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers



Allocation of Bits



- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

PDP-8 Instruction Format



Memory Reference Instructions					
Opcode		D/I	Z/C	Displacement	
0	2	3	4	5	11

Input/Output Instructions						
1	1	0	Device			Opcode
0	2	3			8	9 11

Register Reference Instructions											
Group 1 Microinstructions				CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
1	1	1	0	4	5	6	7	8	9	10	11

Group 2 Microinstructions											
1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 Microinstructions											
1	1	1	1	CLA	MQA	0	SQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Direct/Indirect address
 Z/C = Page 0 or Current page
 CLA = Clear Accumulator
 CLL = Clear Link
 CMA = CoMplement Accumulator
 CML = CoMplement Link
 RAR = Rotate Accumulator Right
 RAL = Rotate Accumulator Left
 BSW = Byte SWap

IAC = Increment ACcumulator
 SMA = Skip on Minus Accumulator
 SZA = Skip on Zero Accumulator
 SNL = Skip on Nonzero Link
 RSS = Reverse Skip Sense
 OSR = Or with Switch Register
 HLT = HaLT
 MQA = Multiplier Quotient into Accumulator
 SQL = Multiplier Quotient Load

Figure 11.5 PDP-8 Instruction Formats



Variable-Length Instructions



- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
 - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
 - Sometimes multiple instructions are fetched

x86 Instruction Format

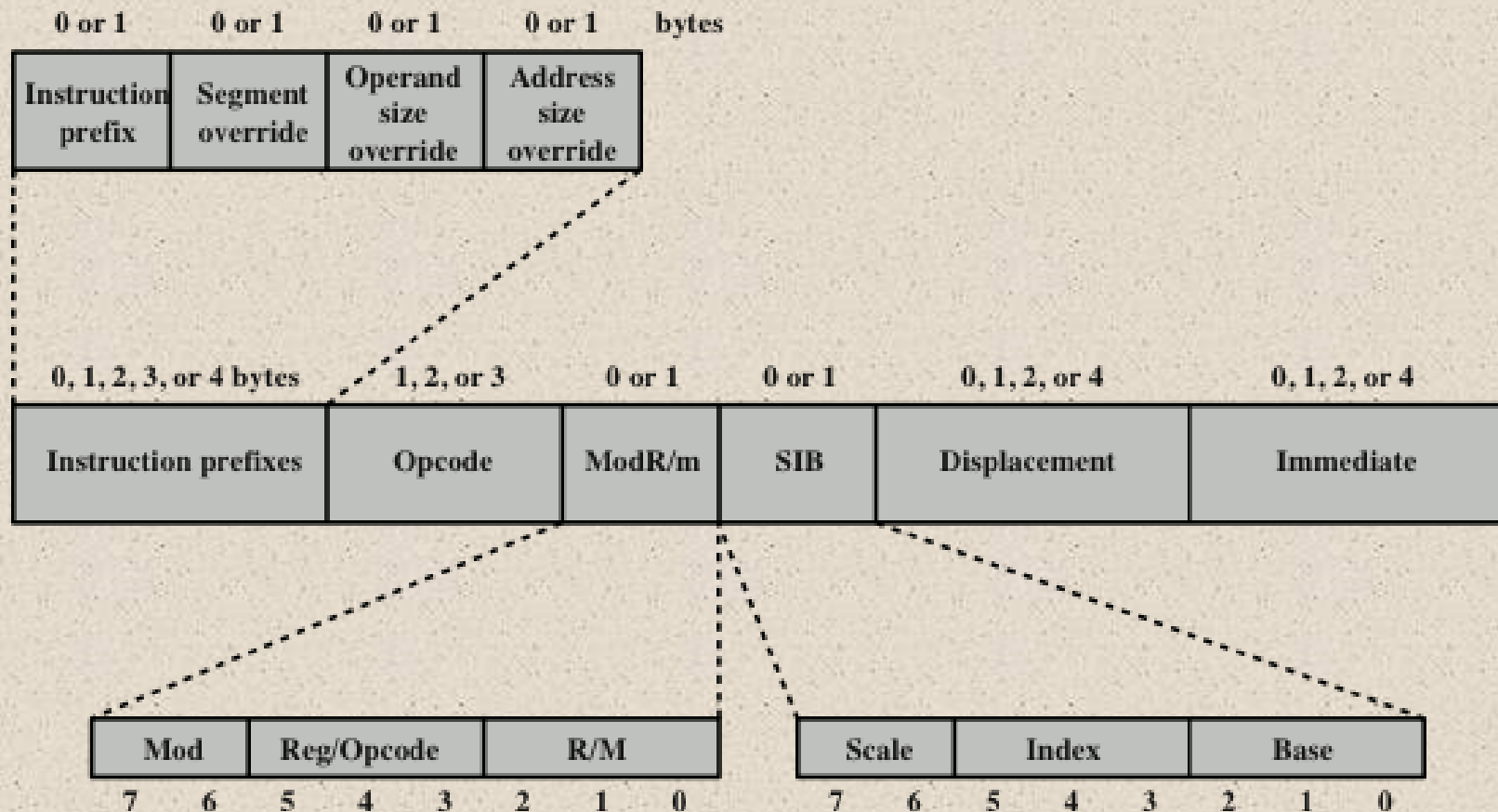


Figure 13.9 x86 Instruction Format

ARM Instruction Formats

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
data processing immediate shift	cond		0			0			0			opcode				S	Rn				Rd				shift amount				shift		0	Rm				
data processing register shift	cond		0			0			0			opcode				S	Rn				Rd				Rs				0	shift		1	Rm			
data processing immediate	cond		0			0			1			opcode				S	Rn				Rd				rotate				immediate							
load/store immediate offset	cond		0			1			0			P	U	B	W	L	Rn				Rd				immediate											
load/store register offset	cond		0			1			1			P	U	B	W	L	Rn				Rd				shift amount				shift		0	Rm				
load/store multiple	cond		1			0			0			P	U	S	W	L	Rn				register list															
branch/branch with link	cond		1			0			1			L	24-bit offset																							

S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Figure 13.10 ARM Instruction Formats

Instruction Cycle

Includes the following stages:

Fetch

Read the next instruction from memory into the processor

Execute

Interpret the opcode and perform the indicated operation

Interrupt

If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt

Instruction Cycle

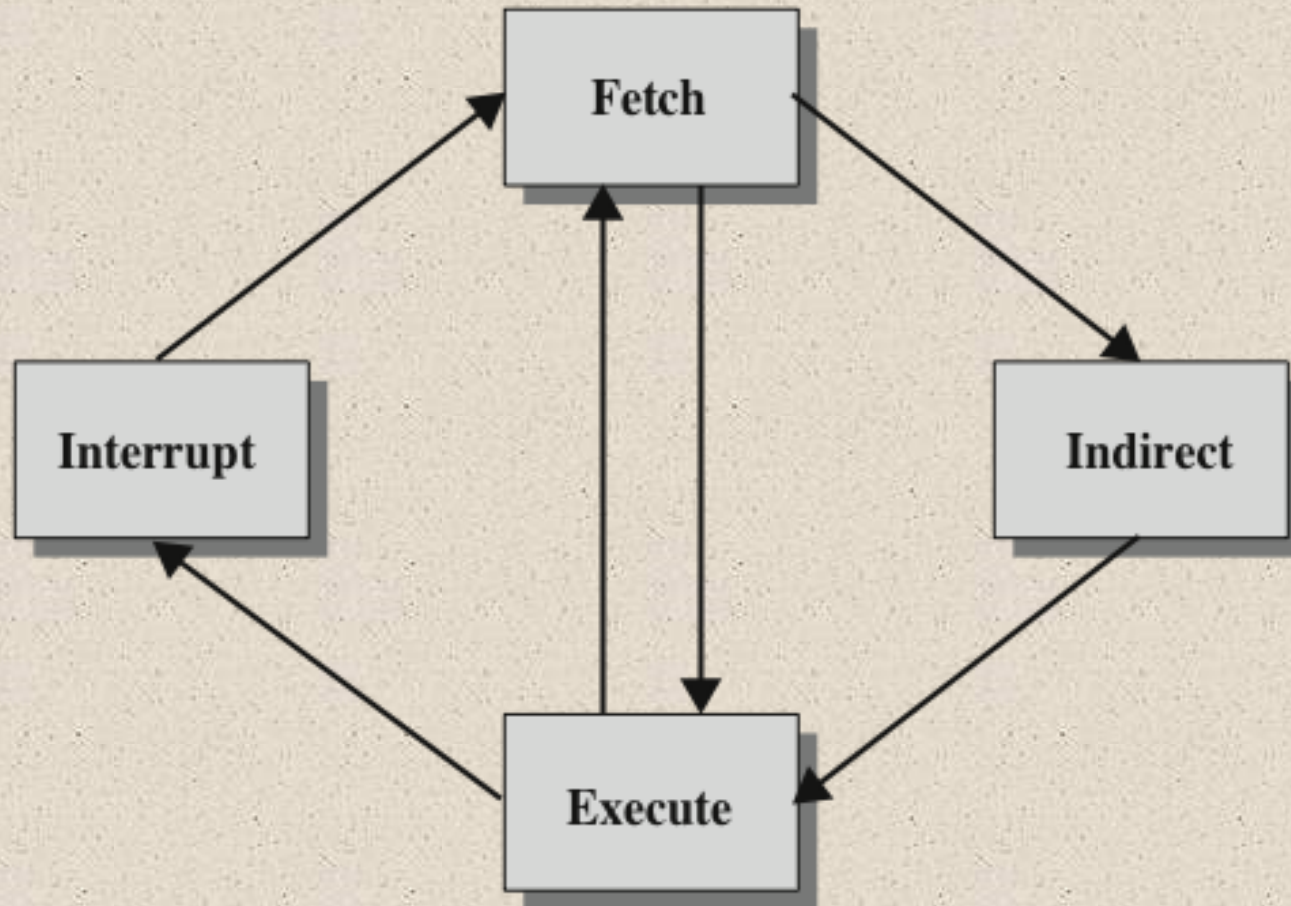


Figure 14.4 The Instruction Cycle

Instruction Cycle State Diagram

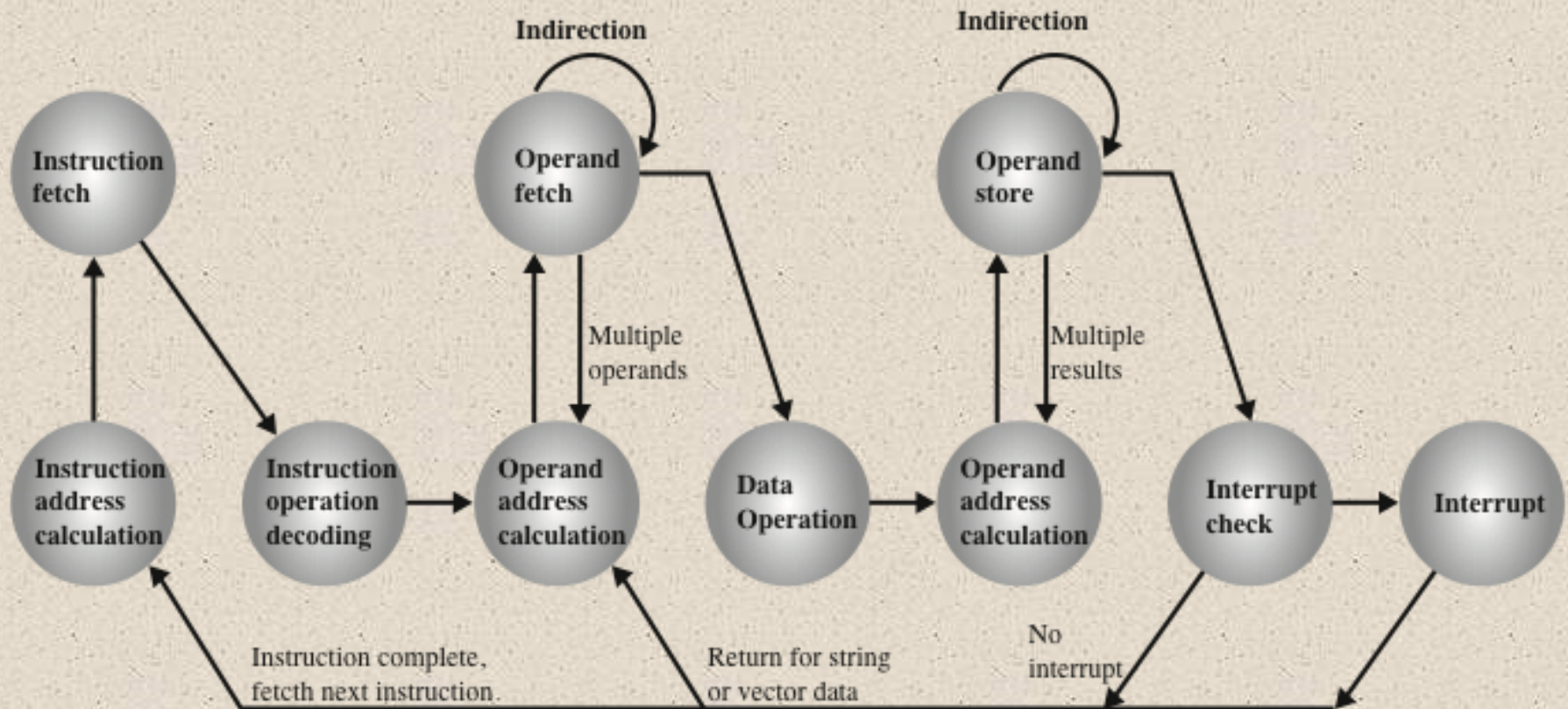
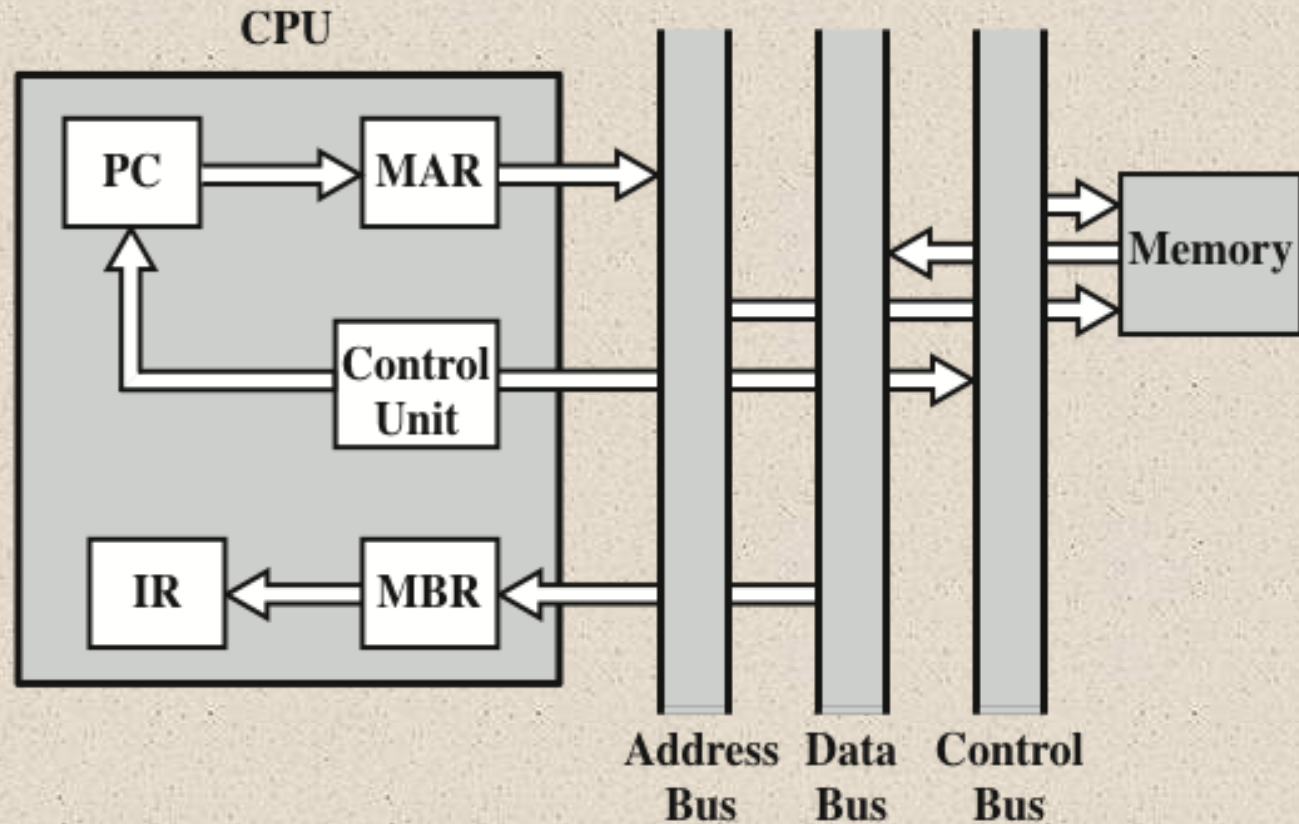


Figure 14.5 Instruction Cycle State Diagram

Data Flow, Fetch Cycle



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

Figure 14.6 Data Flow, Fetch Cycle

Data Flow, Indirect Cycle

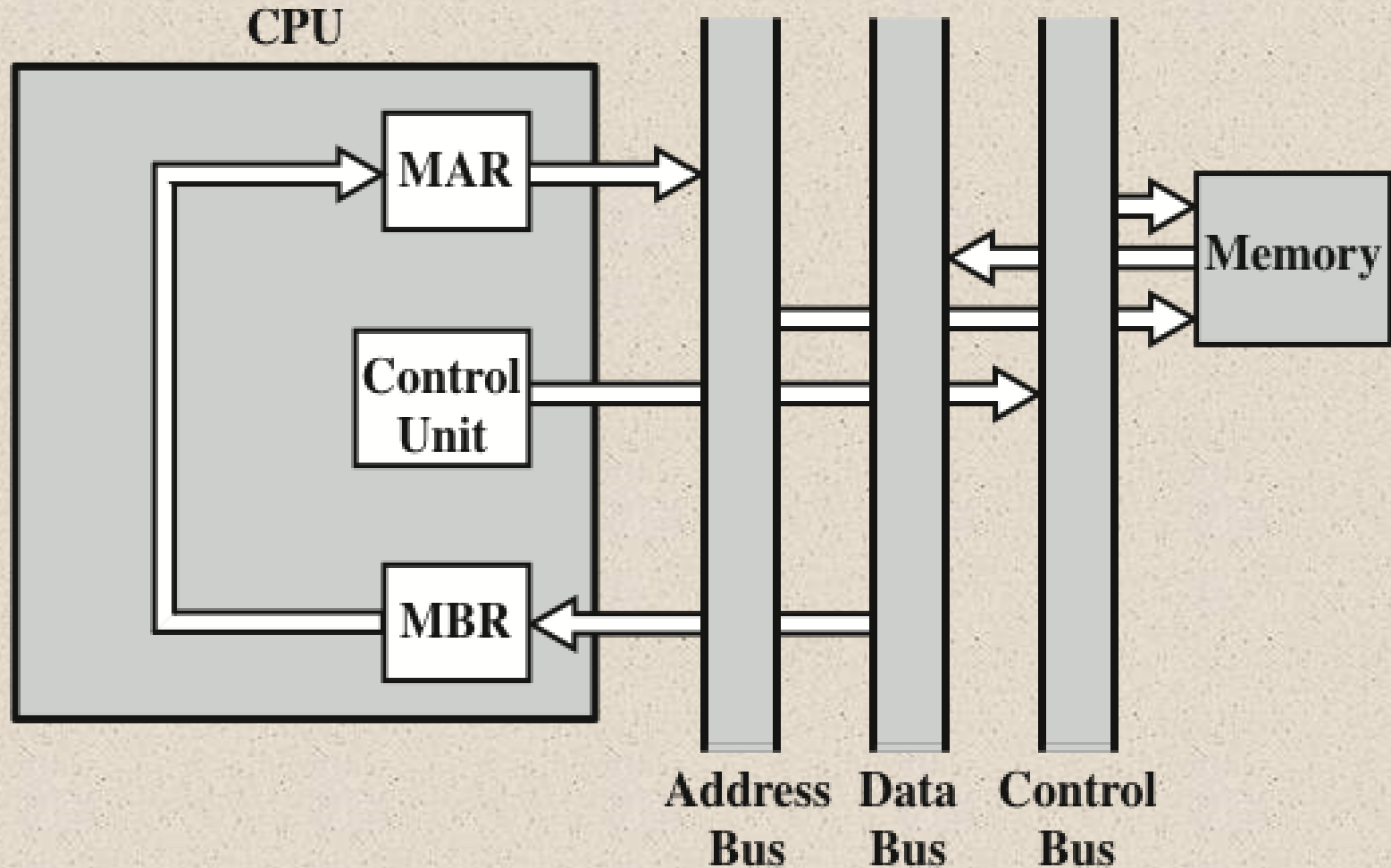


Figure 14.7 Data Flow, Indirect Cycle

Data Flow, Interrupt Cycle

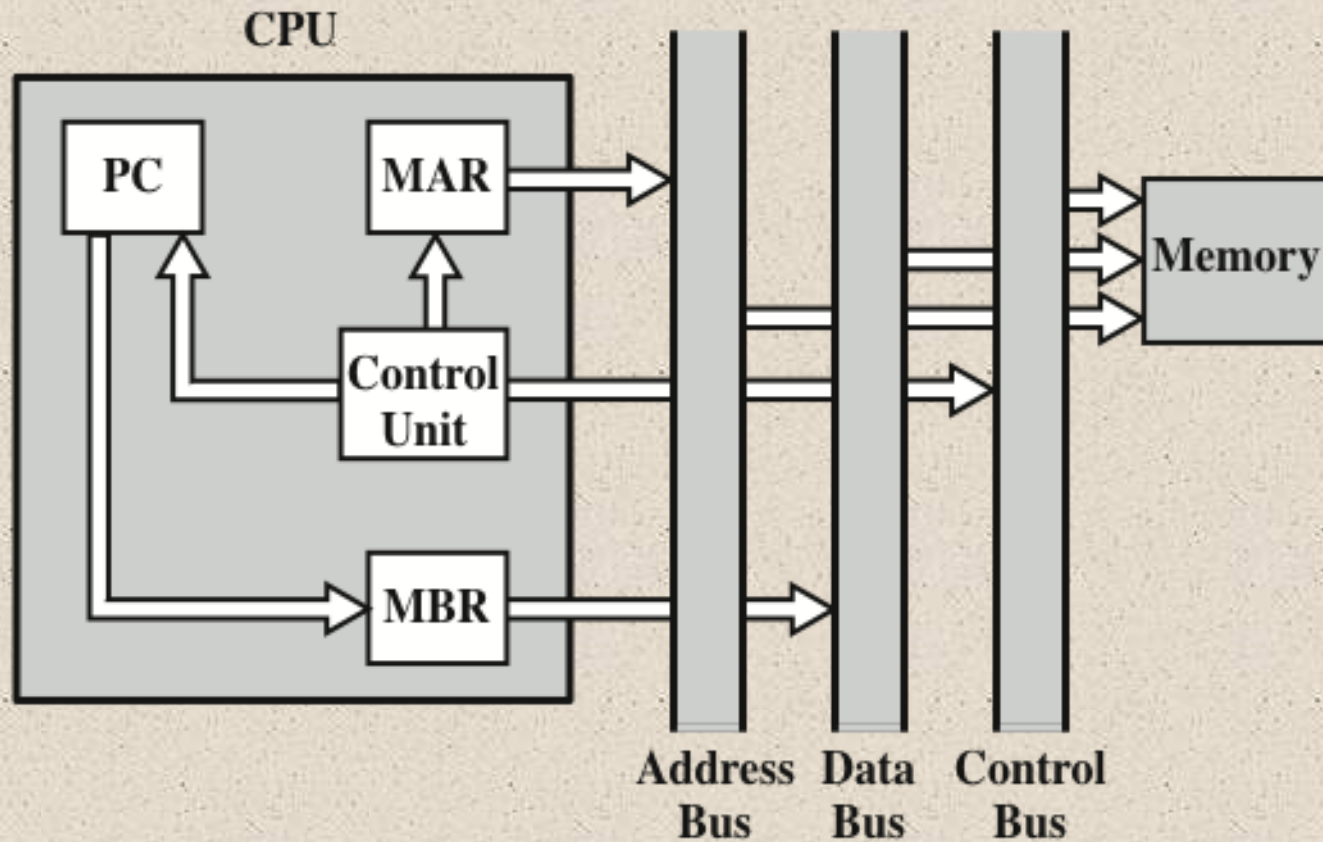


Figure 14.8 Data Flow, Interrupt Cycle

Instruction Formats

■ Three-Address Instructions

■ ADD R1, R2, R3 $R1 \leftarrow R2 + R3$

■ Two-Address Instructions

■ ADD R1, R2 $R1 \leftarrow R1 + R2$

■ One-Address Instructions

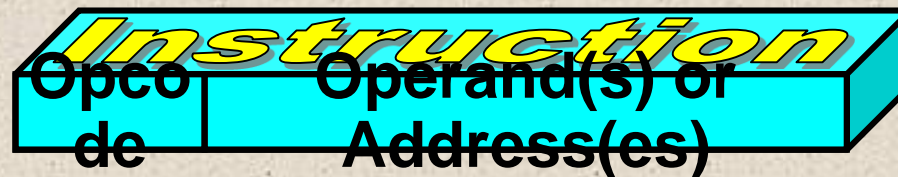
■ ADD M $AC \leftarrow AC + M[AR]$

■ Zero-Address Instructions

■ ADD $TOS \leftarrow TOS + (TOS - 1)$

■ RISC Instructions

■ Lots of registers. Memory is restricted to Load & Store



Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

■ Three-Address

- | | | | |
|----|-----|-----------|-------------------------------|
| 1. | ADD | R1, A, B | ; $R1 \leftarrow M[A] + M[B]$ |
| 2. | ADD | R2, C, D | ; $R2 \leftarrow M[C] + M[D]$ |
| 3. | MUL | X, R1, R2 | ; $M[X] \leftarrow R1 * R2$ |

Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

■ Two-Address

- | | | | |
|----|-----|--------|-----------------------------|
| 1. | MOV | R1, A | ; R1 \leftarrow M[A] |
| 2. | ADD | R1, B | ; R1 \leftarrow R1 + M[B] |
| 3. | MOV | R2, C | ; R2 \leftarrow M[C] |
| 4. | ADD | R2, D | ; R2 \leftarrow R2 + M[D] |
| 5. | MUL | R1, R2 | ; R1 \leftarrow R1 * R2 |
| 6. | MOV | X, R1 | ; M[X] \leftarrow R1 |

Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

■ One-Address

1.	LOAD	A	; $AC \leftarrow M[A]$
2.	ADD	B	; $AC \leftarrow AC + M[B]$
3.	STORE	T	; $M[T] \leftarrow AC$
4.	LOAD	C	; $AC \leftarrow M[C]$
5.	ADD	D	; $AC \leftarrow AC + M[D]$
6.	MUL	T	; $AC \leftarrow AC * M[T]$
7.	STORE	X	; $M[X] \leftarrow AC$

Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

■ Zero-Address

- | | | | |
|----|------|---|--------------------------------|
| 1. | PUSH | A | ; TOS \leftarrow A |
| 2. | PUSH | B | ; TOS \leftarrow B |
| 3. | ADD | | ; TOS \leftarrow (A + B) |
| 4. | PUSH | C | ; TOS \leftarrow C |
| 5. | PUSH | D | ; TOS \leftarrow D |
| 6. | ADD | | ; TOS \leftarrow (C + D) |
| 7. | MUL | | ; TOS \leftarrow (C+D)*(A+B) |
| 8. | POP | X | ; M[X] \leftarrow TOS |



Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

■ RISC

- | | | | |
|----|-------|------------|---------------------------|
| 1. | LOAD | R1, A | ; R1 \leftarrow M[A] |
| 2. | LOAD | R2, B | ; R2 \leftarrow M[B] |
| 3. | LOAD | R3, C | ; R3 \leftarrow M[C] |
| 4. | LOAD | R4, D | ; R4 \leftarrow M[D] |
| 5. | ADD | R1, R1, R2 | ; R1 \leftarrow R1 + R2 |
| 6. | ADD | R3, R3, R4 | ; R3 \leftarrow R3 + R4 |
| 7. | MUL | R1, R1, R3 | ; R1 \leftarrow R1 * R3 |
| 8. | STORE | X, R1 | ; M[X] \leftarrow R1 |



Functional Requirements for processor



- Operations
- Addressing Modes
- Registers
- I/O Module Interface
- Memory Module Interface
- Interrupts

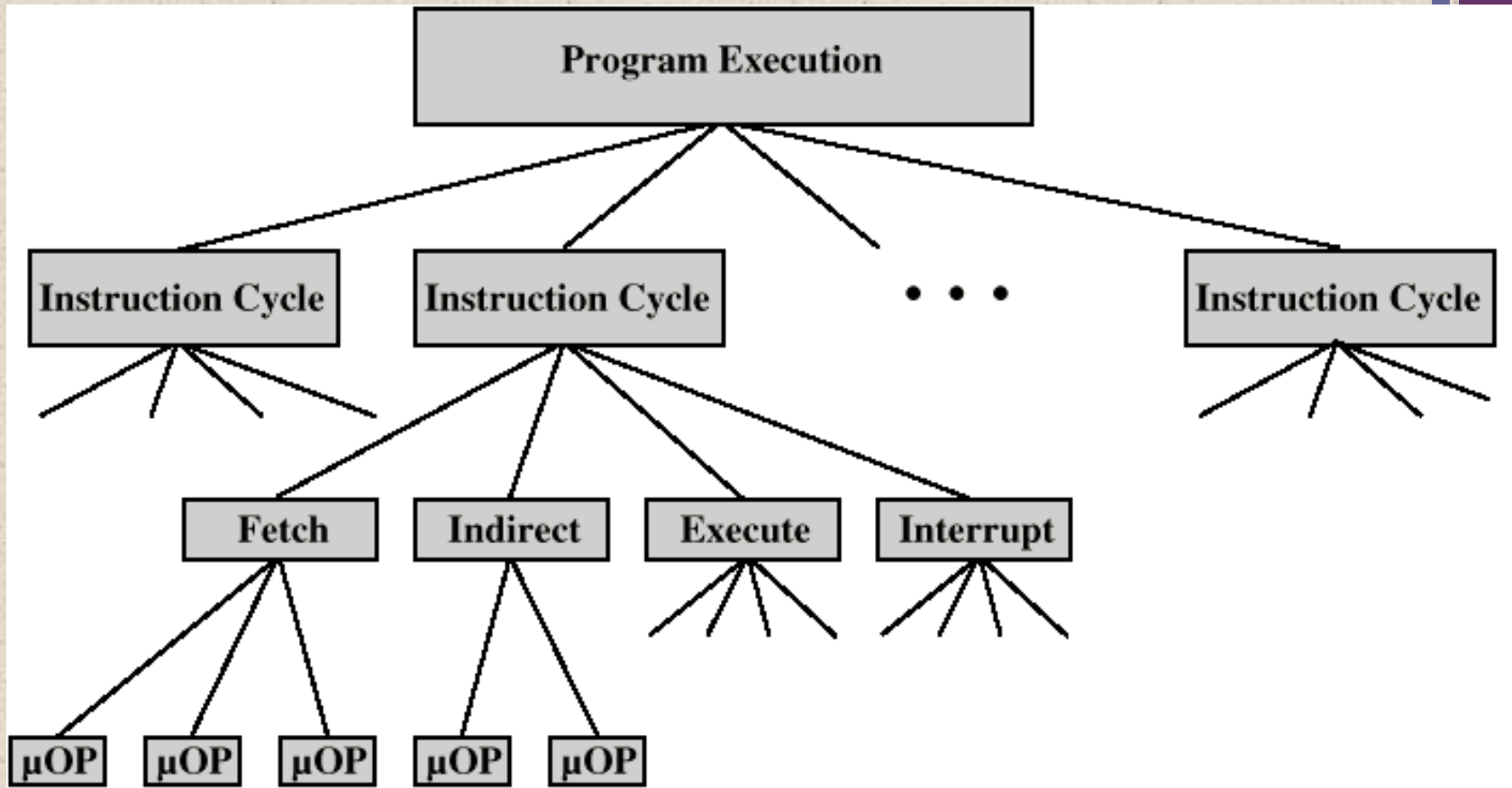


Micro-Operations



- A computer executes a program
- Fetch/execute cycle
- Each cycle has a number of steps Called micro-operations
- Each step does very little
- Atomic operation of CPU

+ Constituent Elements of Program Execution





Fetch - 4 Registers



- Memory Address Register (MAR)
 - Connected to address bus
 - Specifies address for read or write op
- Memory Buffer Register (MBR)
 - Connected to data bus
 - Holds data to write or last data read
- Program Counter (PC)
 - Holds address of next instruction to be fetched
- Instruction Register (IR)
 - Holds last instruction fetched



Fetch Sequence



- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches



Fetch Sequence (symbolic)



- t1: $MAR \leftarrow (PC)$
- t2: $MBR \leftarrow (\text{memory})$
- $PC \leftarrow (PC) + 1$
- t3: $IR \leftarrow (MBR)$
- (tx = time unit/clock cycle)
- or
- t1: $MAR \leftarrow (PC)$
- t2: $MBR \leftarrow (\text{memory})$
- t3: $PC \leftarrow (PC) + 1$
- $IR \leftarrow (MBR)$



Rules for Clock Cycle Grouping



- Proper sequence must be followed
 - $MAR \leftarrow (PC)$ must precede $MBR \leftarrow (memory)$
- Conflicts must be avoided
 - Must not read & write same register at same time
 - $MBR \leftarrow (memory)$ & $IR \leftarrow (MBR)$ must not be in same cycle
- Also: $PC \leftarrow (PC) + 1$ involves addition
 - Use ALU
 - May need additional micro-operations

+ Indirect Cycle



- $MAR \leftarrow (IR_{\text{address}})$ - address field of IR
- $MBR \leftarrow (\text{memory})$
- $IR_{\text{address}} \leftarrow (MBR_{\text{address}})$
- MBR contains an address
- IR is now in same state as if direct addressing had been used
- (What does this say about IR size?)



Interrupt Cycle



- t1: MBR \leftarrow (PC)
- t2: MAR \leftarrow save-address
- PC \leftarrow routine-address
- t3: memory \leftarrow (MBR)
- This is a minimum
 - May be additional micro-ops to get addresses
 - N.B. saving context is done by interrupt handler routine, not micro-ops

+ Execute Cycle (ADD)



- Different for each instruction
- e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1
- t1: $MAR \leftarrow (IR_{\text{address}})$
- t2: $MBR \leftarrow (\text{memory})$
- t3: $R1 \leftarrow R1 + (MBR)$
- Note no overlap of micro-operations

+ Execute Cycle (ISZ)

■ ISZ X - increment and skip if zero

- t1: $MAR \leftarrow (IR_{address})$
- t2: $MBR \leftarrow (memory)$
- t3: $MBR \leftarrow (MBR) + 1$
- t4: $memory \leftarrow (MBR)$
- if $(MBR) == 0$ then $PC \leftarrow (PC) + 1$

■ Notes:

- if is a single micro-operation
- Micro-operations done during t4

+ Execute Cycle (BSA)

- BSA X - Branch and save address
 - Address of instruction following BSA is saved in X
 - Execution continues from X+1
 - t1: $MAR \leftarrow (IR_{\text{address}})$
 - $MBR \leftarrow (PC)$
 - t2: $PC \leftarrow (IR_{\text{address}})$
 - $memory \leftarrow (MBR)$
 - t3: $PC \leftarrow (PC) + 1$



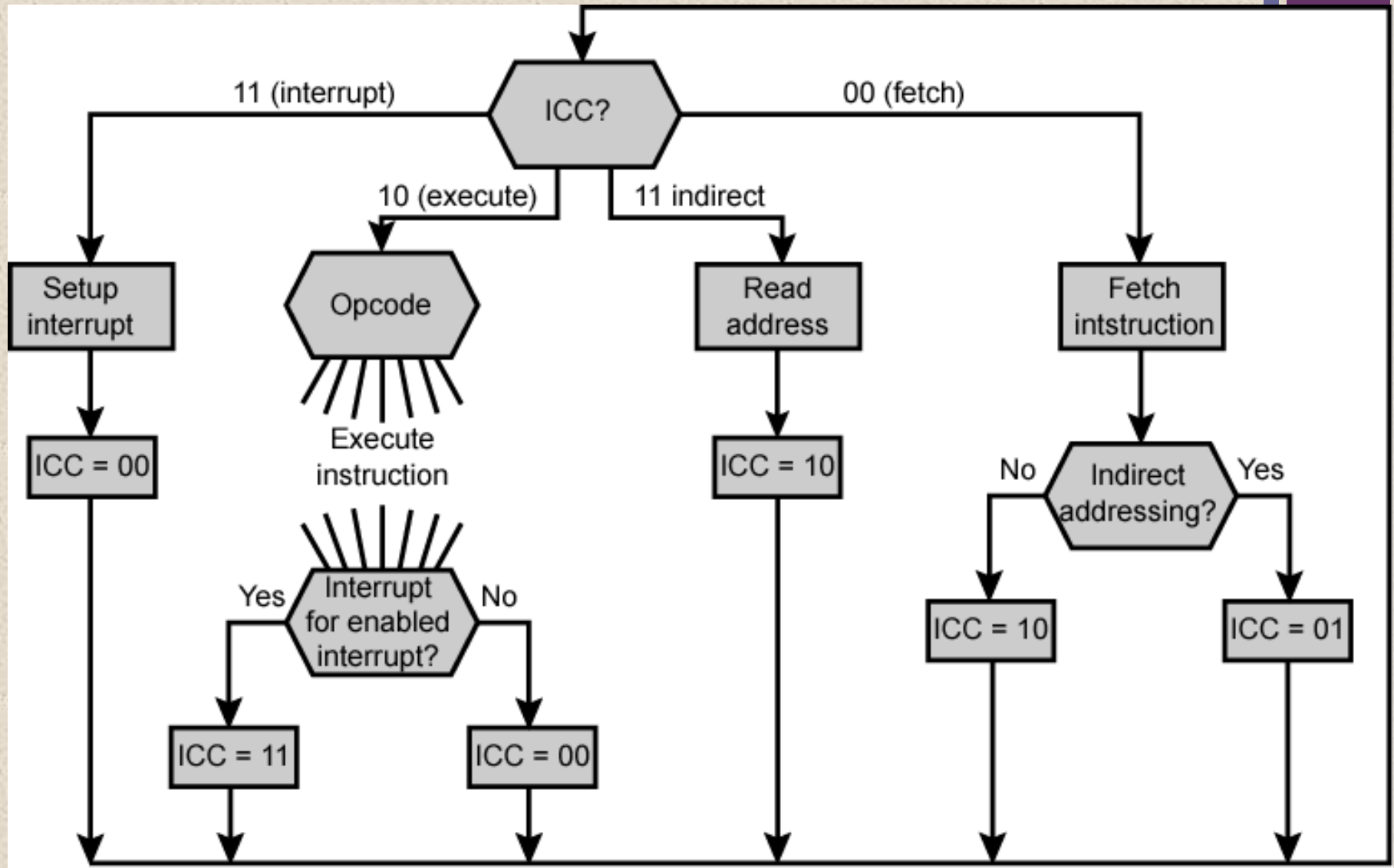
Instruction Cycle



- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
 - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
 - Instruction cycle code (ICC) designates which part of cycle processor is in
 - 00: Fetch
 - 01: Indirect
 - 10: Execute
 - 11: Interrupt



Flowchart for Instruction Cycle





Functional Requirements



- Define basic elements of processor
- Describe micro-operations processor performs
- Determine functions control unit must perform



Functions of Control Unit



- Sequencing
 - Causing the CPU to step through a series of micro-operations
- Execution
 - Causing the performance of each micro-op
- This is done using Control Signals

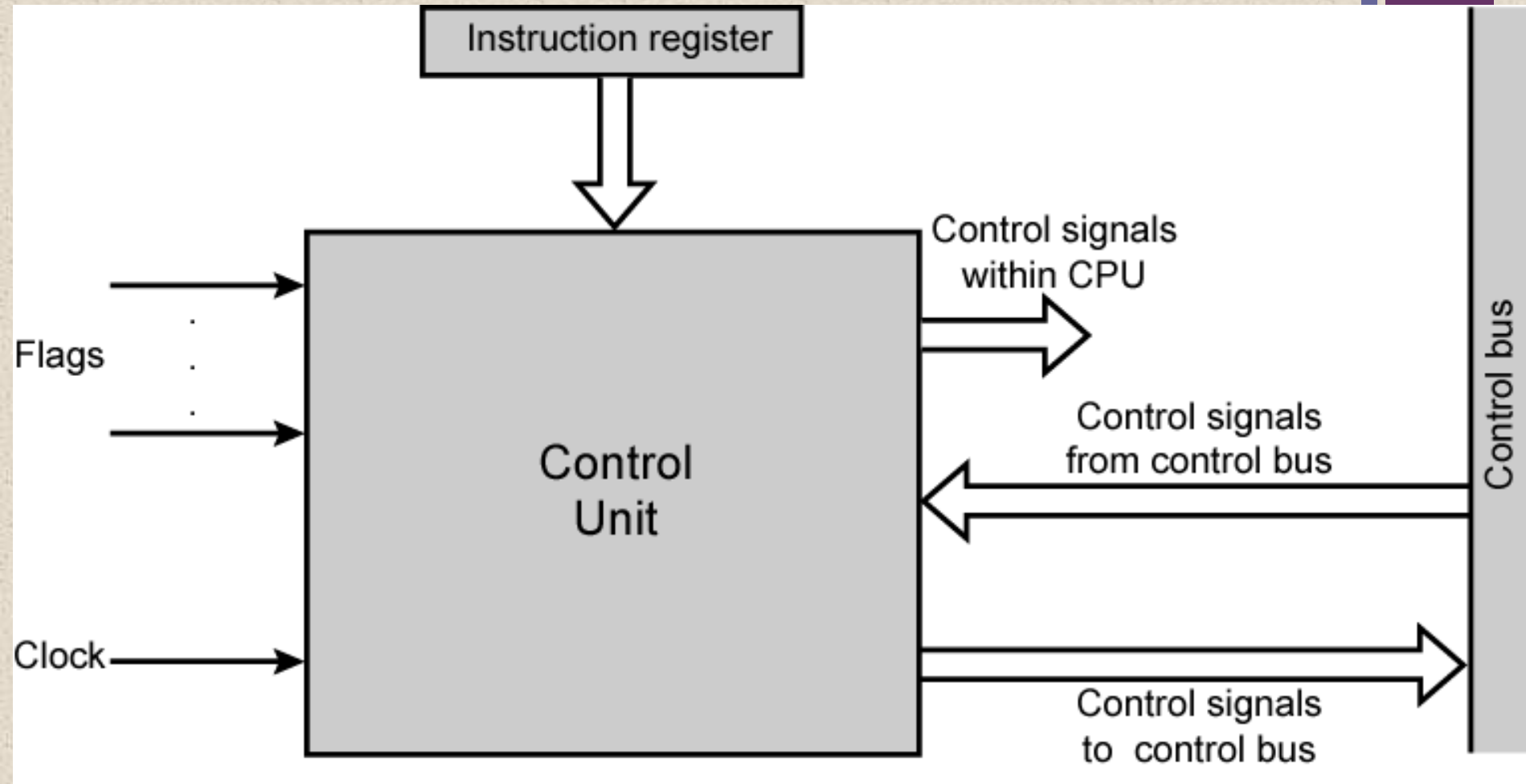


Control Signals



- Clock
 - One micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register
 - Op-code for current instruction
 - Determines which micro-instructions are performed
- Flags
 - State of CPU
 - Results of previous operations
- From control bus
 - Interrupts
 - Acknowledgements

+ Model of Control Unit





Control Signals - output



- Within CPU
 - Cause data movement
 - Activate specific functions
- Via control bus
 - To memory
 - To I/O modules



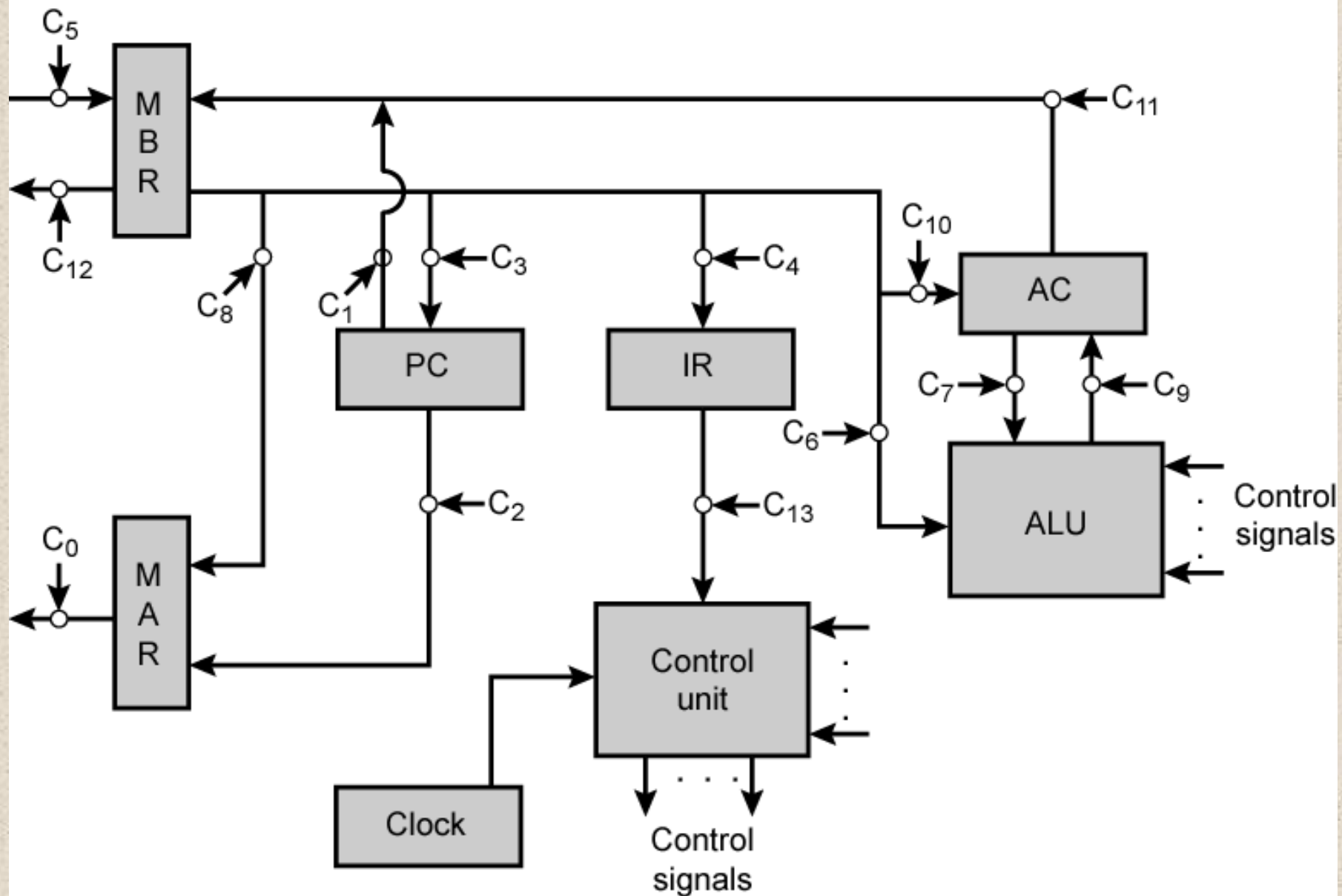
Example Control Signal Sequence - Fetch



- $MAR \leftarrow (PC)$
 - Control unit activates signal to open gates between PC and MAR
- $MBR \leftarrow (\text{memory})$
 - Open gates between MAR and address bus
 - Memory read control signal
 - Open gates between data bus and MBR



Data Paths and Control Signals





	Micro-operations	Active Control Signals
Fetch:	$t_1: \text{MAR} \leftarrow (\text{PC})$	C_2
	$t_2: \text{MBR} \leftarrow \text{Memory}$ $\text{PC} \leftarrow (\text{PC}) + 1$	C_5, C_R
	$t_3: \text{IR} \leftarrow (\text{MBR})$	C_4
Indirect:	$t_1: \text{MAR} \leftarrow (\text{IR}(\text{Address}))$	C_8
	$t_2: \text{MBR} \leftarrow \text{Memory}$	C_5, C_R
	$t_3: \text{IR}(\text{Address}) \leftarrow (\text{MBR}(\text{Address}))$	C_4
Interrupt:	$t_1: \text{MBR} \leftarrow (\text{PC})$	C_1
	$t_2: \text{MAR} \leftarrow \text{Save-address}$ $\text{PC} \leftarrow \text{Routine-address}$	
	$t_3: \text{Memory} \leftarrow (\text{MBR})$	C_{12}, C_W

C_R = Read control signal to system bus.

C_W = Write control signal to system bus.

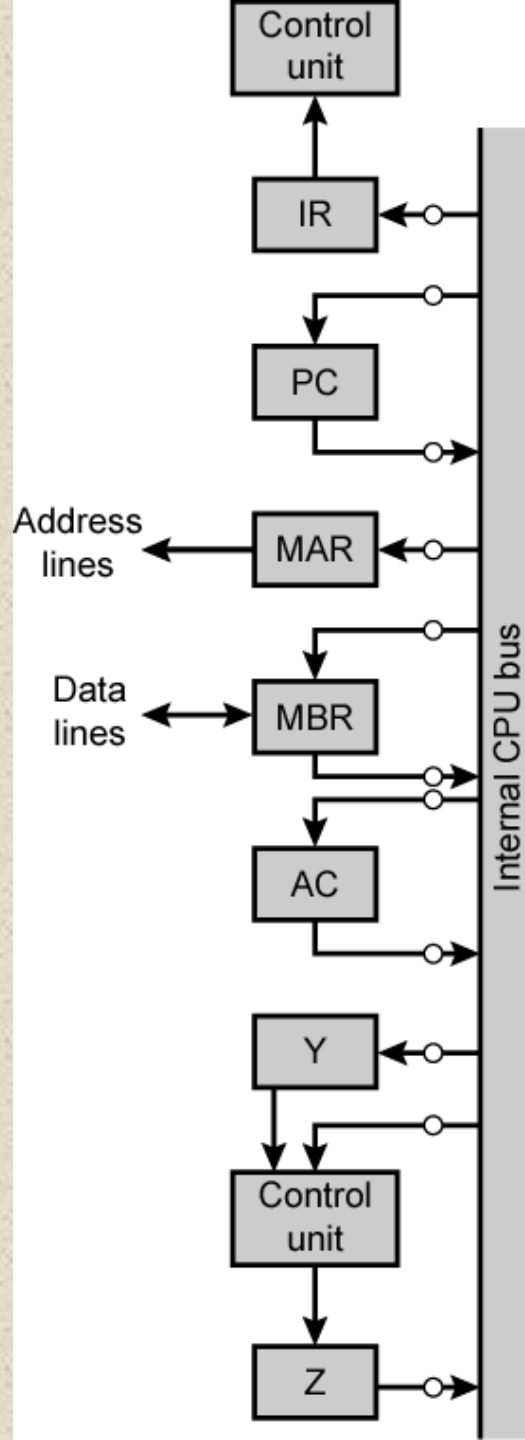
+ Internal Organization



- Usually a single internal bus
- Gates control movement of data onto and off the bus
- Control signals control data transfer to and from external systems bus
- Temporary registers needed for proper operation of ALU

+

CPU with Internal Bus





Hardwired Implementation (1)



- Control unit inputs
- Flags and control bus
 - Each bit means something
- Instruction register
 - Op-code causes different control signals for each different instruction
 - Unique logic for each op-code
 - Decoder takes encoded input and produces single output
 - n binary inputs and 2^n outputs



Hardwired Implementation (2)

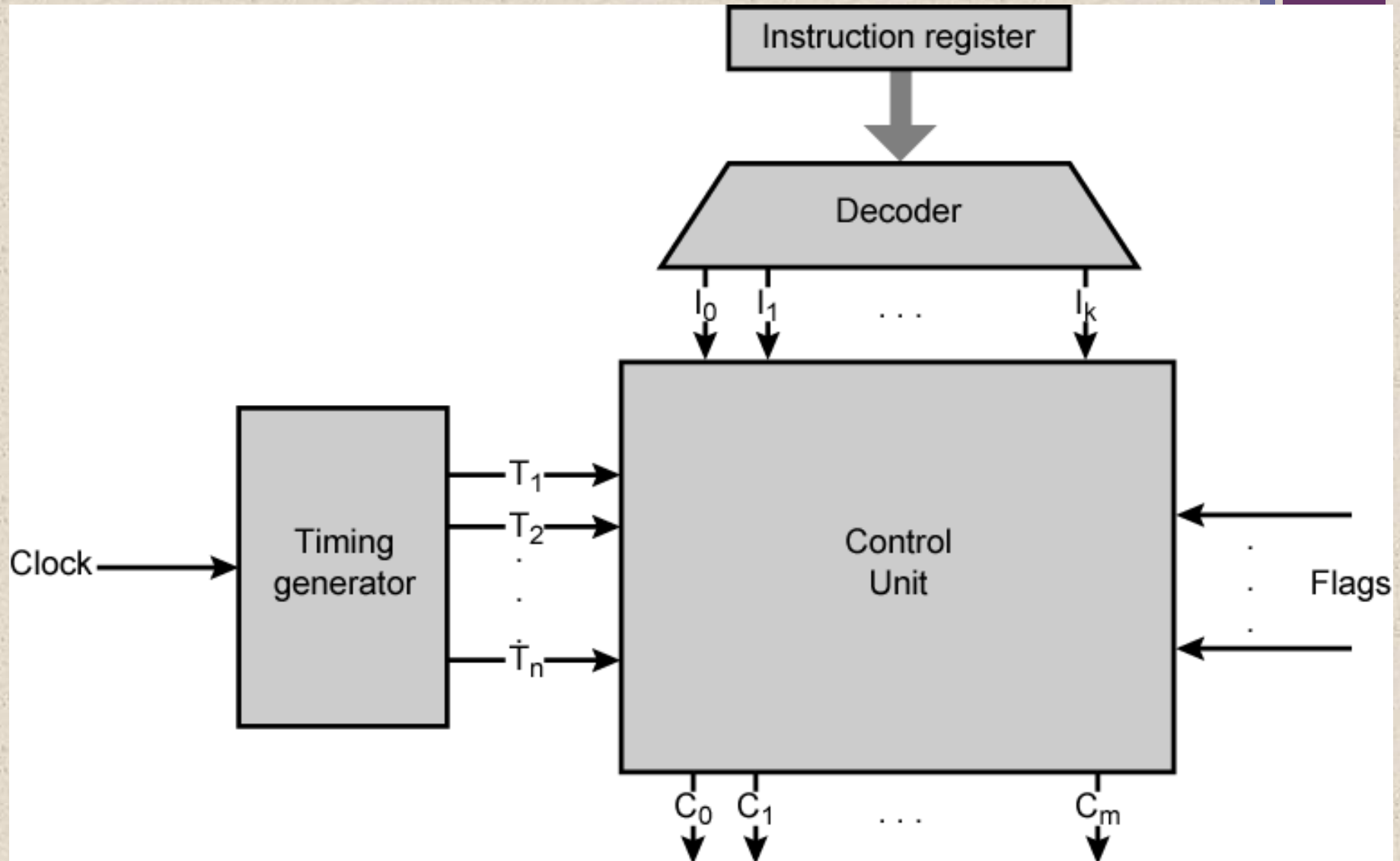


■ Clock

- Repetitive sequence of pulses
- Useful for measuring duration of micro-ops
- Must be long enough to allow signal propagation
- Different control signals at different times within instruction cycle
- Need a counter with different control signals for t1, t2 etc.

+

Control Unit with Decoded Inputs





Problems With Hard Wired Designs



- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions



Micro-programmed Control



- Use sequences of instructions to control complex operations
Called micro-programming or firmware



Implementation



- All the control unit does is generate a set of control signals
- Each control signal is on or off
- Represent each control signal by a bit
- Have a control word for each micro-operation
- Have a sequence of control words for each machine code instruction
- Add an address to specify the next micro-instruction, depending on conditions

+ Micro-program Word Length



- Based on 3 factors
 - Maximum number of simultaneous micro-operations supported
 - The way control information is represented or encoded
 - The way in which the next micro-instruction address is specified



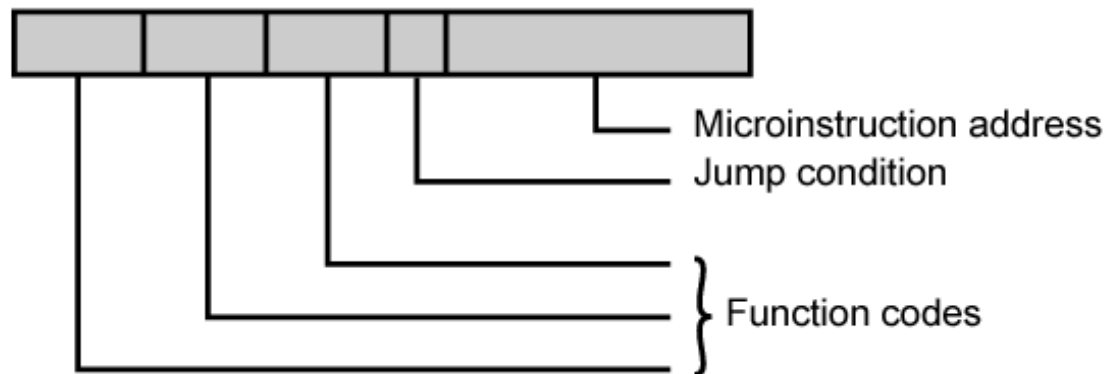
Micro-instruction Types



- Each micro-instruction specifies single (or few) micro-operations to be performed
 - (*vertical* micro-programming)
- Each micro-instruction specifies many different micro-operations to be performed in parallel
 - (*horizontal* micro-programming)

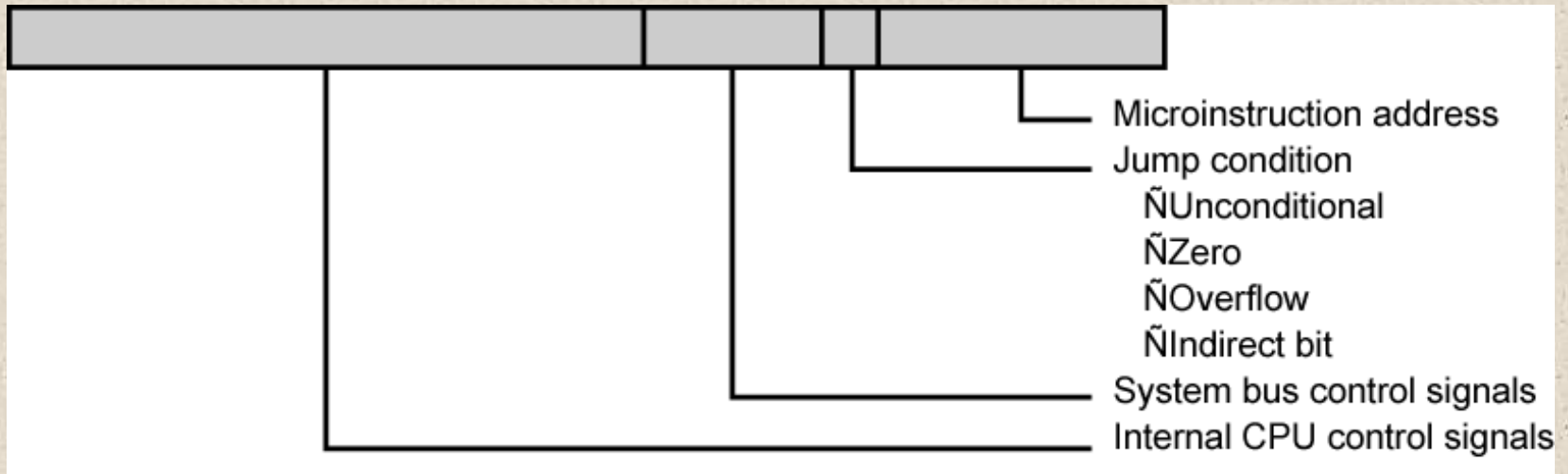
+ Vertical Micro-programming

- Width is narrow
- n control signals encoded into $\log_2 n$ bits
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated



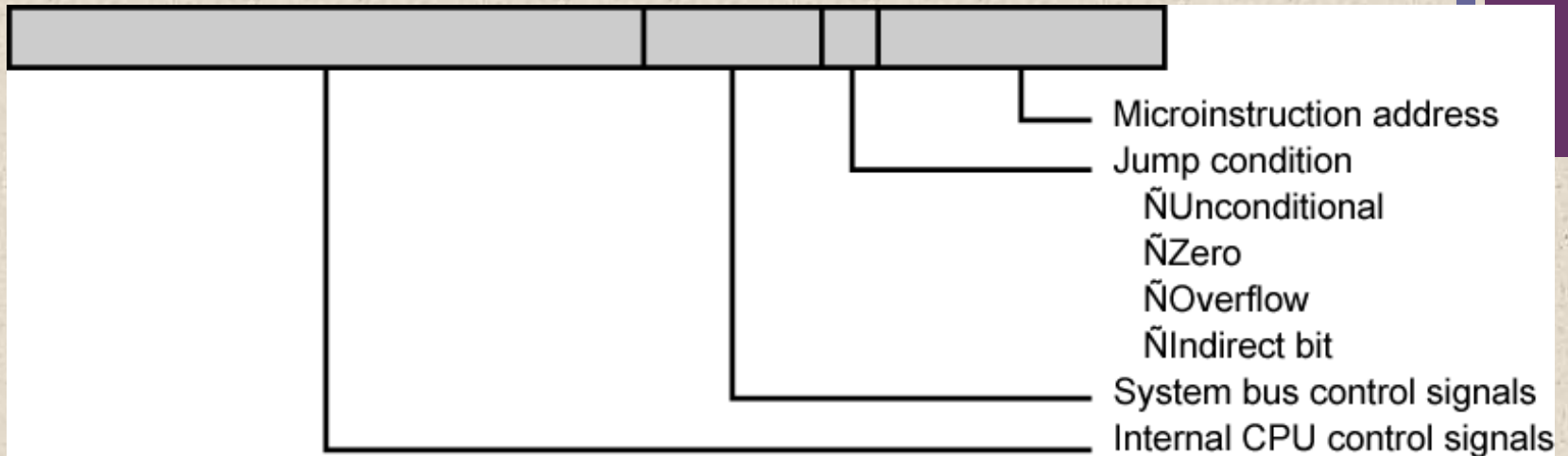
+ Horizontal Micro-programming

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information

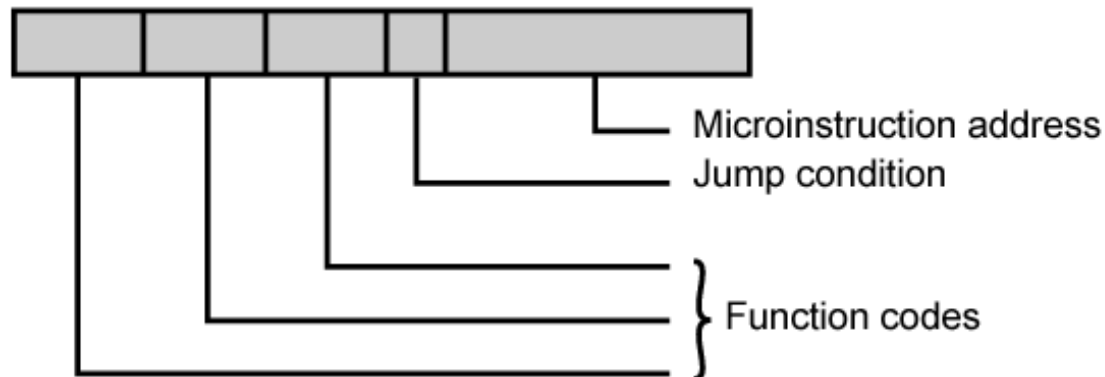




Typical Microinstruction Formats



(a) Horizontal microinstruction



(b) Vertical microinstruction



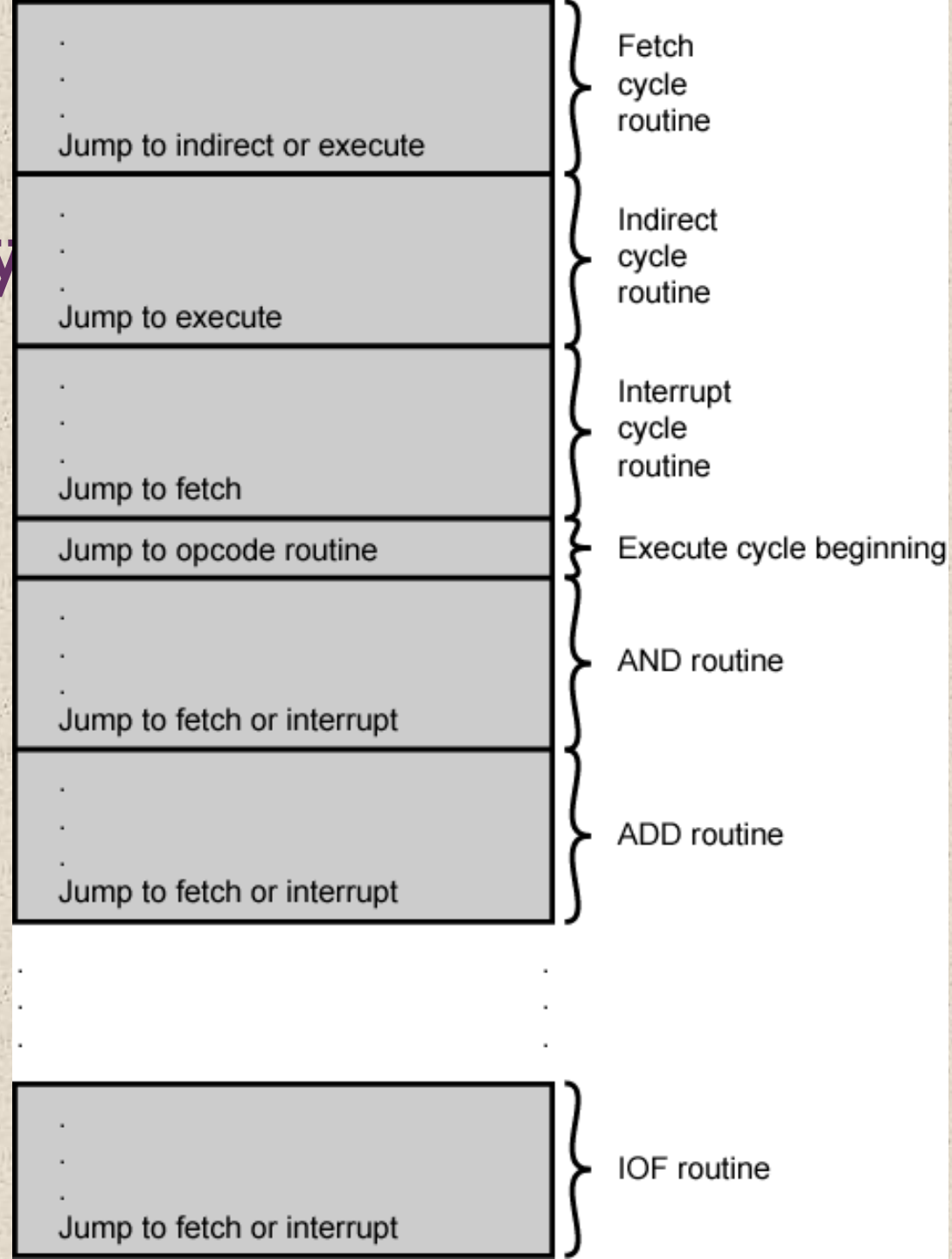
Compromise



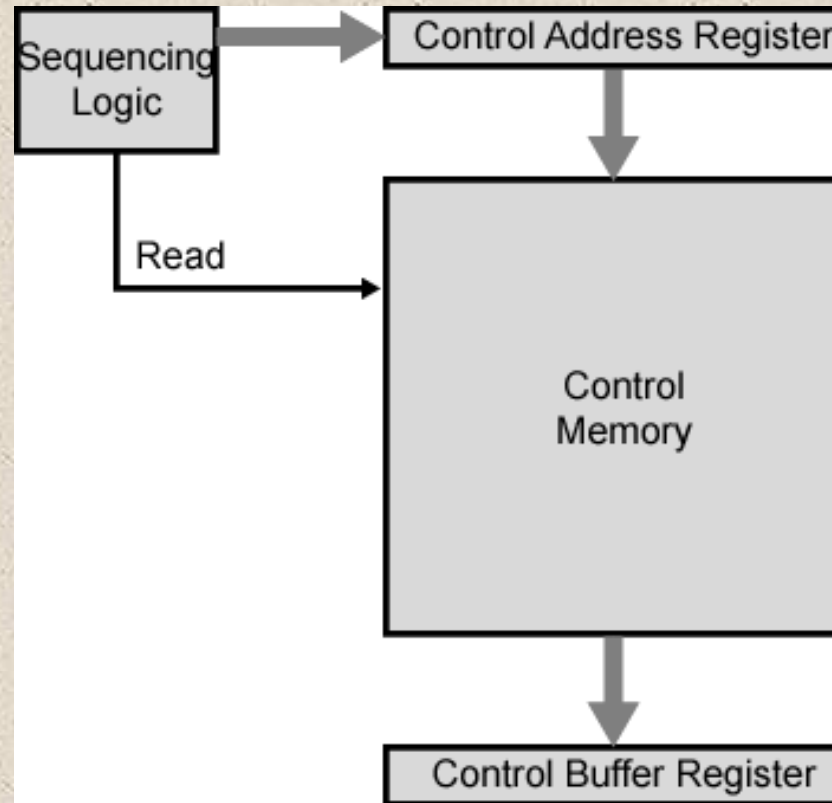
- Divide control signals into disjoint groups
- Implement each group as separate field in memory word
- Supports reasonable levels of parallelism without too much complexity

+

Organization of Control Memory



+Control Unit Microarchitecture



+ Control Unit Function



- Sequence logic unit issues read command
- Word specified in control address register is read into control buffer register
- Control buffer register contents generates control signals and next address information
- Sequence logic loads new address into control buffer register based on next address information from control buffer register and ALU flags

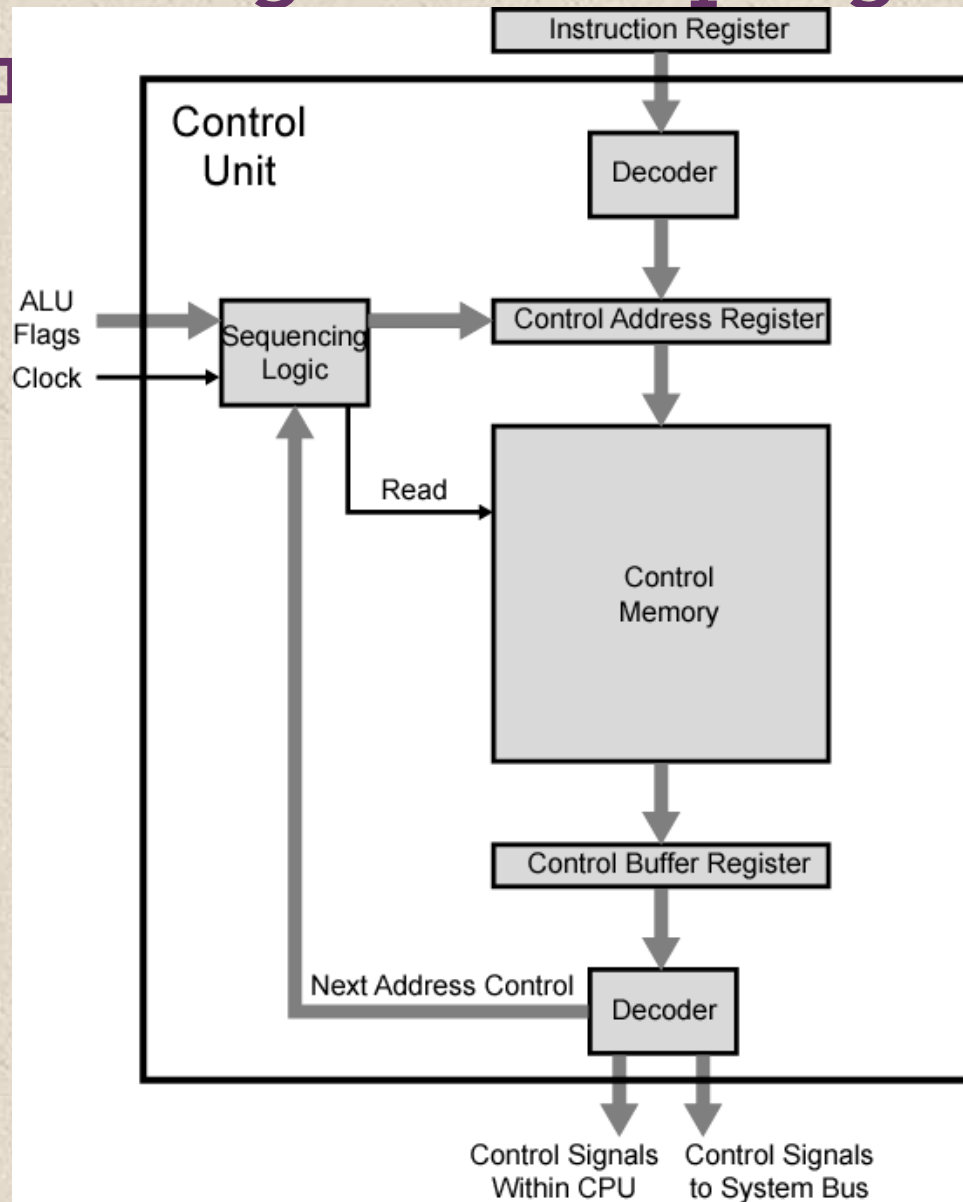
+ Next Address Decision



- Depending on ALU flags and control buffer register
 - Get next instruction
 - Add 1 to control address register
 - Jump to new routine based on jump microinstruction
 - Load address field of control buffer register into control address register
 - Jump to machine instruction routine
 - Load control address register based on opcode in IR

+

Functioning of Microprogrammed Control



+ Advantages and Disadvantages of Microprogramming



- Simplifies design of control unit
 - Cheaper
 - Less error-prone
- Slower



Tasks Done By Microprogrammed Control Unit



- Microinstruction sequencing
- Microinstruction execution
- Must consider both together

+ Design Considerations



- Size of microinstructions
- Address generation time
 - Determined by instruction register
 - Once per cycle, after instruction is fetched
 - Next sequential address
 - Common in most designed
 - Branches
 - Both conditional and unconditional

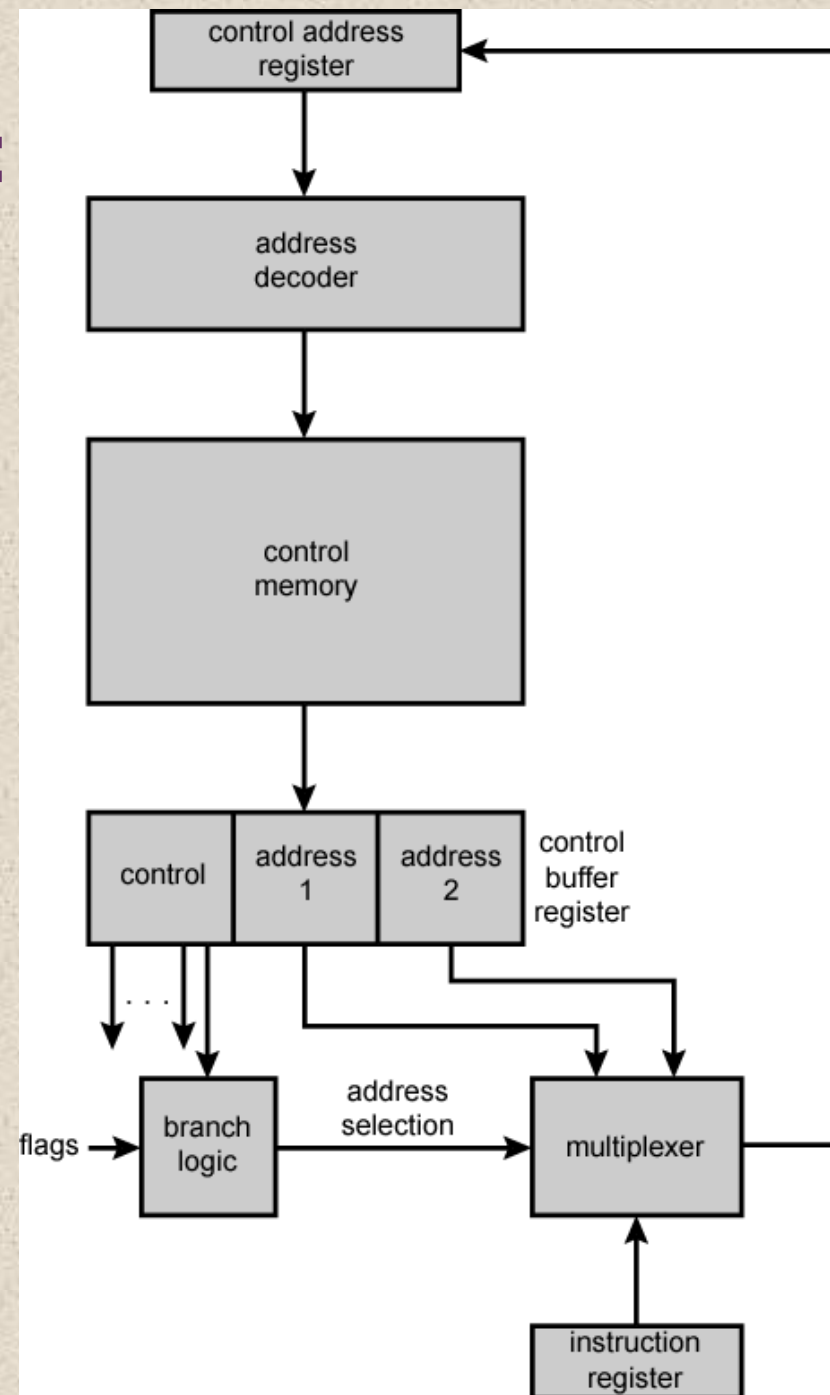


Sequencing Techniques



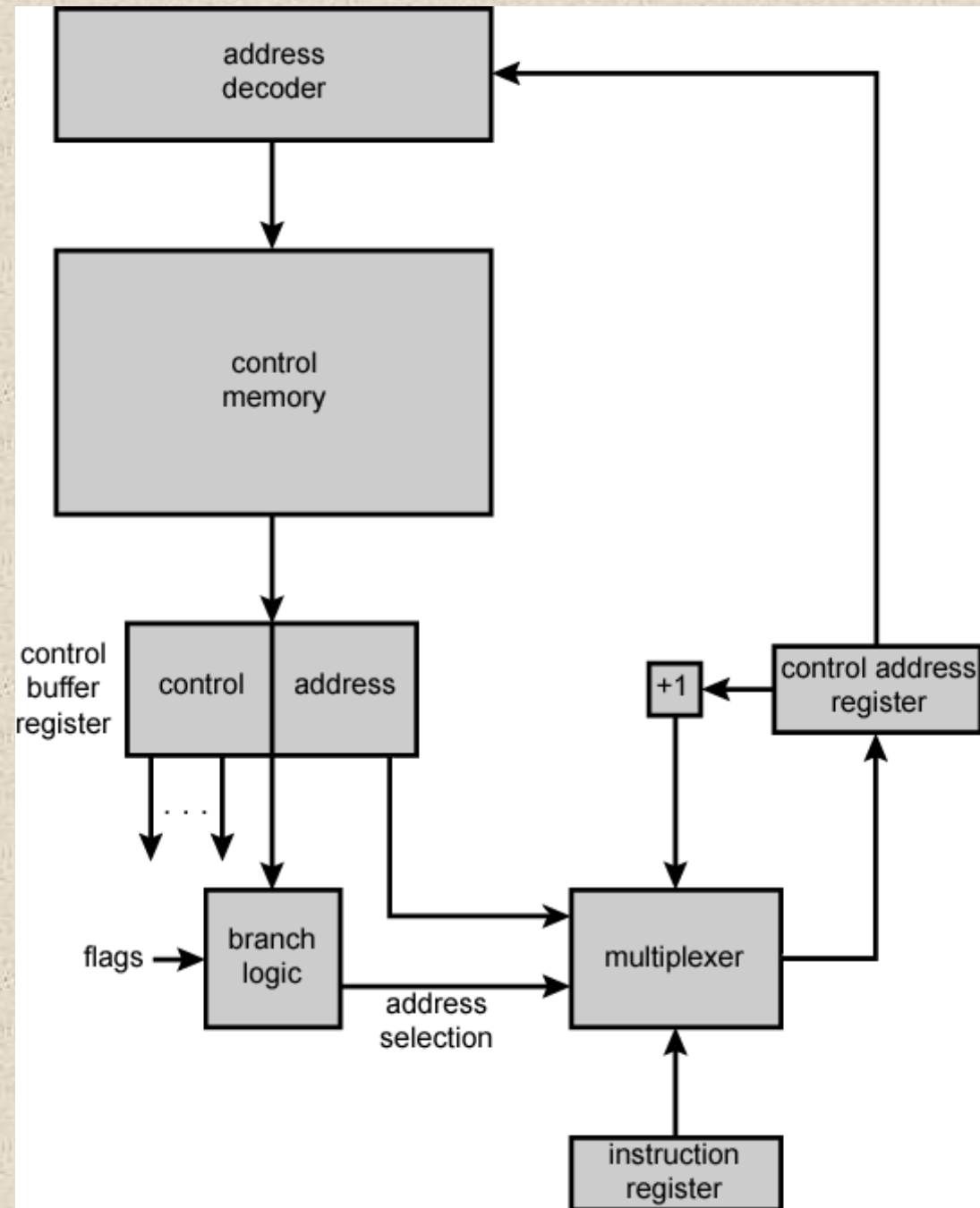
- Based on current microinstruction, condition flags, contents of IR, control memory address must be generated
- Based on format of address information
 - Two address fields
 - Single address field
 - Variable format

+ Branch Control Logic: Two Address Fields



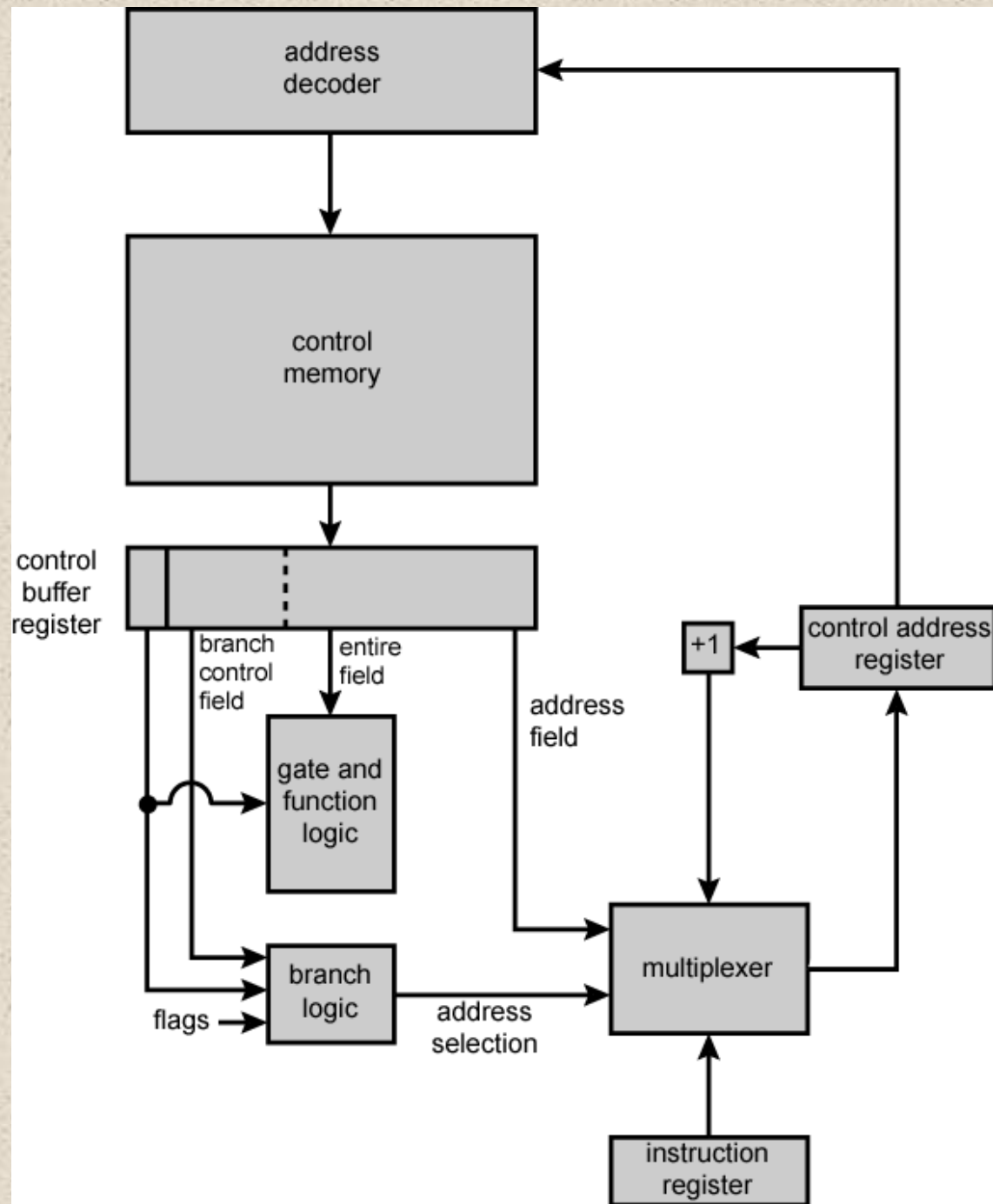
+

Branch Control Logic: Single Address Field



+

Branch Control Logic: Variable Format



Address Generation

Explicit	Implicit
Two-field	Mapping
Unconditional Branch	Addition
Conditional branch	Residual control



Execution



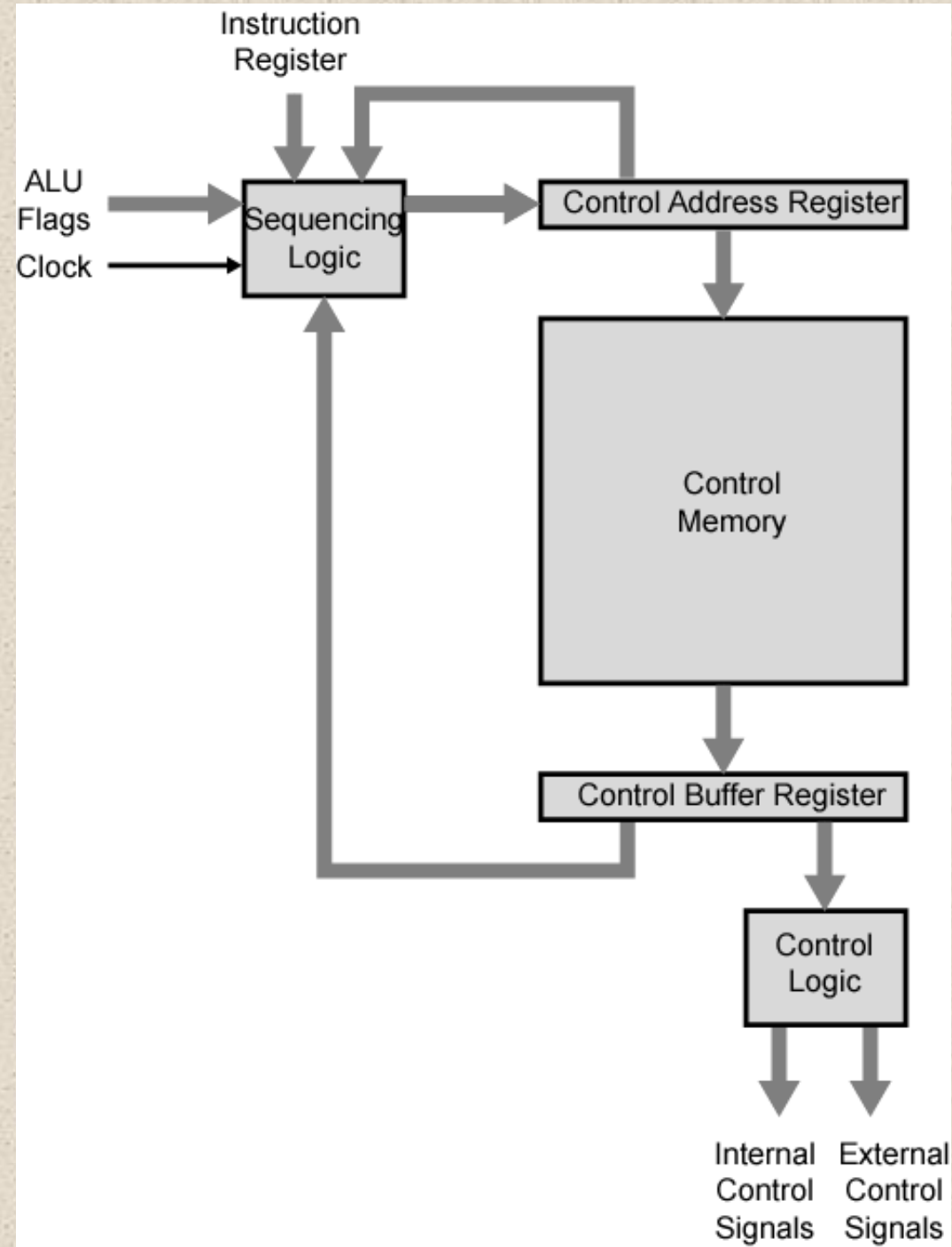
- The cycle is the basic event
- Each cycle is made up of two events
 - Fetch
 - Determined by generation of microinstruction address
 - Execute

+ Execute



- Effect is to generate control signals
- Some control points internal to processor
- Rest go to external control bus or other interface

+ Control Unit Organization





A Taxonomy of Microinstructions



- Vertical/horizontal
- Packed/unpacked
- Hard/soft microprogramming
- Direct/indirect encoding



Improvements over Wilkes



- Wilkes had each bit directly produced a control signal or directly produced one bit of next address
- More complex address sequencing schemes,
 - using fewer microinstruction bits, are possible
- Require more complex sequencing logic module
- Control word bits can be saved by encoding and subsequently decoding control information



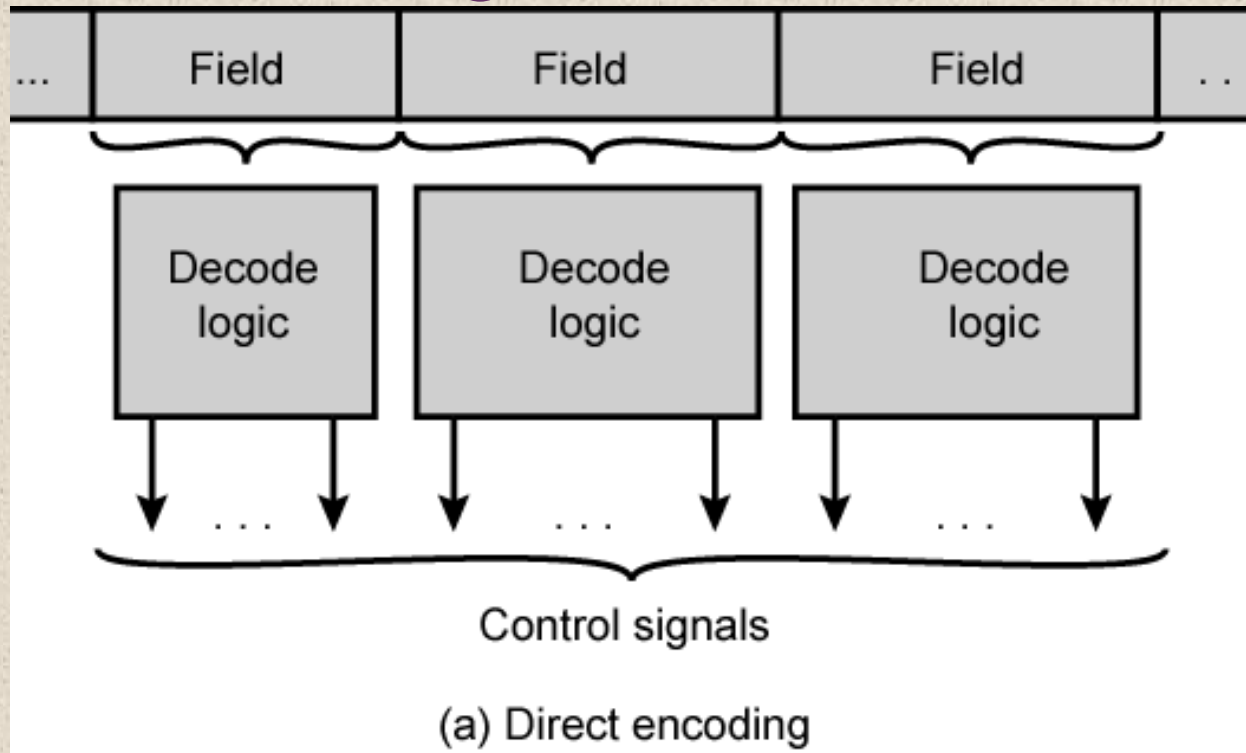
Specific Encoding Techniques



- Microinstruction organized as set of fields
- Each field contains code
- Activates one or more control signals
- Organize format into independent fields
 - Field depicts set of actions (pattern of control signals)
 - Actions from different fields can occur simultaneously
- Alternative actions that can be specified by a field are mutually exclusive
 - Only one action specified for field could occur at a time

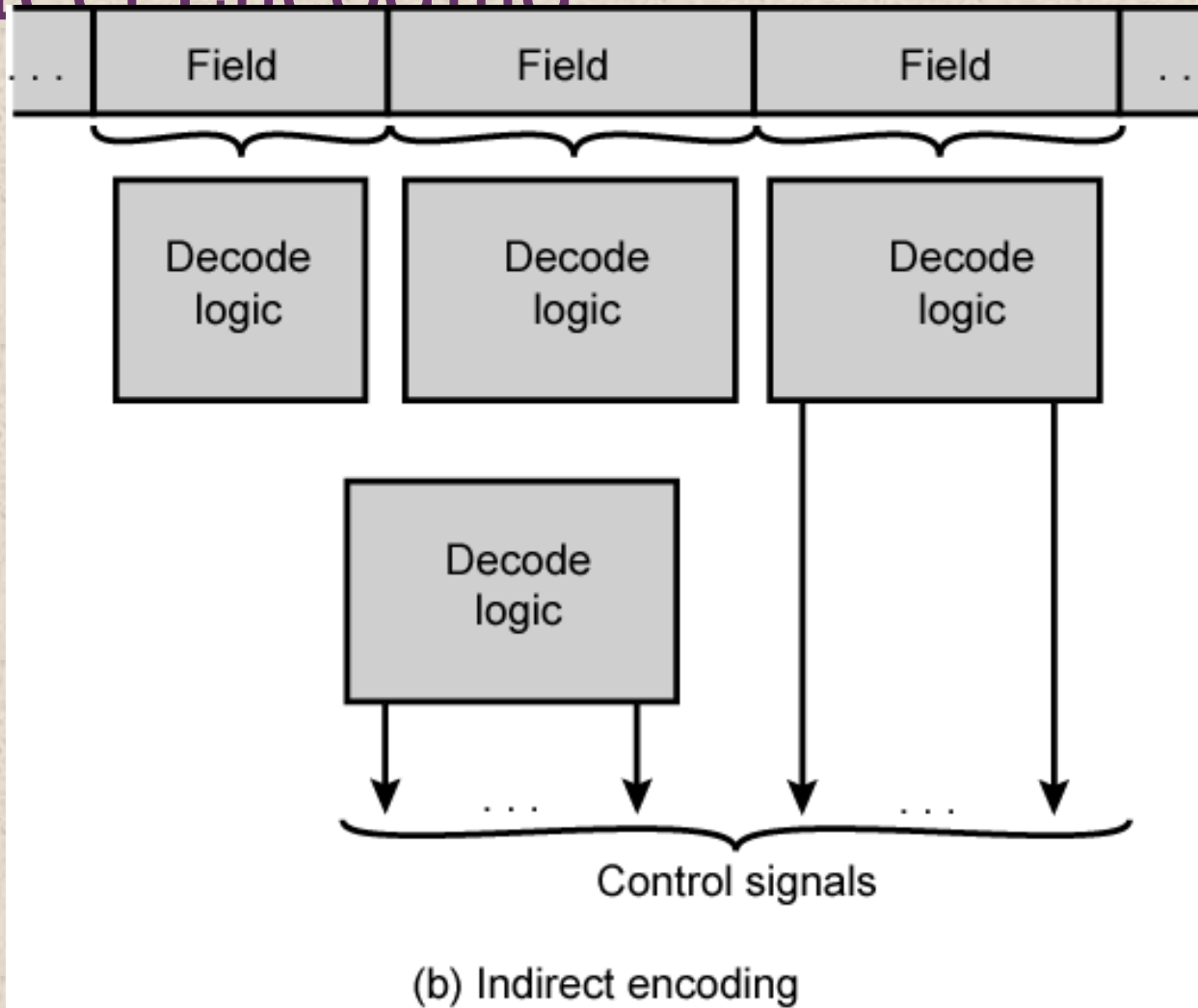
+ Microinstruction Encoding

Direct Encoding



+ Microinstruction Encoding

Indirect Encoding





Parallel Processing

Parallel Processing

It is a mode of operation in which a process is split into parts, which are executed simultaneously on different processors attached to the same computer.

Multiple Processor Organization (Flynn's Classification)

- Single instruction, single data stream - SISD
- Single instruction, multiple data stream - SIMD
- Multiple instruction, single data stream - MISD
- Multiple instruction, multiple data stream - MIMD

Single Instruction, Single Data Stream - SISD

- Single processor executes a single instruction stream to operate on data stored in a single memory
- Uniprocessors fall into this category

Single Instruction, Multiple Data Stream

- SIMD

- Single machine instruction
- Controls simultaneous execution
- Number of processing elements
- Lockstep basis
- Each processing element has associated data memory
- Each instruction executed on different set of data by different processors
- Vector and array processors

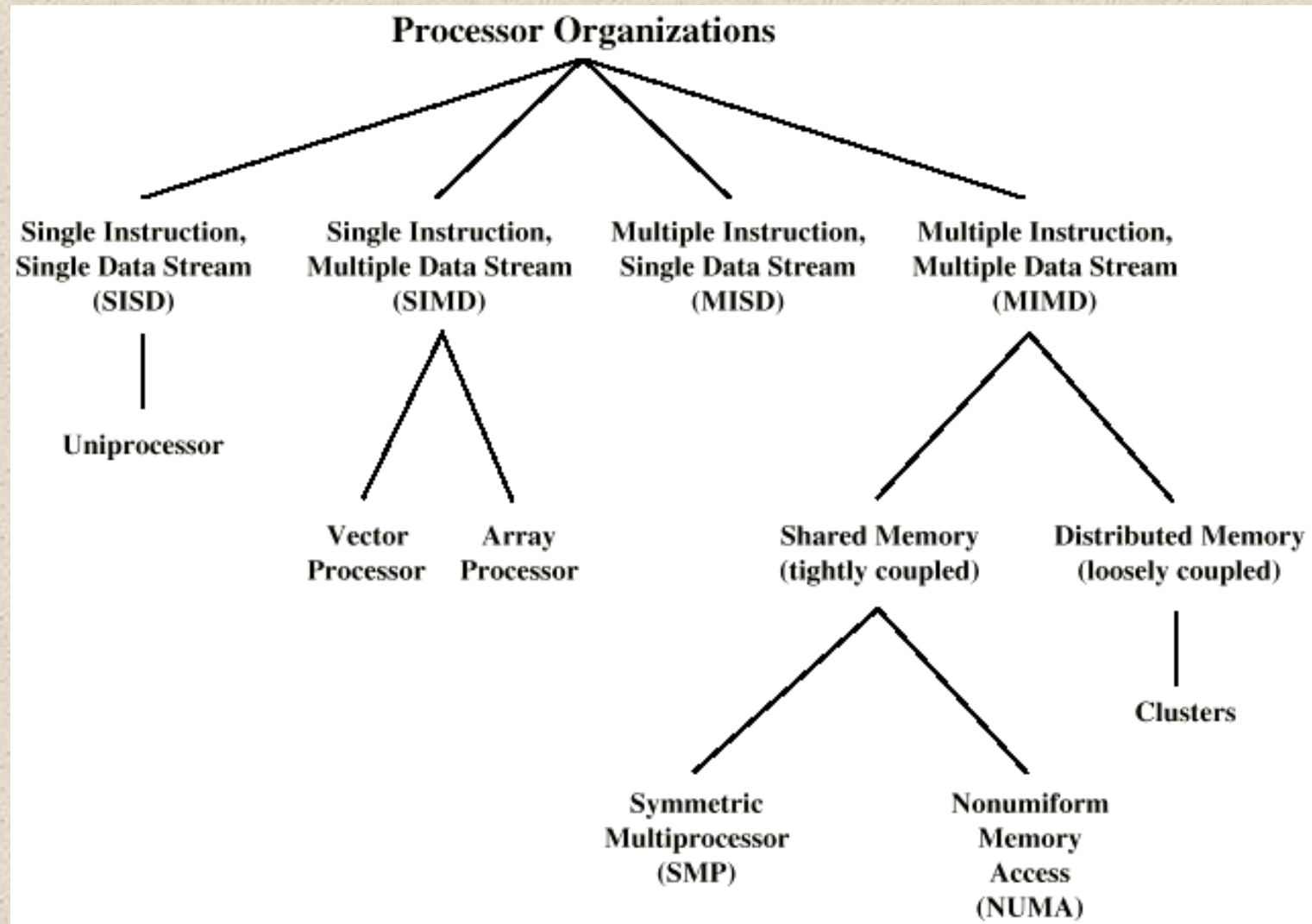
Multiple Instruction, Single Data Stream - MISD

- Sequence of data
- Transmitted to set of processors
- Each processor executes different instruction sequence
- Never been implemented

Multiple Instruction, Multiple Data Stream- MIMD

- Set of processors
- Simultaneously execute different instruction sequences
- Different sets of data
- SMPs(Symmetric Multiprocessor), clusters and NUMA (Non Uniform Memory Access)systems

Taxonomy of Parallel Processor Architectures



MIMD - Overview

- General purpose processors
- Each can process all instructions necessary
- Further classified by method of processor communication

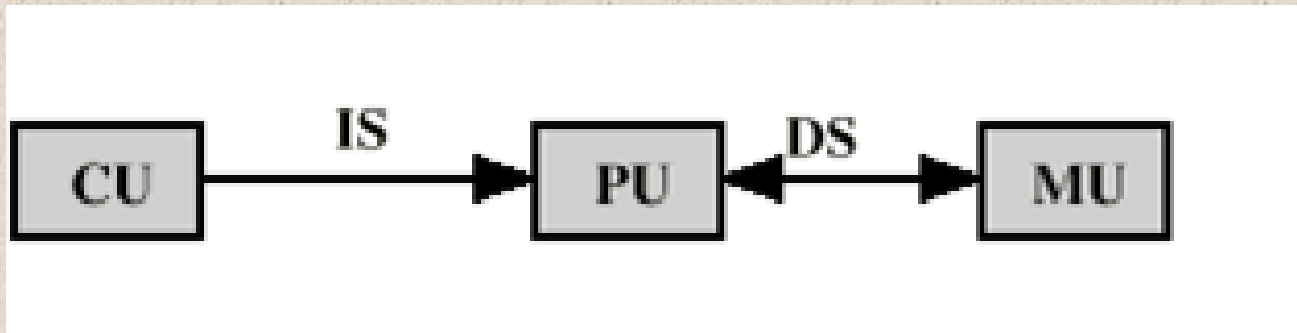
Tightly Coupled - SMP

- Processors share memory
- Communicate via that shared memory
- Symmetric Multiprocessor (SMP)
 - Share single memory or pool
 - Shared bus to access memory
 - Memory access time to given area of memory is approximately the same for each processor

Loosely Coupled - Clusters

- Collection of independent uniprocessors or SMPs
- Interconnected to form a cluster
- Communication via fixed path or network connections

Parallel Organizations - SISD



CU -Control Unit

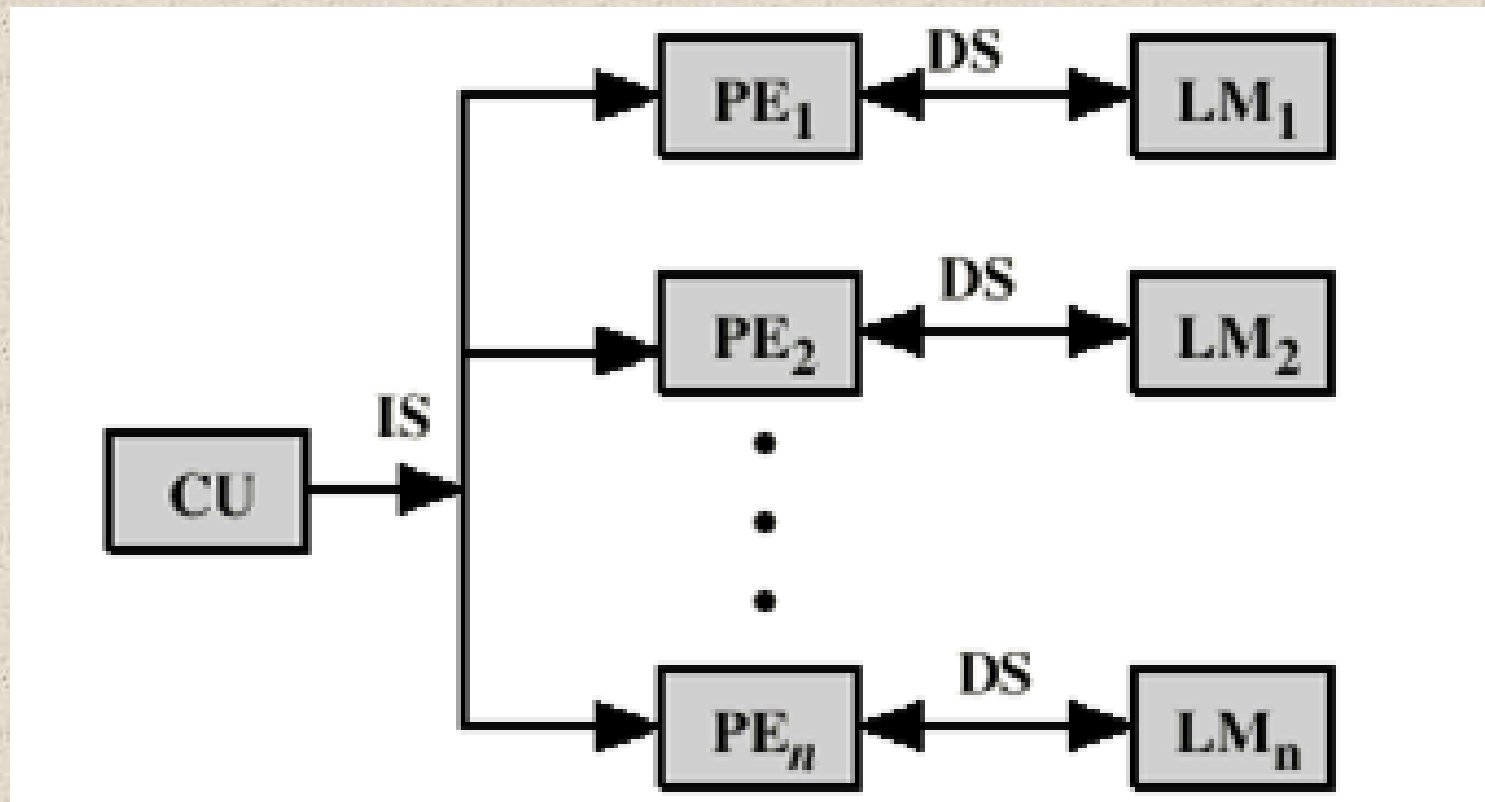
IS -Instruction Stream

PU -Processing Unit

DS- Data Stream

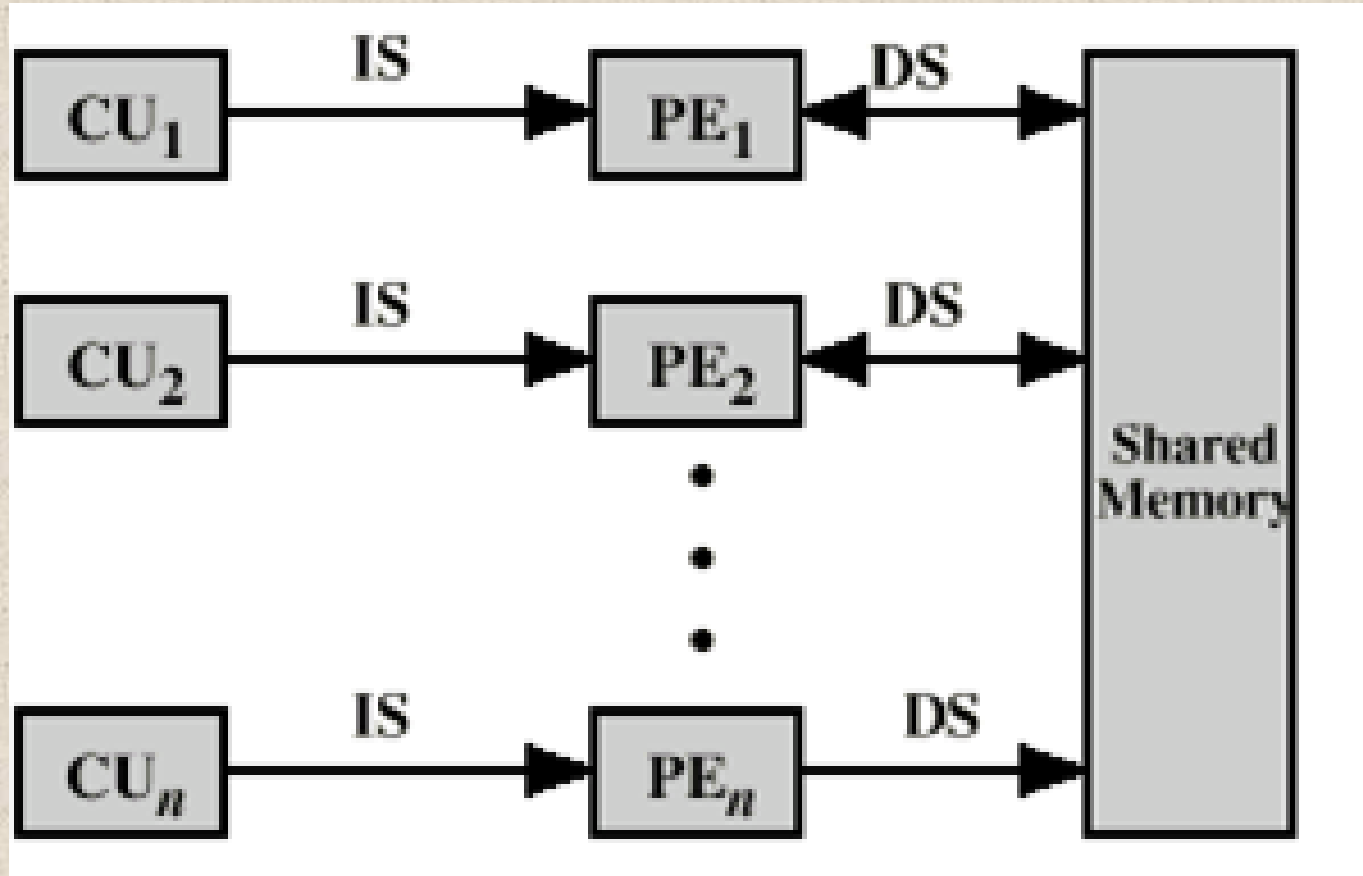
MU -Memory Unit

Parallel Organizations - SIMD



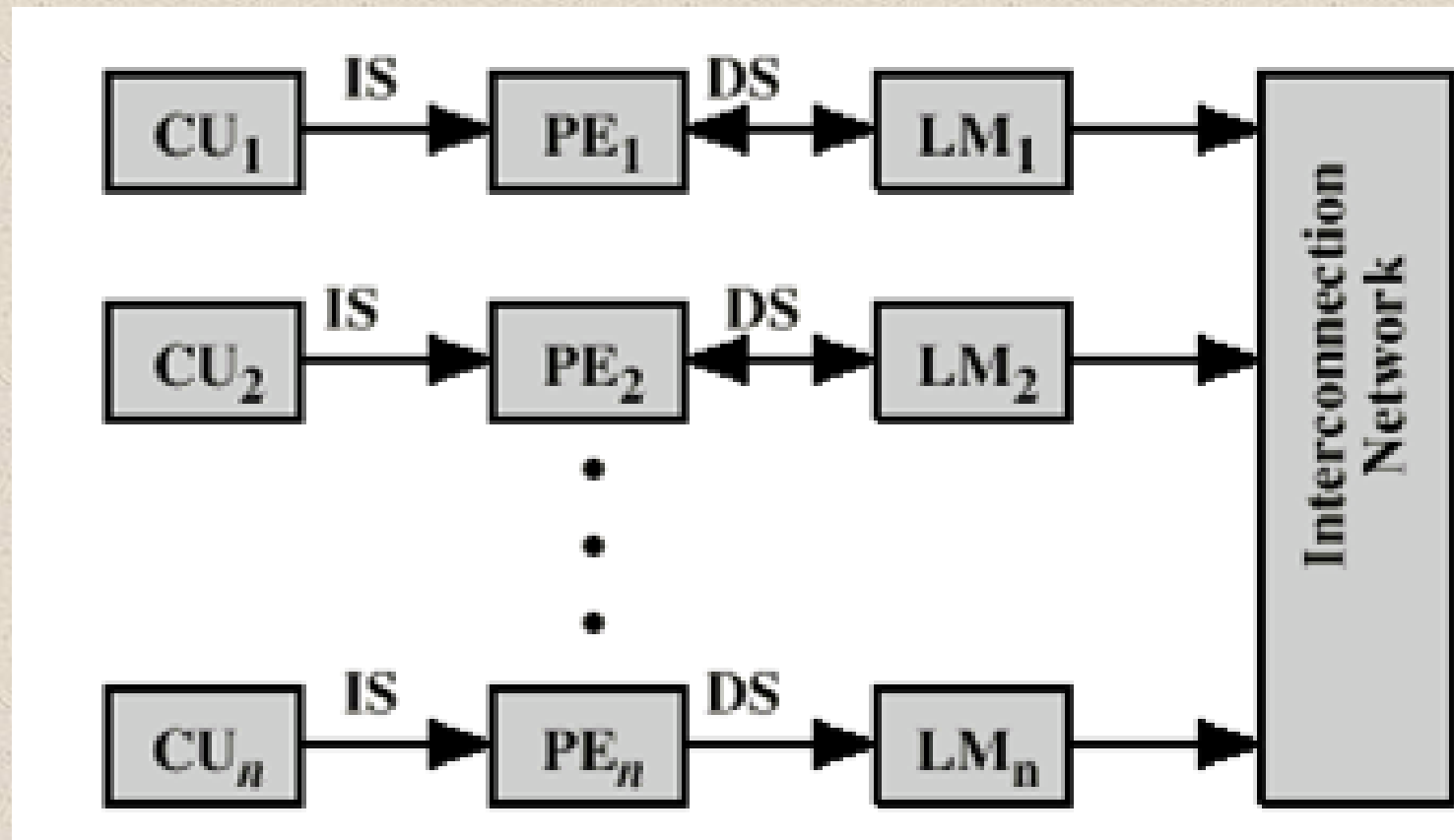
LM - Local Memory

Parallel Organizations - MIMD Shared Memory



Parallel Organizations - MIMD

Distributed Memory



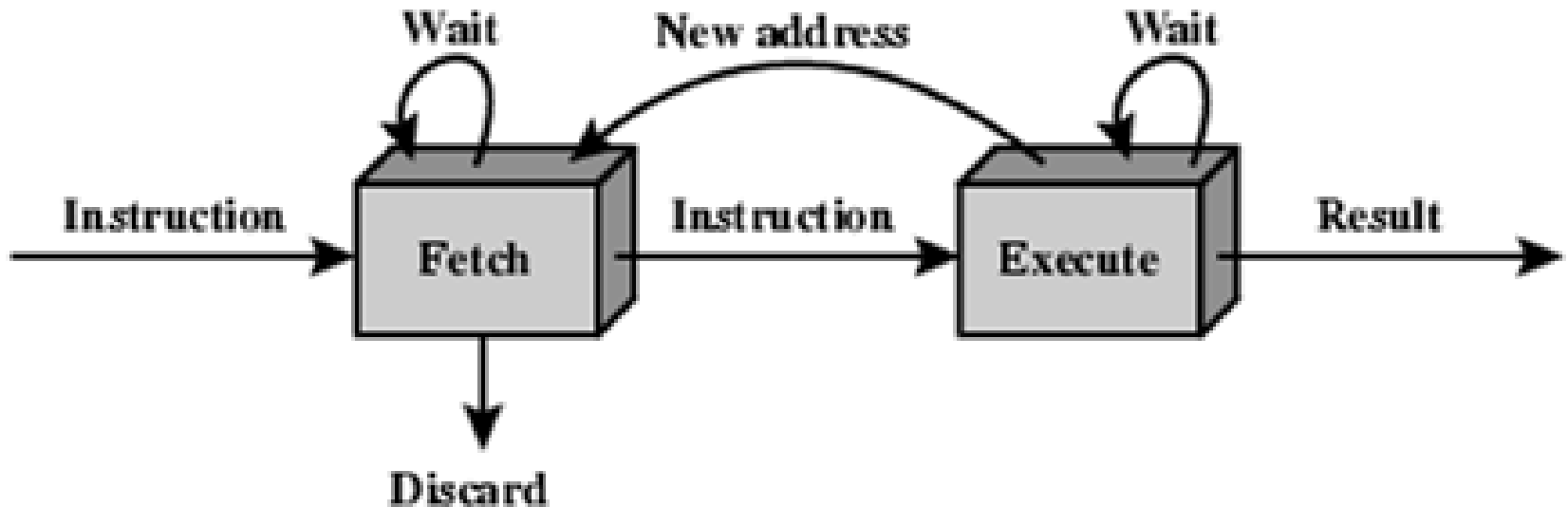
Pipelining

- 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



Comparison of Superscalar and Superpipeline Approaches

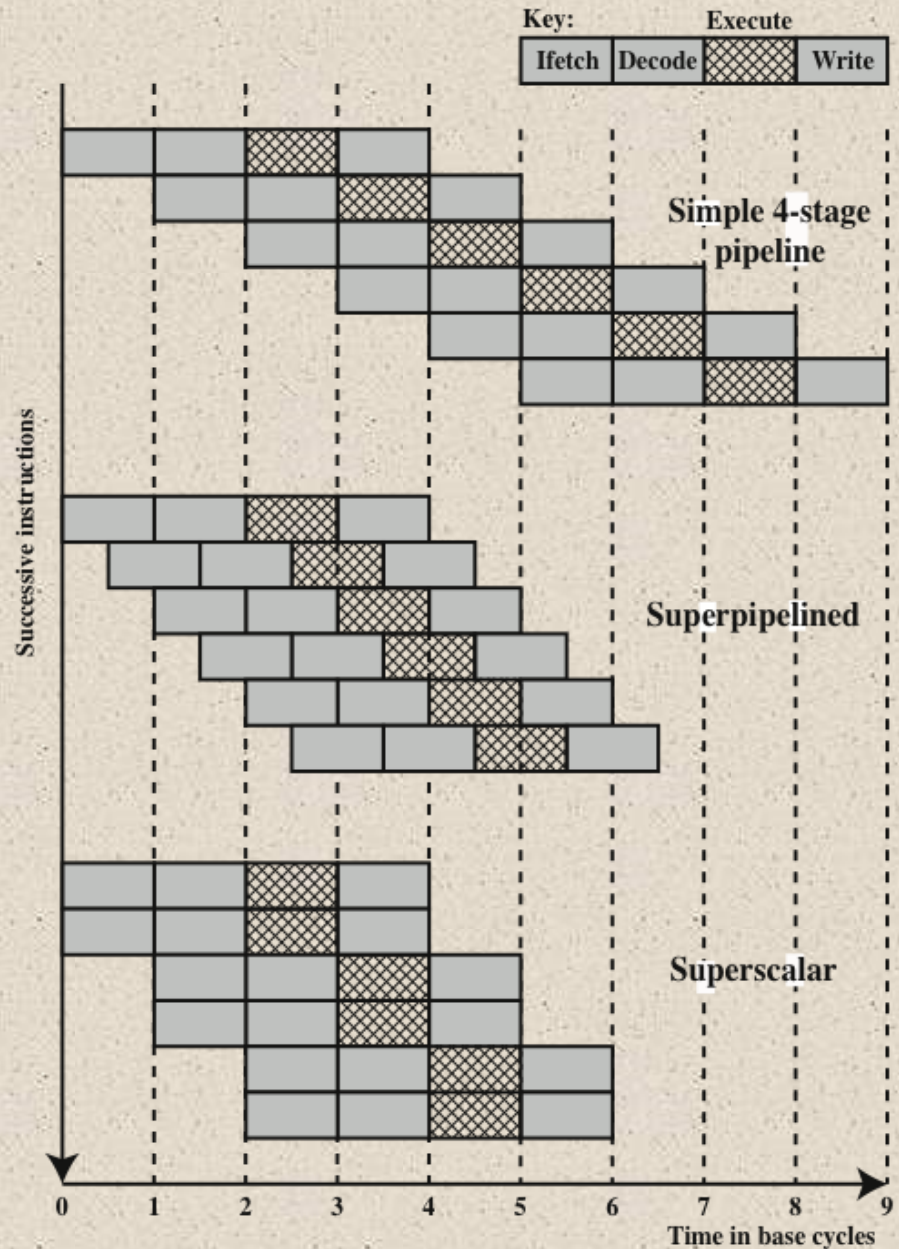


Figure 16.2 Comparison of Superscalar and Superpipeline Approaches

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
- 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

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

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 clock 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		
I3				FI			DI	FO	EI	WO	
I4							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

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

The Effect of a Conditional Branch on Instruction Pipeline Operation

