Chapter 12

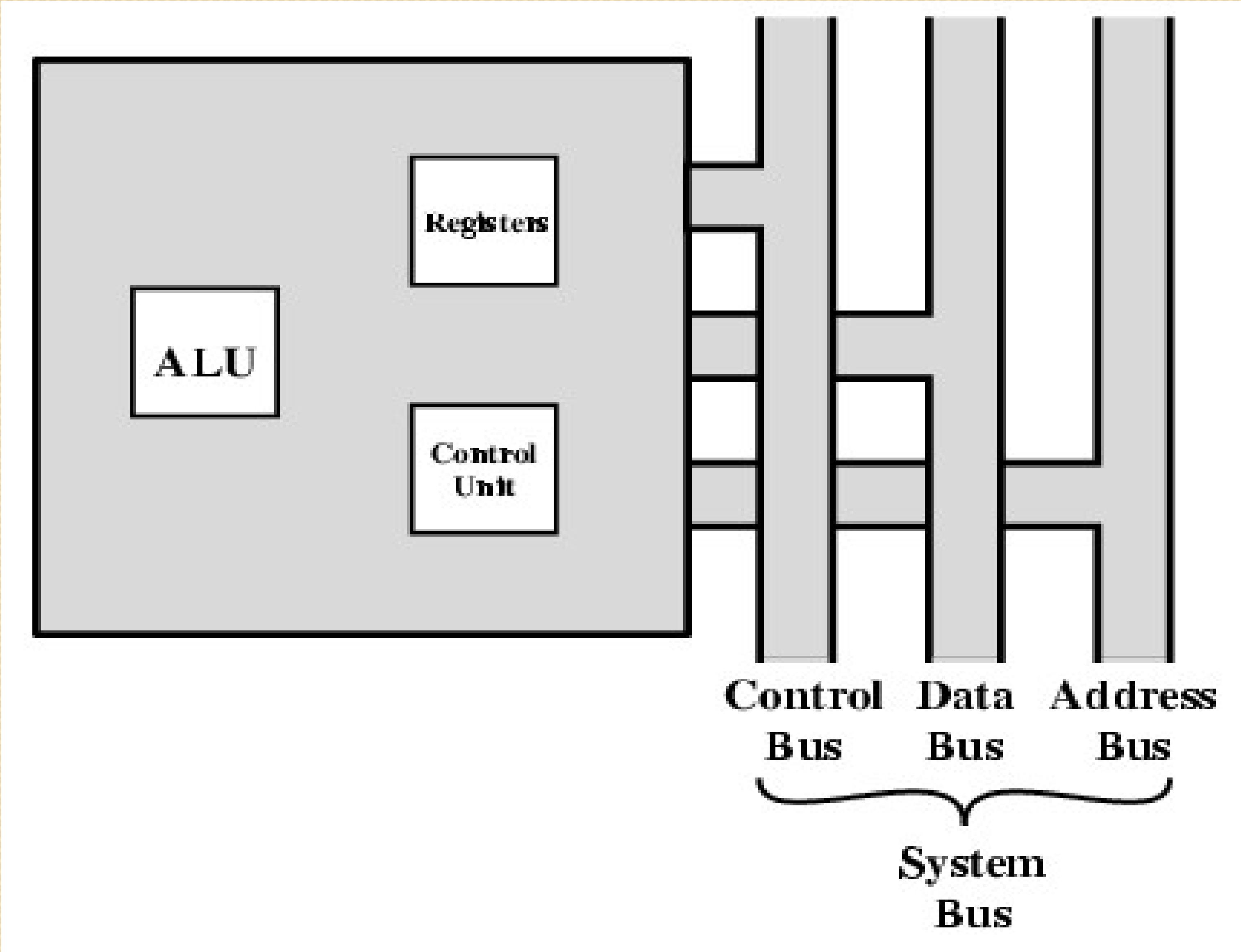
Processor Structure and Function

Module 3- Central Processing Unit	
3.1	CPU architecture, Register organization, Instruction formats and addressing modes(Intel processor).,Basic instruction cycle. Control unit Operation ,Micro operations : Fetch, Indirect, Interrupt , Execute cycle Control of the processor, Functioning of micro programmed control unit, Micro instruction Execution and Sequencing, Applications of Micro programming
3.2	RISC v/s CISC processors, RISC and CISC Architecture, RISC pipelining, Case study on SPARC

CPU Structure

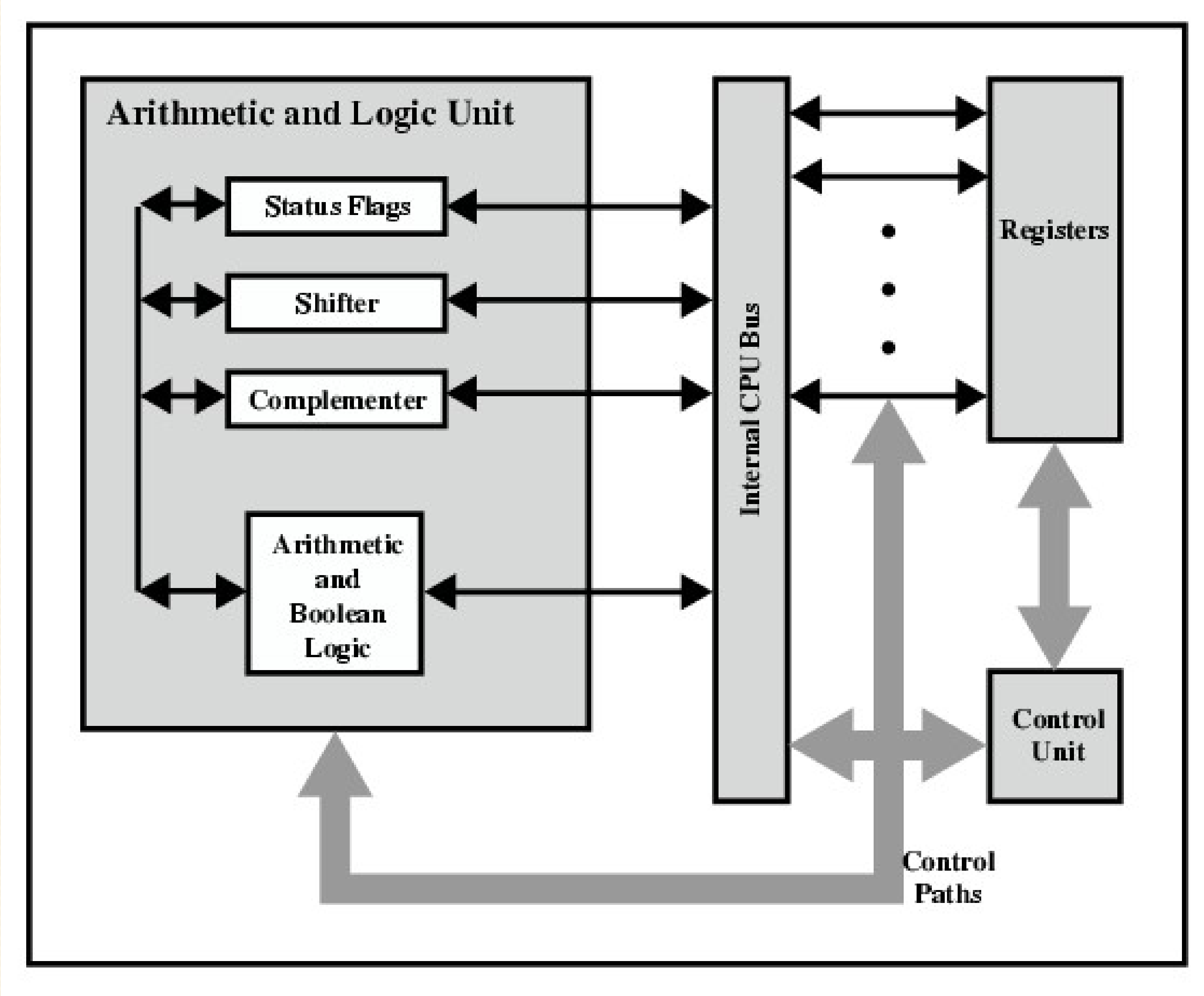
- CPU must:
  - Fetch instructions-reads an instruction from memory
  - Interpret instructions-instruction is decoded to determine what action is required.
  - Fetch data-The execution of an instruction may require reading data from memory or an I/O module
  - Process data-The execution of an instruction 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.

CPU With Systems Bus



- The ALU does the actual computation or processing of data.
- The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU.
- In addition, there is a minimal internal memory, consisting of a set of storage locations, called *registers*

CPU Internal Structure



- The data transfer and logic control paths are indicated, including an element labeled internal processor bus.
- This element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory.

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
- 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

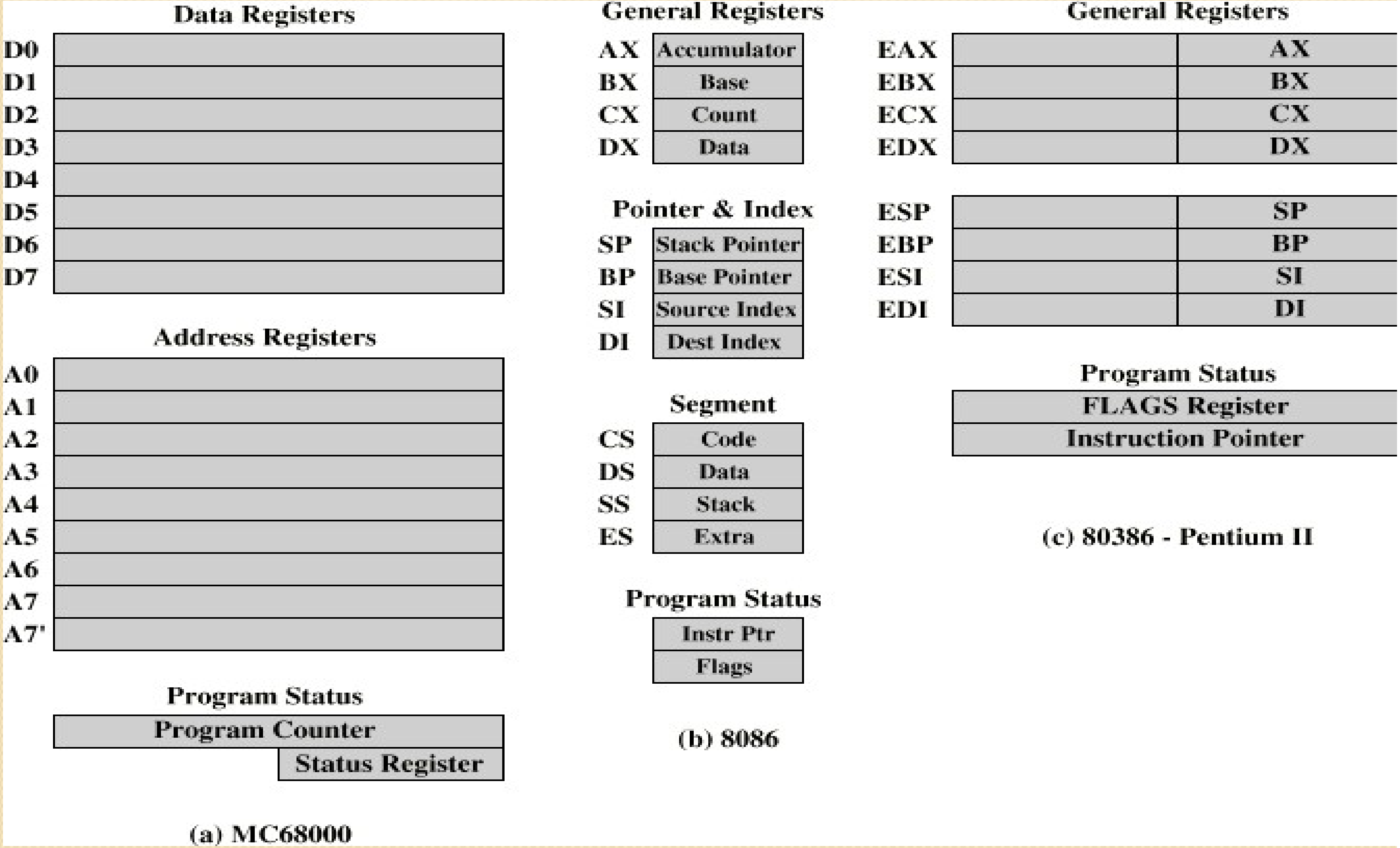
•may be referenced by means of the machine language that the processor executes

- General Purpose-can be assigned to a variety of functions by the programmer
- Data-used only to hold data and cannot be employed in the calculation of an operand address
- Address-somewhat general purpose, or they may be devoted to a particular addressing mode.
  - Segment pointers: In a machine with segmented addressing, a segment register holds the address of the base of the segment.

    •Index registers: These are used for indexed addressing and may be auto indexed.

    •Stack pointer-there is a dedicated register that points to the top of the stack.
- Condition Codes- (Refer next slides)

Example Register Organizations



(a) MC68000

(b) 8086

(c) 80386 - Pentium II

- The MC68000 partitions its 32-bit registers into eight data registers and nine address registers.
- The eight data registers are used primarily for data manipulation and are also used in addressing as index registers.
- The address registers contain 32-bit addresses; two of these registers are also used as stack pointers, one for users and one for the operating system
- 32-bit program counter and a 16-bit status register.

- Intel-special purpose and general purpose registers
- four 16-bit data registers that are addressable on a byte or 16-bit basis, and four 16-bit pointer and index registers.
- The data registers can be used as general purpose in some instructions.
- four 16-bit segment registers
- Compact encoding at the cost of reduced flexibility.
- The 8086 also includes an instruction pointer and a set of 1-bit status and control flags

- user-visible register organization for the Intel 80386,which is a 32-bit microprocessor designed as an extension of the 8086.
- The 80386 uses 32-bit registers.
- However, to provide upward compatibility for programs written on the earlier machine, the 80386 retains the original register organization embedded in the new organization

General Purpose Registers

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

- Why make them general purpose?
  - Increase flexibility and programmer options
  - Increase instruction size & complexity

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(Flag Reg)

- Condition codes are bits set by the processor hardware as the result of operations.
- In addition to the result itself being stored in a register or memory, a condition code is also set
- Condition code bits are collected into one or more 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

Table 12.1 Condition Codes

Advantages	Disadvantages
1. Because condition codes are set by normal arithmetic and data movement instructions, they should reduce the number of COMPARE and TEST instructions needed. 2. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH. 3. Condition codes facilitate multiway branches. For example, a TEST instruction can be followed by two branches, one on less than or equal to zero and one on greater than zero.	1. Condition codes add complexity, both to the hardware and software. Condition code bits are often modified in different ways by different instructions, making life more difficult for both the microprogrammer and compiler writer. 2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections. 3. Often condition code machines must add special non-condition-code instructions for special situations anyway, such as bit checking, loop control, and atomic semaphore operations. 4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

Control & Status Registers

- Program Counter (PC)-Contains the address of an instruction to be fetched
- Instruction Decoding 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

movement of data between the processor and memory
Within the processor, data must be presented to the ALU for processing.
The ALU may have direct access to the MBR and user-visible 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/current instruction being executed

Program Status Word

program status word (PSW) contain status information.
The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

- A set of bits
- Includes Condition Codes
- Sign of last result
- Zero-set when result is 0
- Carry
- Equal
- Overflow
- Interrupt enable/disable-enable or disable interrupts.
- Supervisor-Indicates whether the processor is executing in supervisor or user mode.

- The user state is the default (normal) state of operation, in which user programs are executed.
- The supervisor state is a special mode of operation to which the user has no access.
- When it is in the supervisor state, the processor and its actions are entirely controlled by the Operating System (OS).

- Processor operations mostly involve processing data.
- This data can be stored in memory and accessed from thereon.
- However, reading data from and storing data into memory slows down the processor
- To speed up the processor operations, the processor includes some internal memory storage locations, called **registers**.
- The registers store data elements for processing without having to access the memory. A limited number of registers are built into the processor chip.

The registers are grouped into three categories −
General registers,
Control registers, and
Segment registers.
The general registers are further divided into the following groups −
Data registers,
Pointer registers, and
Index registers.

General Registers

- **AX is the primary accumulator**; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.
- **BX is known as the base register**, as it could be used in indexed addressing (the content of a given index register gets added to an instruction's address part so as to obtain the effective address.).
- **CX is known as the count register**, as the ECX, CX registers store the loop count in iterative operations.
- **DX is known as the data register**. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values.

Pointer Registers

- **Instruction Pointer (IP)** − The 16-bit IP register stores the offset address (*The offset address, which is a part of the address, is added to the start of the segment to address a memory location within the memory segment*) of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.
- **Stack Pointer (SP)** − The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.
  - The stack is a block of memory that may be used for temporarily storing the contents of registers inside CPU. • Stack is accessed by using SP and SS

- **Base Pointer (BP)** − The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine.
  - It is primarily used in accessing parameters passed by the stack .
  - The address in SS (Stack) register is combined with the offset in BP to get the location of the parameter.
  - BP can also be combined with DI (Destination index-16 bit register) and SI (Source index-16 bit register) as base register for special addressing.

Index Registers

SI and DI, are used for indexed addressing and sometimes used in addition and subtraction.

There are two sets of index pointers −

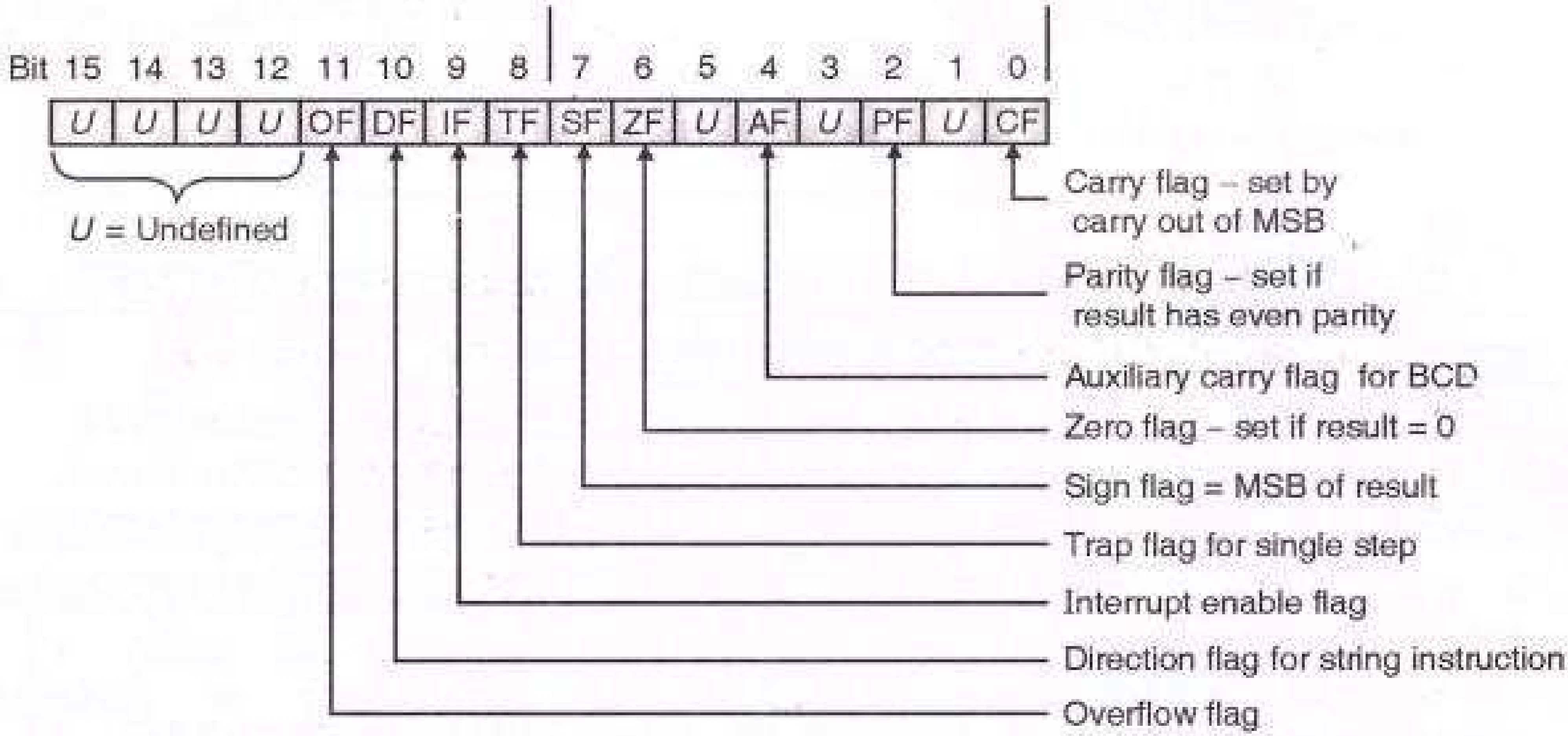
Source Index (SI) − It is used as source index for string operations. It is used in the **pointer addressing of data** and as a source in some string-related operations

Destination Index (DI) − It is used as destination index for string operations. It is used in the **pointer addressing of data** and as a destination in some string-related operations.

Control Registers

- The 32-bit instruction pointer register and the 32-bit flags register combined are considered as the control registers.
- Many instructions involve comparisons and mathematical calculations and change the status of the flags and some other conditional instructions test the value of these status flags to take the control flow to other location.
- Flags-Condition of the microprocessor and control its operation
- The common flag bits are:

FLAG REGISTER 8086



8086 flag register format

- **Overflow Flag (OF)** − It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.
- **Direction Flag (DF)** − It determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.

- **Trap Flag (TF)** − It allows setting the operation of the processor in single-step mode.

  The DEBUG program we used sets the trap flag, so we could step through the

  execution one instruction at a time.

- **Sign Flag (SF)** − It shows the sign of the result of an arithmetic operation. This

  flag is set according to the sign of a data item following the arithmetic operation.

  The sign is indicated by the high-order of leftmost bit. A positive result clears the

  value of SF to 0 and negative result sets it to 1.

- **Zero Flag (ZF)** − It indicates the result of an arithmetic or comparison operation. A

  nonzero result clears the zero flag to 0, and a zero result sets it to 1.

- **Interrupt Flag (IF)** − It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.

- **Auxiliary Carry Flag (AF)** − It contains the carry from **bit 3 to bit 4 following an arithmetic operation**; used for specialized arithmetic. The AF is set when a 1- byte arithmetic operation causes a carry from bit 3 into bit 4.

- **Parity Flag (PF)** − It indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an **odd number of 1-bits sets the parity flag to 1**.

- **Carry Flag (CF)** − It contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a *shift* or *rotate* operation.

Segment Registers

- Segments are specific areas defined in a program for **containing data, code and stack**. There are three main segments −
- **Code Segment** − It contains all the instructions to be executed. A 16-bit Code Segment register or CS register stores the **starting address of the code segment**.
- **Data Segment(DS,ES)** − It contains data, constants and work areas. A 16-bit Data Segment register or DS register **stores the starting address of the data segment.**
- **Stack Segment** − It contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure. The Stack Segment register or SS register **stores the starting address of the stack.**

INSTRUCTION FORMAT (PENTIUM)

- An instruction format defines the layout of the bits of an instruction, in terms of its constituent fields
- Opcodes-Operation Code
- Operands-Data

Instruction=Opcodes+Operands.

- Instruction :- MOV A (Destination),B(Source)
- Examples:

MOV AX,BX Add AX,4 JMP MUL 3,5

Instruction Formats

- Layout of bits in an instruction
- Includes **opcode**
- Includes (implicit or explicit) **operand(s**)
- Usually more than one instruction format in an instruction set

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed

User wants more opcodes, operands (greater flexibility in implementing certain functions**), addressing modes , address range**

All of these things (opcodes, operands, addressing modes, address range) require bits and push in the direction of longer instruction length-more space

Allocation of Bits

- Number of addressing modes-
- Number of operands

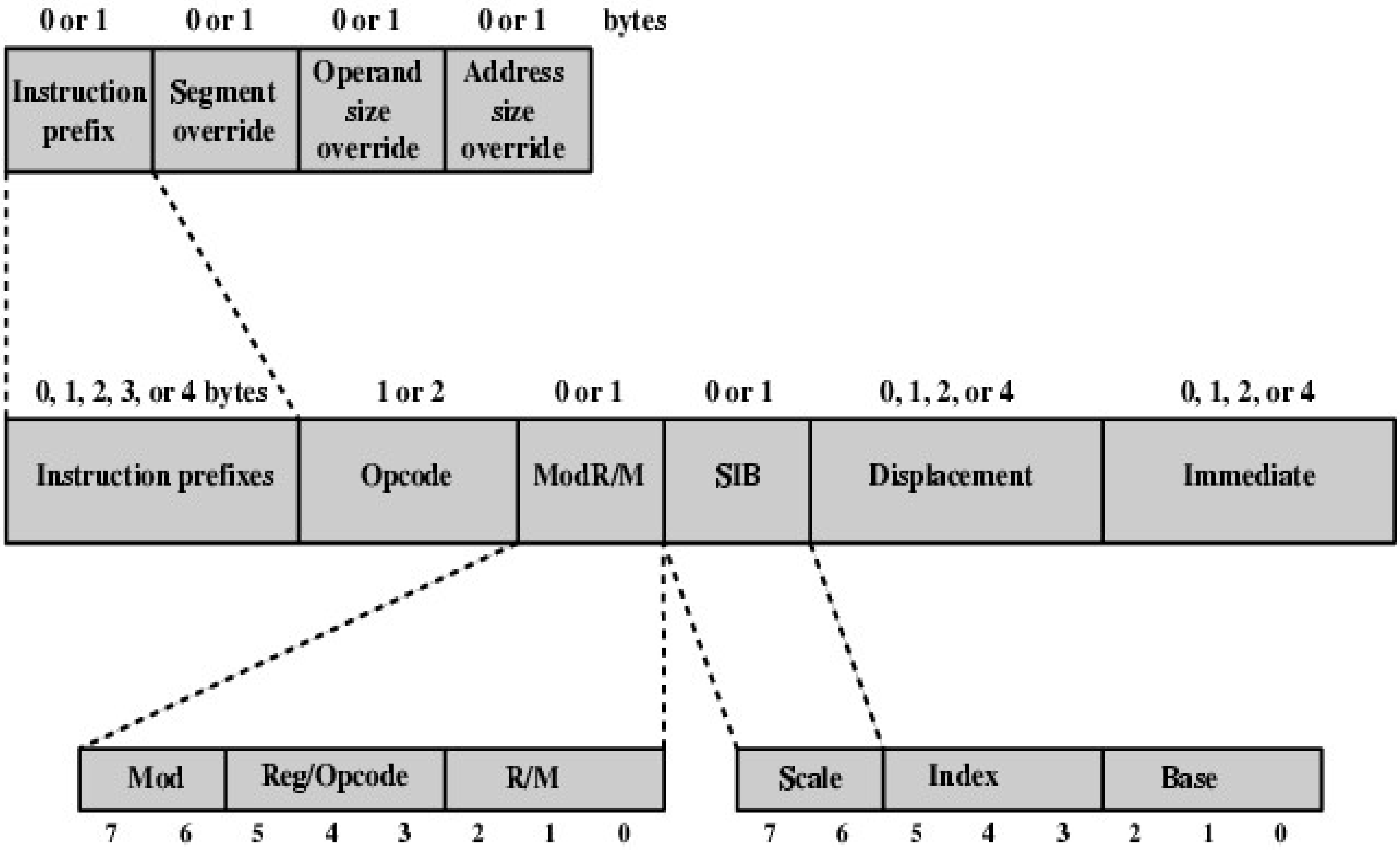
•Register versus memory-A machine must have registers so that data can be brought into the processor for processing. The more that registers can be used for operand references, the fewer bits are needed

•Number of register sets-Some architectures, including that of the x86, have a collection of two or more specialized sets (data & addressing). a functional split requires fewer bits to be used in the instruction.

- Address range-the range of addresses that can be referenced is related to the number of address bits. it is still convenient to allow rather large displacements from the register address, which requires a relatively large number of address bits in the instruction.

- Address Granularity-In a system with 16- or 32-bit words, an address can reference a word or a byte at the designer's choice. Byte addressing is convenient for character manipulation but requires, for a fixed size memory, more address bits.

Pentium Instruction Format



- Instructions are made up of from zero to four optional instruction prefixes, a 1- or 2-byte opcode, an optional address specifier (which consists of the ModR/m byte and the Scale Index byte) an optional displacement, and an optional immediate field.

- **Instruction Prefixes-LOCK** prefix (ensure the use of shared memory in multiprocessor environments.) or one of the **REPEAT** prefixes(REP,REPE,REPNE,REPZ,REPNZ…. specify repeated operation of a string, which enables the x86 to process strings much faster)

- **Segment Override**-explicitly specifies which segment register an instruction should use

PENTIUM INSTRUCTION FORMAT

- **Address Size**-16 or 32 bit (switch)-either 16- or 32-bit addresses. The address size determines the displacement size in instructions and the size of address offsets generated during effective address calculation. address size prefix switches between 32-bit and 16-bit address generation.

- **Operand Size**-16 or 32 bit (switch)-An instruction has a default operand size of 16 or 32 bits, and the operand prefix switches between 32-bit and 16-bit operands.

PENTIUM INSTRUCTION FORMAT

Opcode- 1, 2, or 3 bytes in length. It include bits that specify if data is byte- or full-size, direction of data operation (to or from memory), and whether an immediate data field must be sign extended.

- **Mod R/m**-addressing information

  -specifies whether an operand is in a register or in memory; if it is in memory, then fields within the byte specify the addressing mode to be used

  -consists of three fields: The Mod field (2 bits) combines with the r/m field to form 32 possible values: 8 registers and 24 indexing modes;

  -The Reg/Opcode field (3 bits) specifies either a register number or three more bits of opcode information;

  -the r/m field (3 bits) can specify a register as the location of an operand, or it can form part of the addressing

- Mod+ r/m (combined info of registers and addressing modes)
- Register-register or opcode info

- **SIB-**specify fully the addressing mode
  - **Scale (2 bits)-**scale factor for scaled indexing
  - **Index (3 bits)**-specifies index register(SI,DI)
  - **Base (3 bits)**-specifies base register(BX)
- **Displacement**-When the addressing-mode specifier indicates that a displacement is used, an 8-, 16-, or 32-bit signed integer displacement field is added.
- **Immediate**-provides the value of 8/16/32 bit operand

William Stallings Computer Organization and Architecture 6th Edition

Chapter 11 Instruction Sets:

Addressing Modes and Formats

Addressing modes

- The way for which an operand is specified for an instruction in the accumulator, in a general purpose register or in memory location, is called **addressing mode**.
- Different opcodes will use different addressing modes.
- one or more bits in the instruction format can be used as a mode field. The value of the mode field determines which addressing mode is to be used.
- In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

These modes are illustrated in Figure 11.1. In this section, we use the following notation:

A = contents of an address field in the instruction

R = contents of an address field in the instruction that refers to a register

EA = actual (effective) address of the location containing the referenced operand

(X) = contents of memory location X or register X

Immediate Addressing

- In this mode **data is present in address field of instruction .**
- Designed like one address instruction format
  - Note:Limitation in the immediate mode is that the range of constants are restricted by size of address field.
- FEATURES
  - Operand is part of instruction
  - Operand = address field
  - No memory reference to fetch data-saving memory
  - Fast
  - Limited range-size of the number is restricted to the size of the address field

(a) Immediate

MOV AX, 2000

MOV CL, 0A

ADD AL, 45

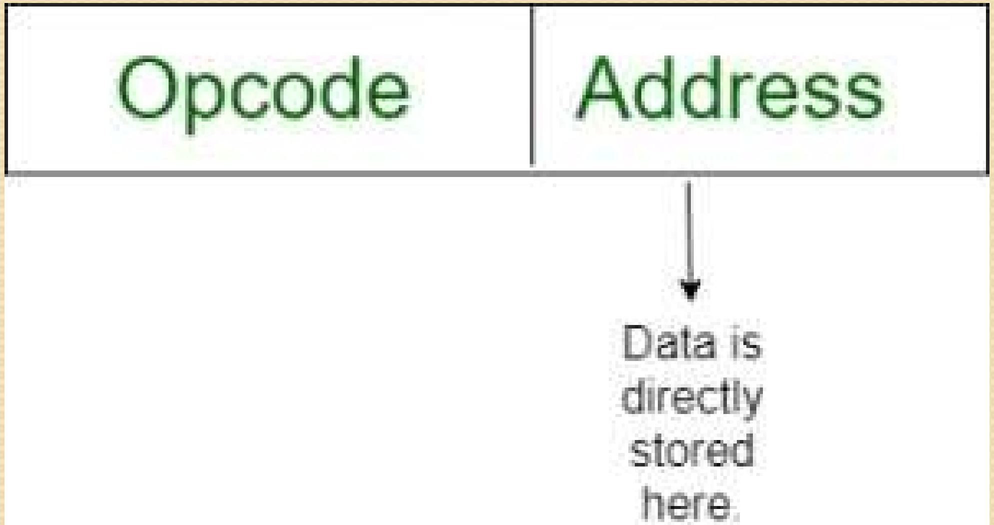
ADD AX, 0000

MOV CX, 4929 H

(source operand-16 bit-

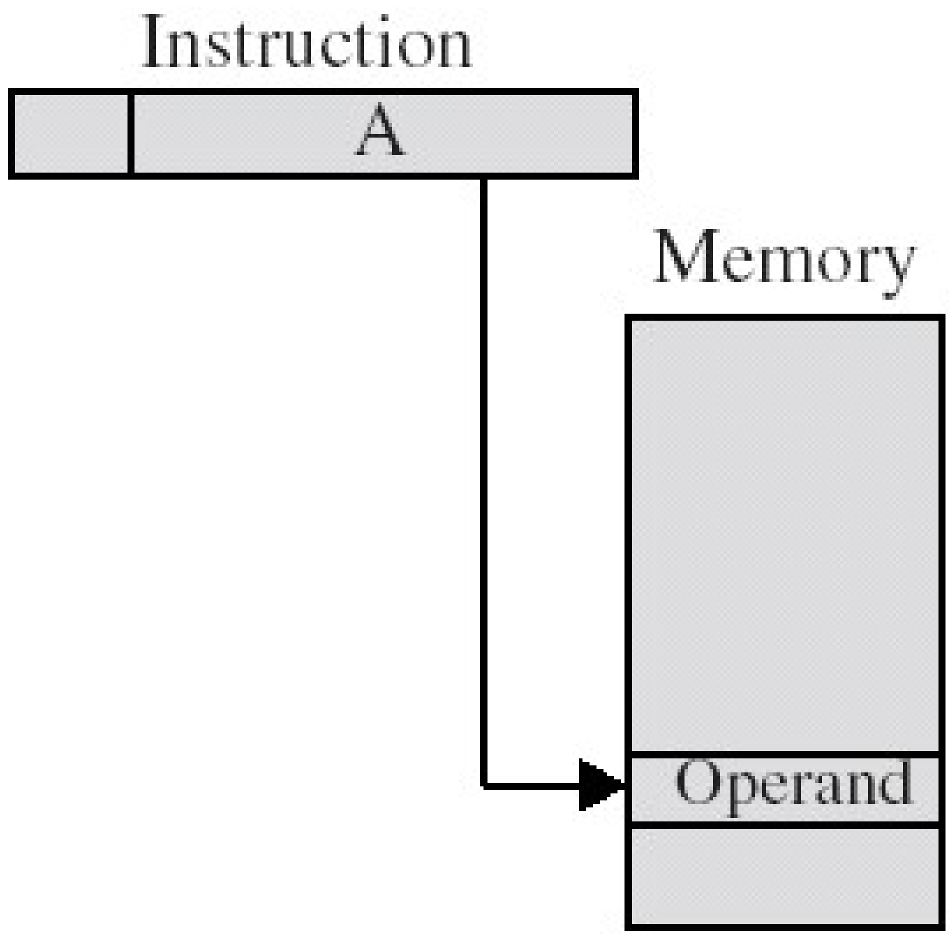
part of instruction)

ADD AX, 2387 H,



MOV AL, FFH (15 bit data)

Direct(M) Addressing Diagram



(b) Direct

Direct Addressing

- Address field contains address of operand
- EFFECTIVE ADDRESS EA = address field (A)
  - Look in memory at address value for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

- Address of the memory location is given directly
- The operand's offset is given in the instruction as an 8 bit or

  16 bit displacement element.

- In this addressing mode the 16 bit effective address of the

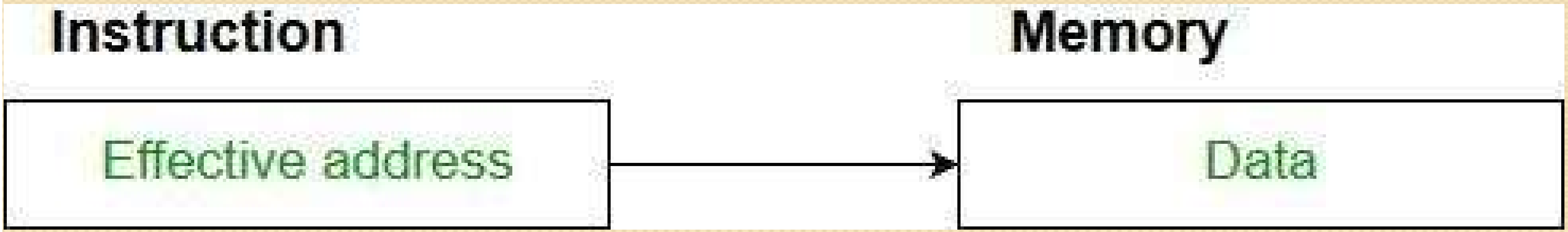
  data is the part of the instruction.

- *Here only one memory reference operation is required to*

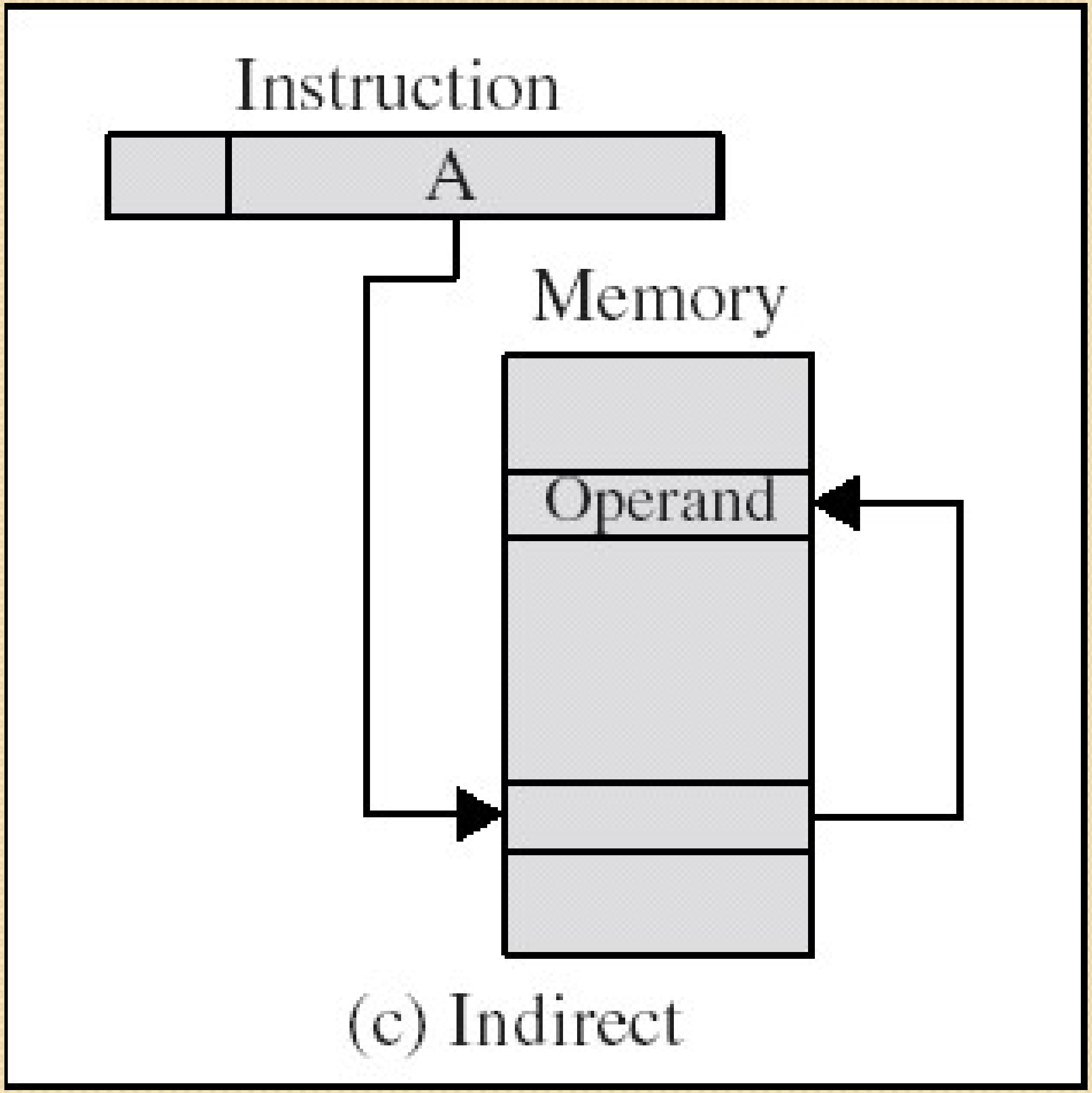
  access the data.

MOV AX, [0500]

ADD AL,[0301]



Indirect Addressing Diagram



- In this mode address field of instruction contains the address of effective address.
- Here two references are required. 1st reference to get effective address. 2nd reference to access the data.

.

Indirect Addressing (1/2)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA =(A)
  - Look in A, find address (A) and look there for operand
- e.g. ADD AX, (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2/2)

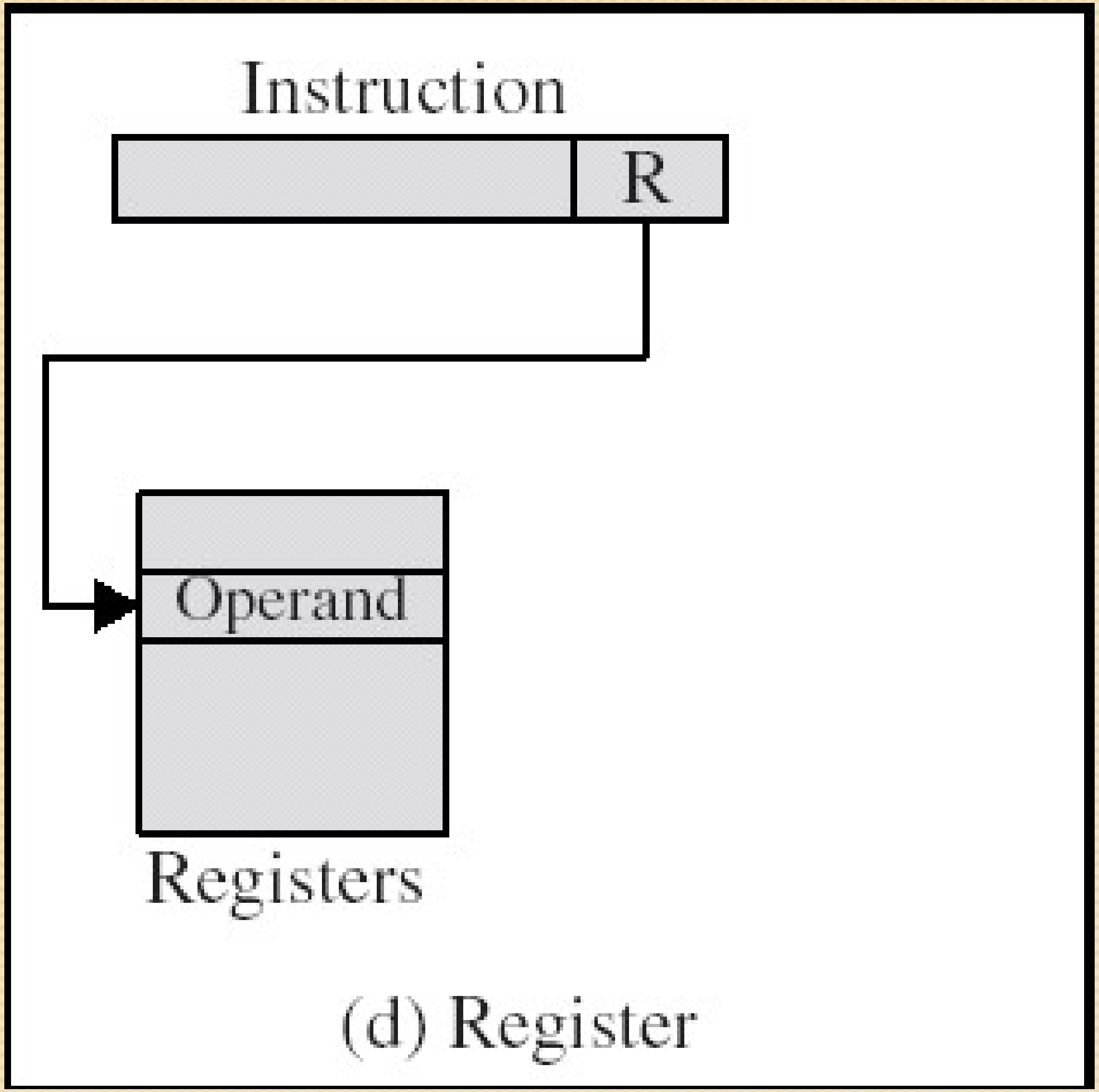
- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Based on the availability of Effective address, Indirect mode is of two kind:

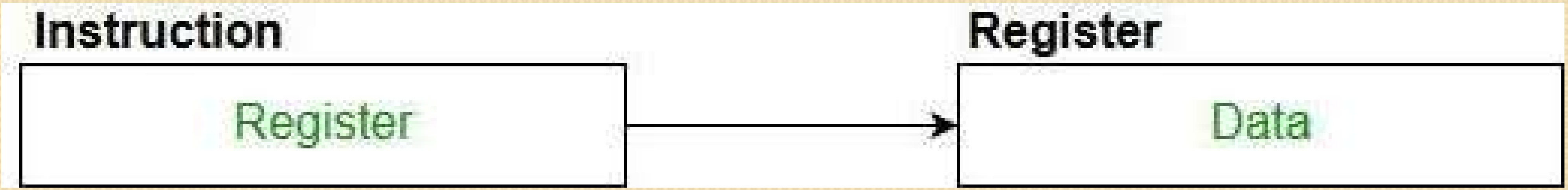
- REGISTER INDIRECT: In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.
  - *Here one register reference,one memory reference is required to access the data.*

- MEMORY INDIRECT: In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.
  - *Here two memory reference is required to access the data*

Register Addressing Diagram



- In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers.
- The data is in the register that is specified by the instruction.
  - *Here one register reference is required to access the data.*



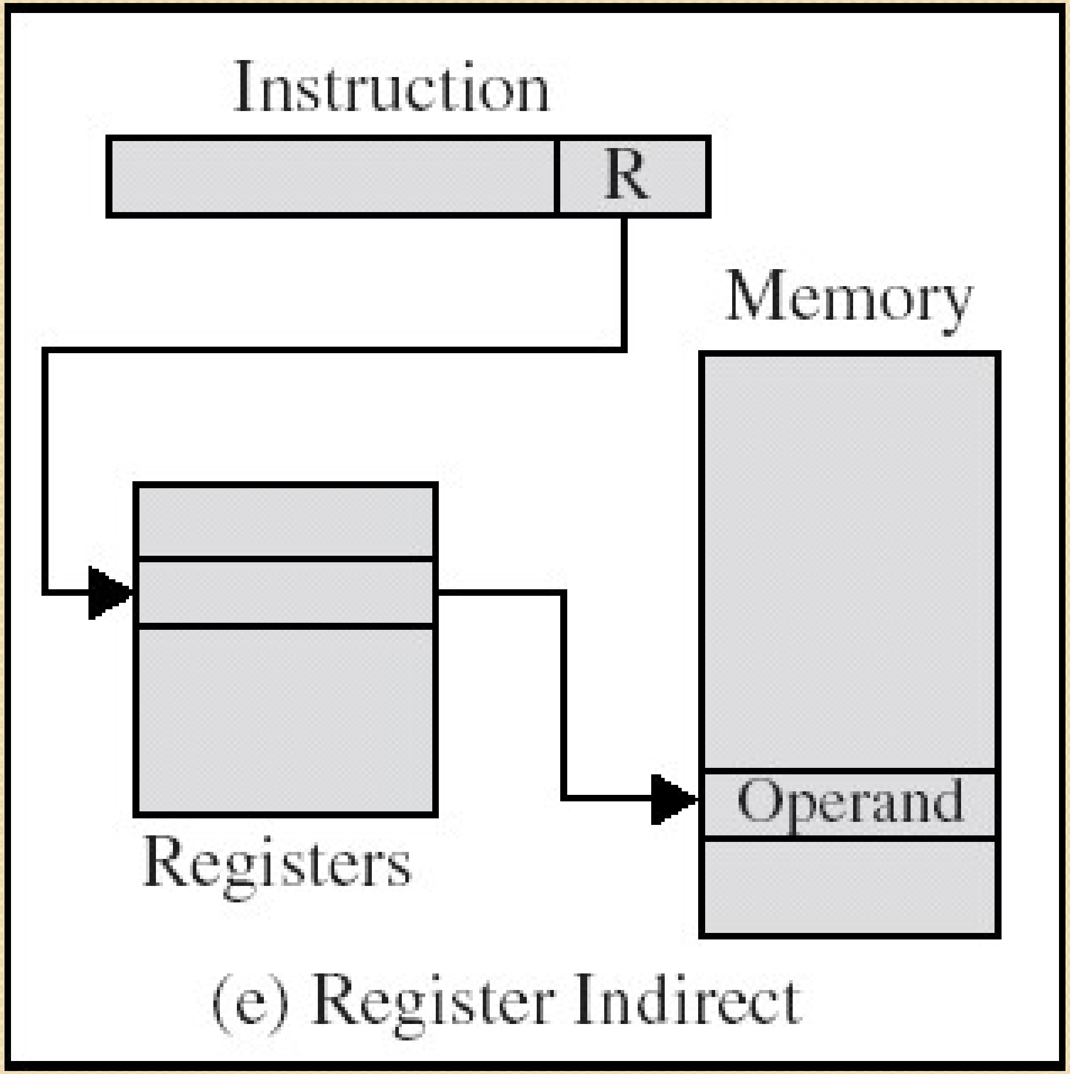
Register Addressing (1/2)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
  - MOV AX, BX
  - ADD AX, BX

Register Addressing (2/2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
- Similar to Direct addressing

Register Indirect Addressing Diagram



- In this addressing the operand's offset is placed in any one of the registers **BX,BP,SI,DI as specified in the instruction**.
- The effective address of the data is in the base register or an index register that is specified by the instruction.
  - *Here two register reference is required to access the data.*

Register Indirect Addressing

MOV AX, [BX]

(move the contents of memory location s addressed by the register BX to the register AX)

MOV AX, [DI]

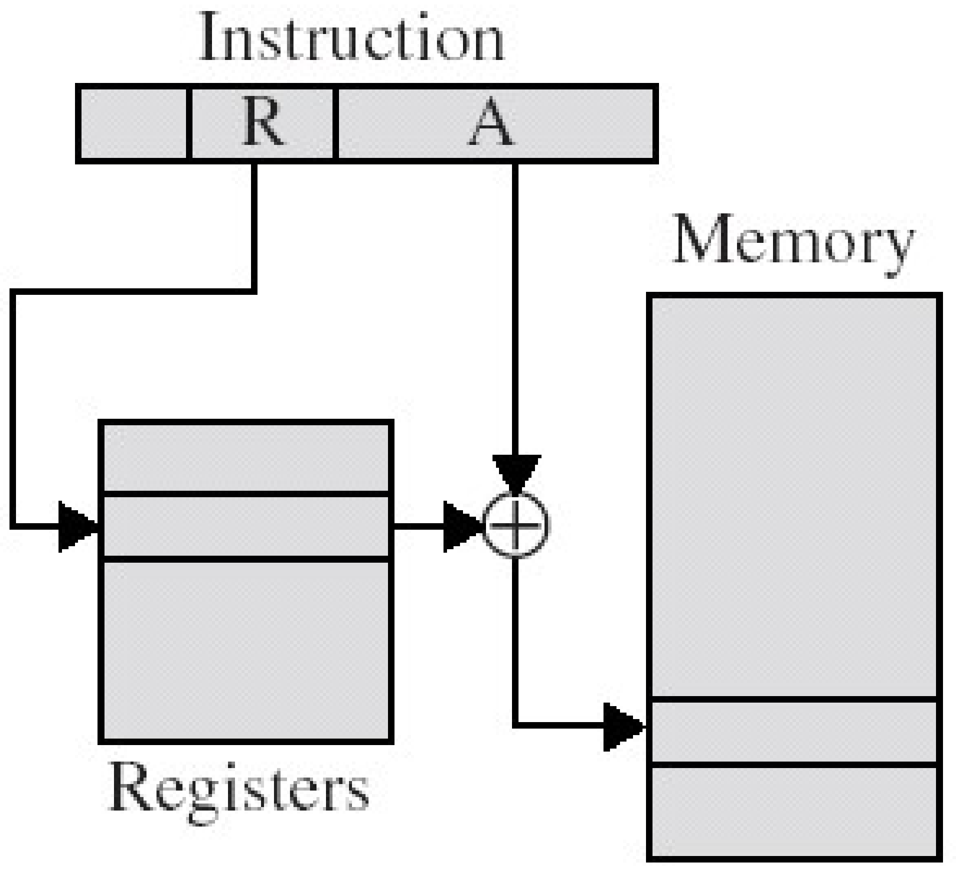
ADD AL, [BX]

MOV AX, [SI]

- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

- the address space limitation (limited range of addresses) of the address field is overcome by having that field refer to a wordlength location containing an address.
- In addition, register indirect addressing uses one less memory reference than indirect addressing.

Displacement Addressing Diagram



(f) Displacement

Displacement Addressing

- Combination of direct and register indirect
- $EA = A + (R)$
- Effective address=start address + displacement
- Effective address=Offset + (Segment Register)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

- The value contained in one address field is used directly. (A)
- The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

- $EA = A + (R)$

- three of the most common uses of displacement addressing:
- Relative addressing
- Base-register addressing
- Indexing

Relative addressing

- PC-relative addressing,
- the implicitly referenced register is the program counter (PC).
- That is, the next instruction address is added to the address field to produce the EA.

Base-Register Addressing

- Base register addressing mode is used to implement inter segment transfer of control.
- The referenced register contains a main memory address, and the address field contains a displacement
- In this mode effective address is obtained by adding base register value to address field value.
- EA= Base register + Address field value.

Indexed Addressing

- The address field references a main memory address, and the referenced register contains a positive displacement from that address.
- The operand’s offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.
  - MOV AX, [SI +05]
- efficient mechanism for performing iterative operations
- Autoindexing-typical INC/DEC

$$EA = A + (R)$$
$$(R) \leftarrow (R) + 1$$

Stack Addressing

•stack is a linear array of locations. It is sometimes referred to as a *pushdown list or last-in-first-out queue*
•stack pointer whose value is the address of the top of the stack.
•form of implied addressing () instructions that comprise only an opcode without an operand

Operand is (implicitly) on top of stack

- e.g.
  - ADD Pop top two items from stack and add and push

  -PUSH AX

  POP AX

- No memory reference

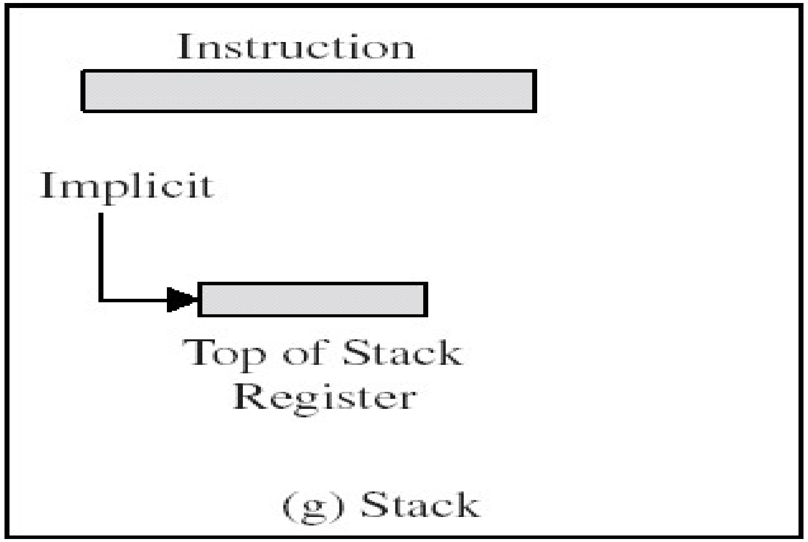


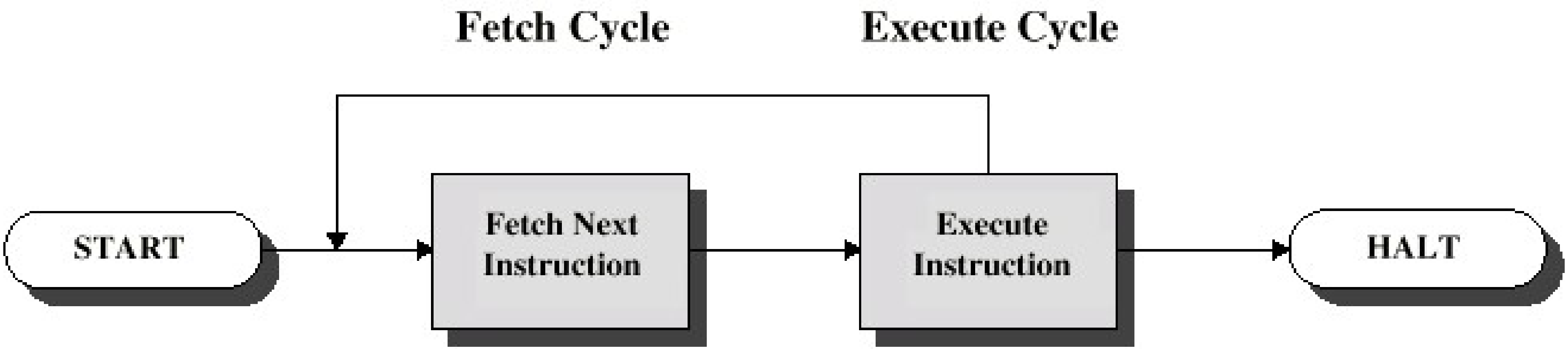
Table 11.1 Basic Addressing Modes

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

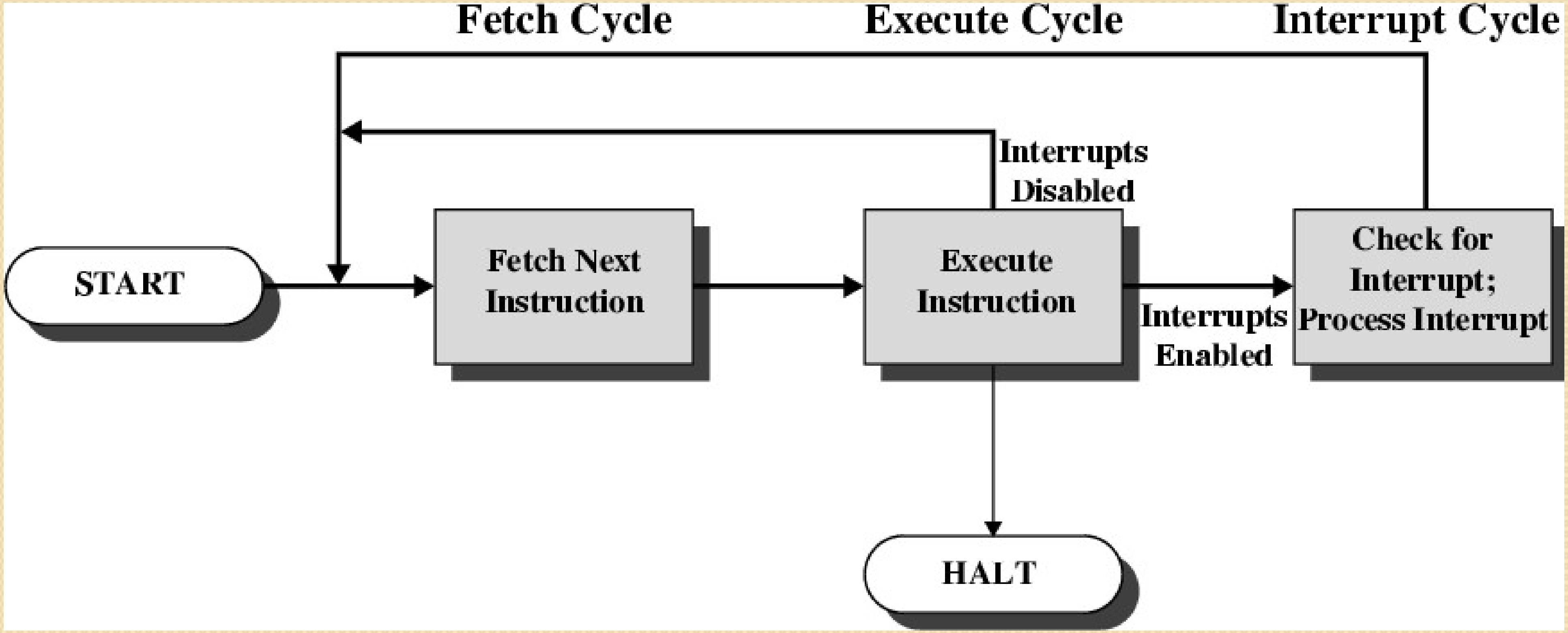
Instruction Cycle

Instruction Cycle

- Two steps:
  - Fetch
  - Execute



Instruction Cycle with Interrupts



Instruction Cycle

- It is the time in which a single instruction is fetched from memory, decoded, and executed
- An **Instruction Cycle** requires the following subcycle:
  - **FETCH**
  - **EXECUTE**
  - **INDIRECT**

Instruction Cycle

- **Fetch**

  Read next instruction from memory into the processor

- **Indirect Cycle (Decode Cycle)**

  May require memory access to fetch operands, therefore more memory accesses.

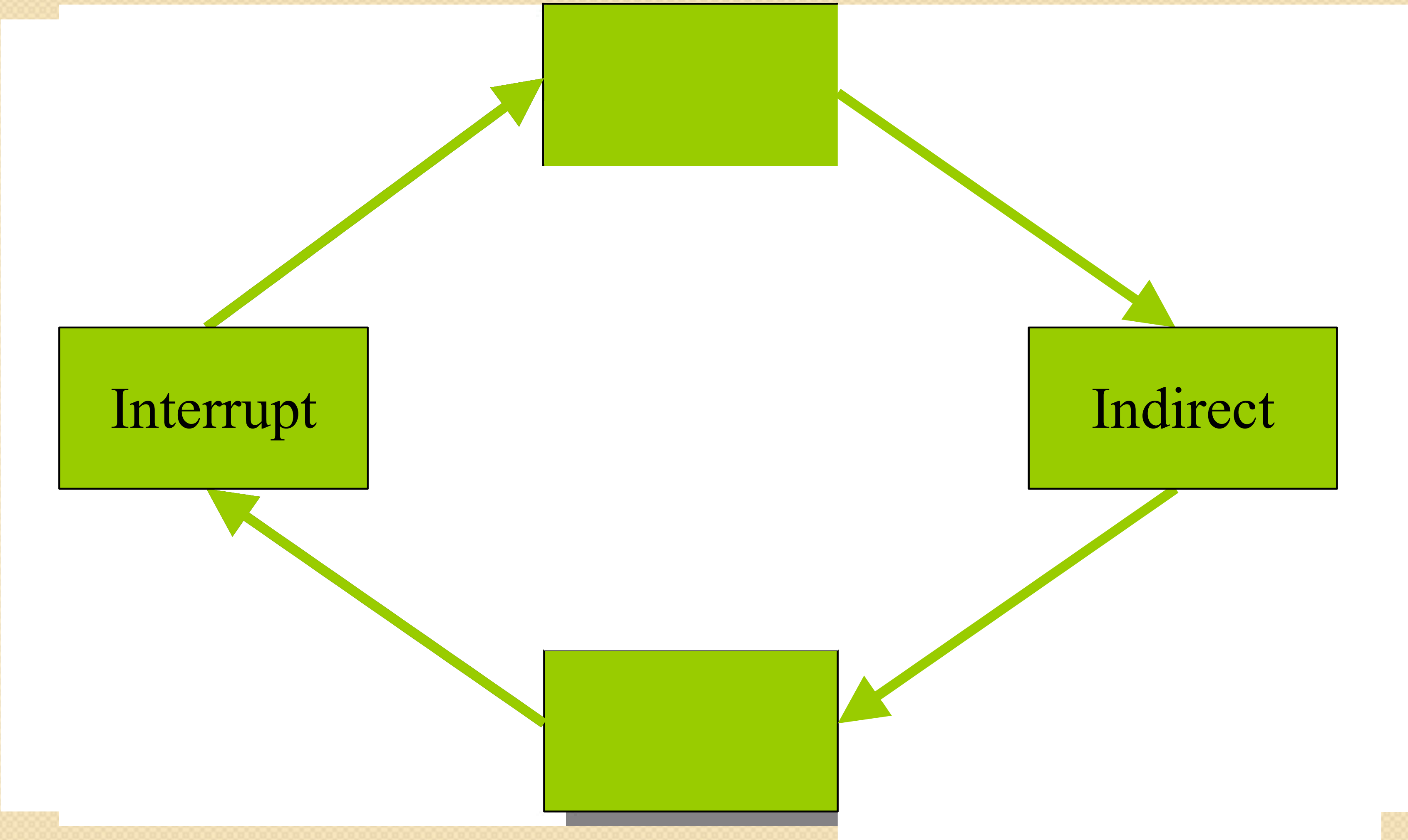
- **Interrupt**

  Save current instruction and service the interrupt

- **Execute**

  Interpret the opcode and perform the indicated operation

Instruction Cycle



Instruction Cycle State Diagram

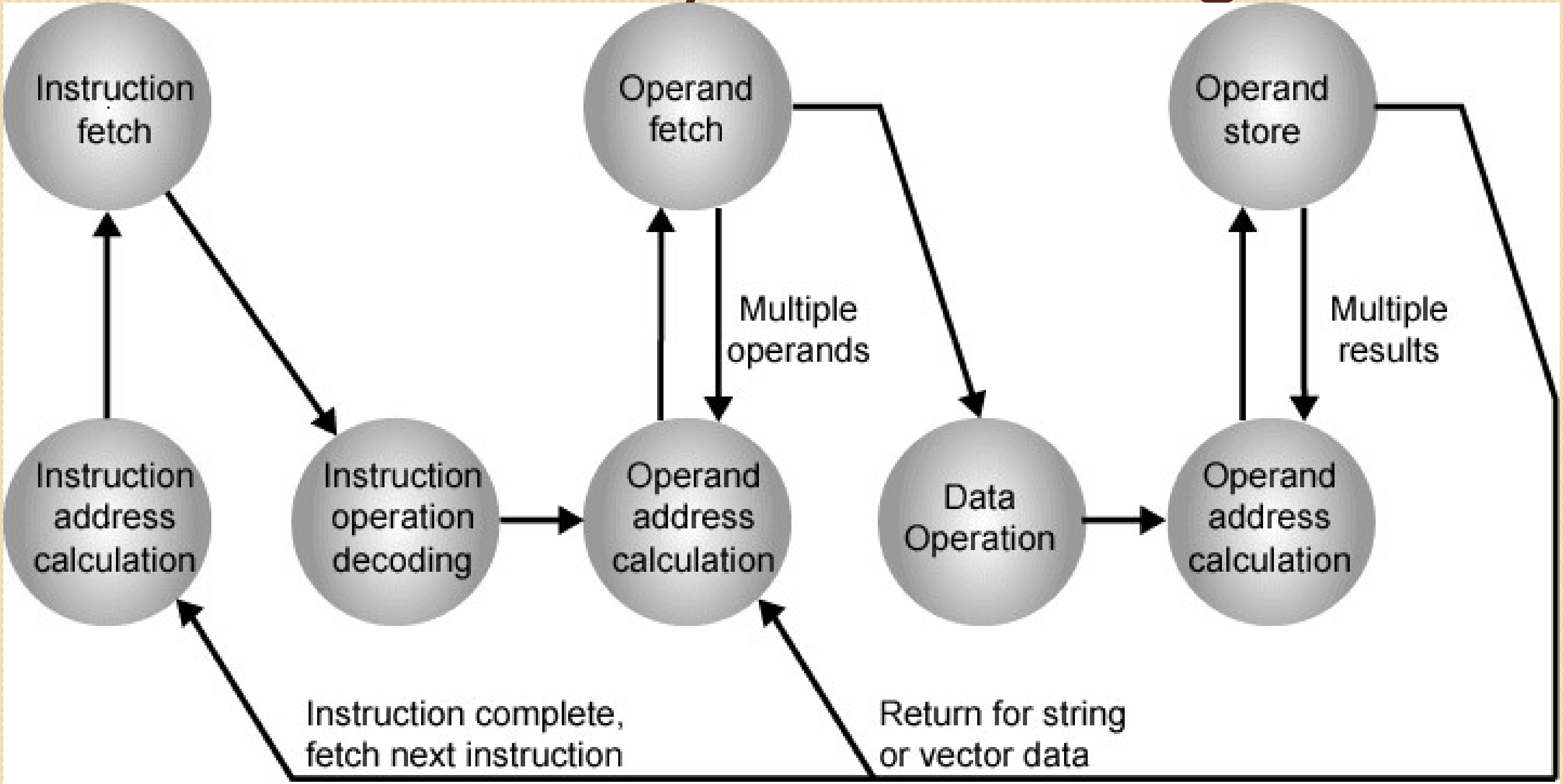


Figure 10.1 Instruction Cycle State Diagram

Instruction Cycle State Diagram

- This illustrates more correctly the nature of the instruction cycle.
- Once an instruction is fetched, its operand specifiers must be identified.
- Each input operand in memory is then fetched, and this process may require indirect addressing.
- Register-based operands need not be fetched.
- Once the opcode is executed, a similar process may be needed to store the result in main memory.

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/current instruction being executed

Fetch Cycle

- **Program Counter (PC)** holds address of next instruction to be fetched
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

Fetch Sequence (symbolic)

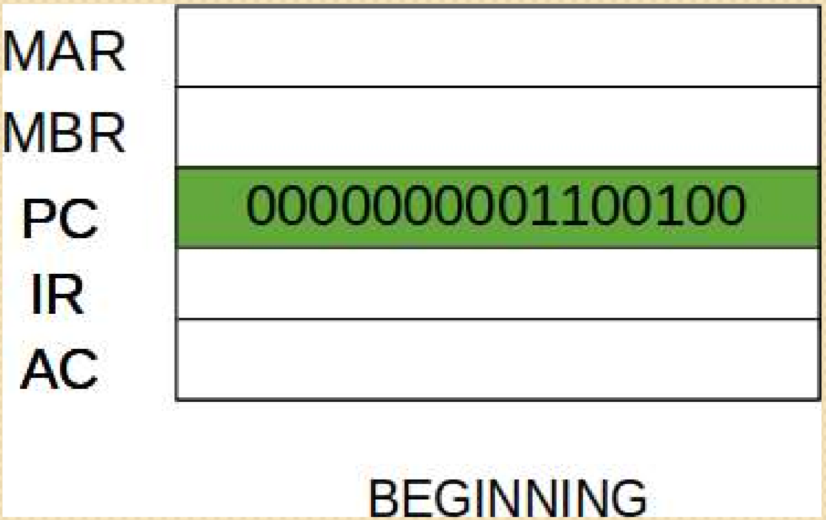
t1 : MAR ⟵ PC

t2 : MBR ⟵ MEMORY

PC ⟵ (PC) + I

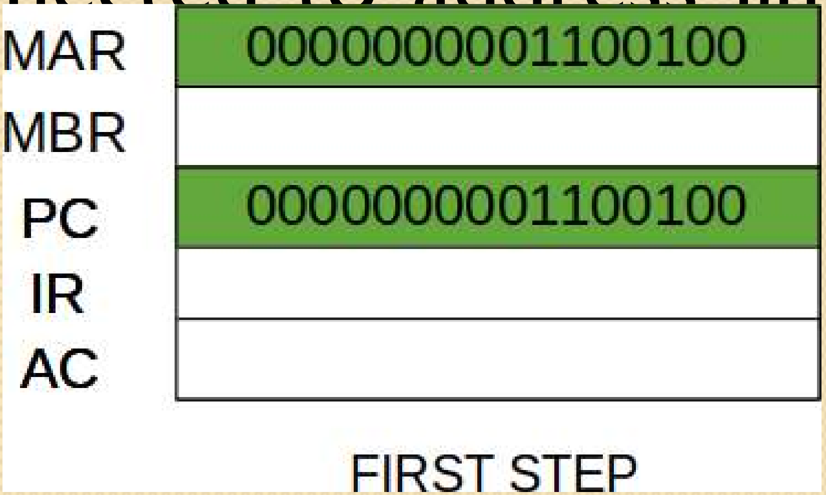
t3 : IR ⟵ (MBR)

- **The Fetch Cycle –**
  At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).



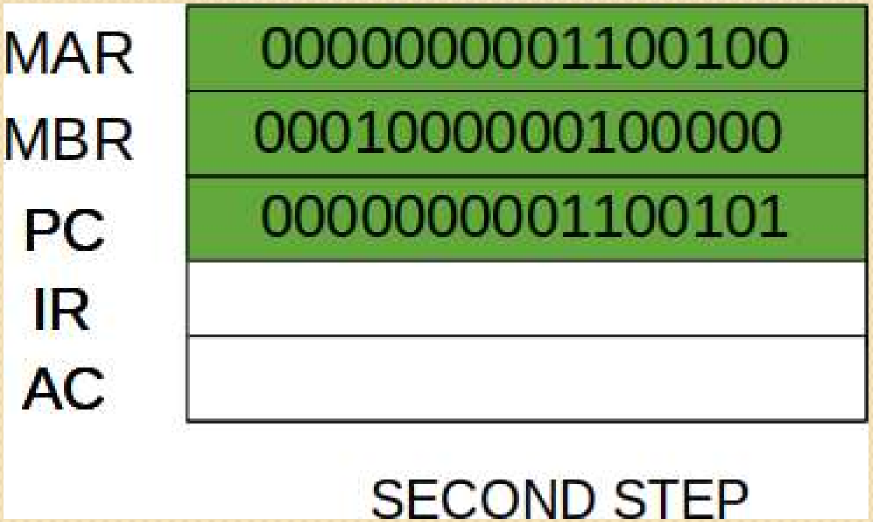
Step 1:

- The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.



Step 2:

- The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR).
- Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)



Step 3:

- The content of the MBR is moved to the instruction register(IR)

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100100
IR	0001000000100000
AC	

THIRD STEP

Indirect Cycle

t1 : MAR $\leftarrow$ (IR(ADDRESS))

t2 : MBR $\leftarrow$ MEMORY

t3 : IR(ADDRESS) $\leftarrow$ (MBR(ADDRESS))

Step 1:

The address field of the instruction is transferred to the MAR.

This is used to fetch the address of the operand. **Step 2**:

The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing) **Step 3**:

The IR is now in the state, as if indirect addressing has not been occurred.

Interrupt Cycle

- At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not.
- If an enabled interrupt has occurred then Interrupt Cycle occurs.
- The nature of this cycle varies greatly from one machine to another.

t1 : MBR $\leftarrow$ (PC)

t2 : MAR $\leftarrow$ SAVE_ADDRESS

PC $\leftarrow$ ROUTINE_ADDRESS

t3 : MEMORY $\leftarrow$ (MBR)

- Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.

  Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.

  PC is loaded with the address of the start of the interrupt-processing routine.

  Step 3: MBR, containing the old value of PC, is stored in memory.

  – **Note:** In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save_address and the routine_address before they are transferred to the MAR and PC respectively.a

Execute Cycle (ADD)

Different for each instruction

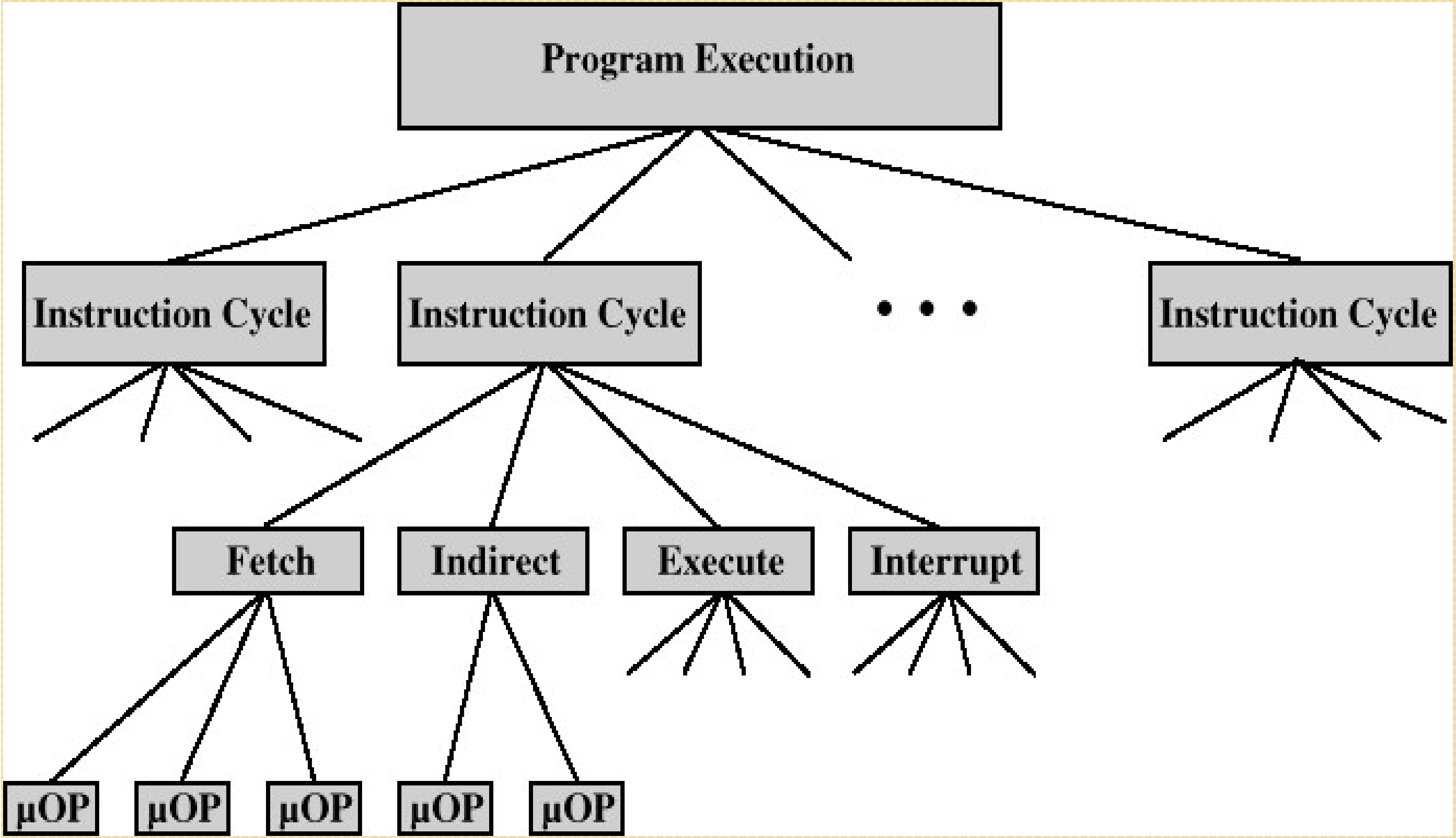
e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1

$$t_1 : \mathrm{IR} \rightarrow \mathrm{MAR}$$

$$t_2 : \mathrm{MEMORY} \rightarrow \mathrm{MDR}$$

$$t_3 : \mathrm{MDR} + \mathrm{R1} \rightarrow \mathrm{R1}$$

Constituent Elements of Program Execution



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

Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical ops

William Stallings
Computer Organization and Architecture
8th Edition

Chapter 15

Control Unit Operation

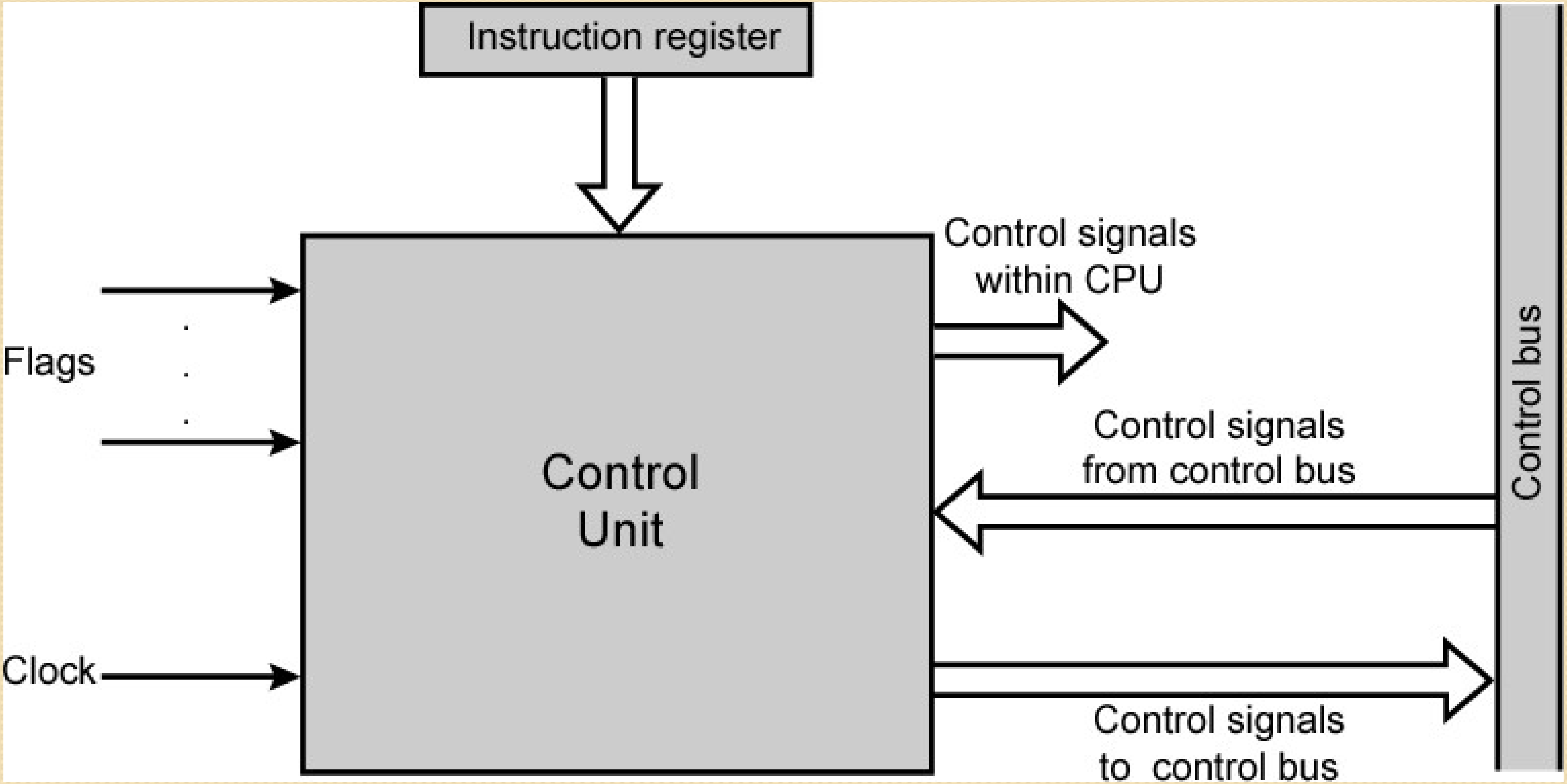
Functional Requirements(of Control Unit)

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

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/current instruction being executed

Model of Control Unit



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(input)

- **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

Control Signals - output

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

Control Unit Organization

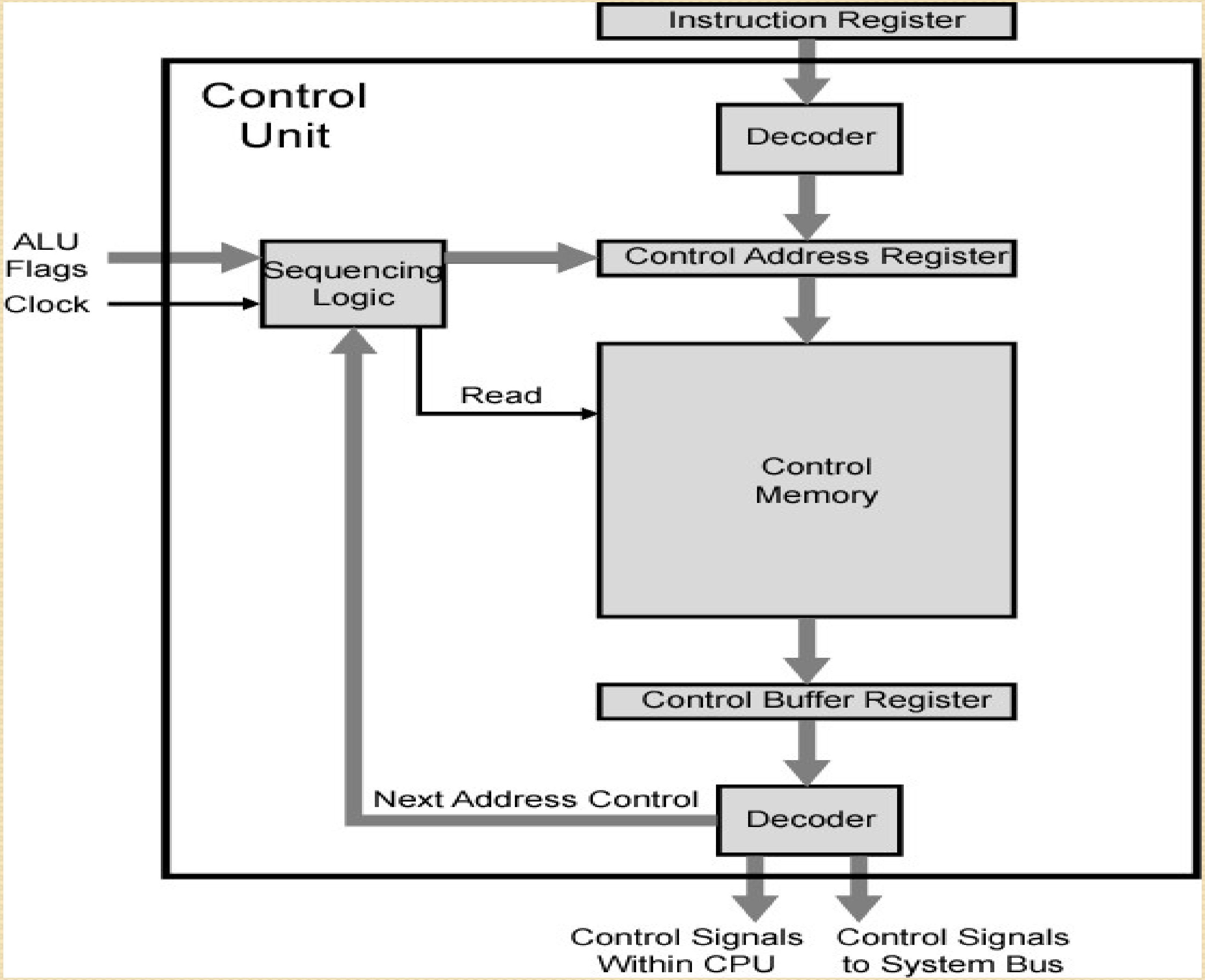
Implementatio n

- 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

Chapter 16
Micro-programmed Control

William Stallings Computer Organization and Architecture 8th Edition

Functioning of Micro programmed Control Unit



Micro programmed Control Unit Fun

- 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

Advantages and Disadvantages of Microprogramming

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

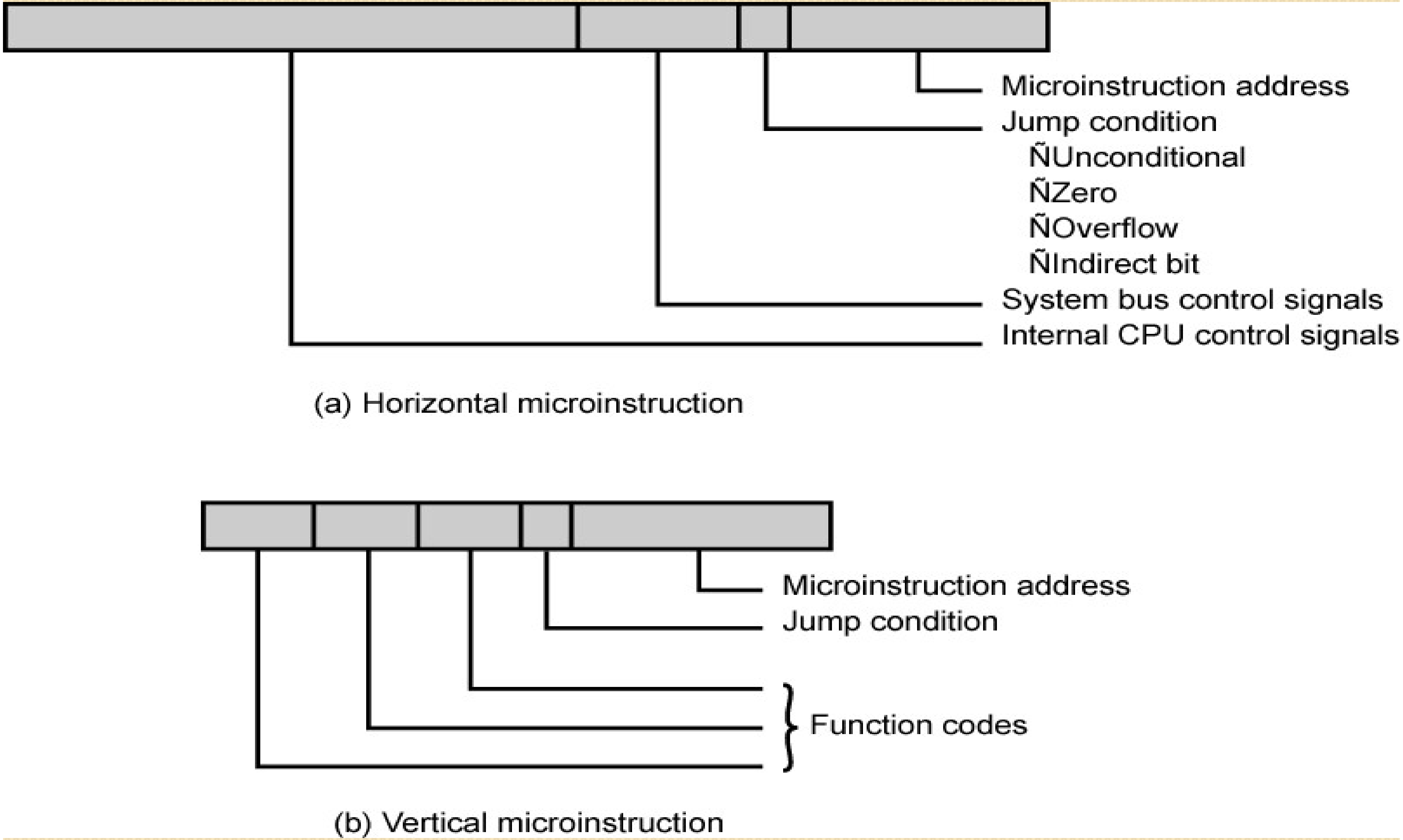
Tasks Done By Microprogrammed Control Unit

- Microinstruction sequencing
- Microinstruction execution
- Must consider both together

Micro-instruction Types

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

Typical Microinstruction Formats



Vertical Micro-programming

- **Width** is **narrow**
- **Limited** ability to express **parallelism**
- Considerable encoding of control information require **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

William Stallings Computer Organization and Architecture 7th Edition

Chapter 13

Reduced Instruction Set Computers

v/s

Complex Instruction Set Computers

Introduction

- The architectural design of the **CPU** is RISC & CISC.
- Hardware fused with software (**Intel v/s Apple**)
- **Intel's** hardware oriented approach is termed as CISC while that of **Apple** is RISC
- **Instruction Set Architecture- Interface** to allow easy communication between the programmer and the hardware.
- ISA- execution of data, copying data, deleting it, editing
- Instruction Set , Addressing Modes,

RISC-Reduced Instruction Set Computer

- RISC processor design has separate **digital circuitry** in the control unit
- Signals needed for the execution of each instruction in the instruction set of the processor.
- Examples of RISC processors:
  - IBM RS6000, MC88100
  - DEC's Alpha 21064, 21164 and 21264 processors

CISC-Complex Instruction Set Computer

- Control unit →micro-electronic circuitry

  - generates a set of control signals →activated by a micro-code

- The primary goal of CISC architecture is to complete a task in as few lines of assembly code as possible.

- **Examples of CISC processors are:**
  - Intel 386, 486, Pentium, Pentium Pro, Pentium II, Pentium III
  - Motorola’s 68000, 68020, 68040, etc.

CISC processor features

- Instruction set with 120-350 instructions
- Variable instruction/data formats
- Small set of general purpose registers(8-24)
- A large number of addressing modes
- High dependency on micro program
- Complex instructions to support HLL features

CISC processor features

- Complex pipelining
- Many functional chips needed to design a computer using CISC
- Difficult to design a superscalar processor

RISC processor features

- Instruction set with limited number of instructions
- Simple instruction format
- Large set of CPU registers
- Very few addressing modes
- Easy to construct a superscalar processor

RISC processor features

- Hardwired control unit for sequencing microinstructions
- Supports on chip cache memory
- All functional units on a single chip
- Simple pipelining

Example for RISC vs. CISC

Consider the the program fragments:

CISC

```
mov ax, 10
mov bx, 5
mul bx, ax
```

RISC

```
      mov ax, 0
      mov bx, 10
      mov cx, 5
Begin add ax, bx
      loop Begin
```

The total clock cycles for the CISC version might be:

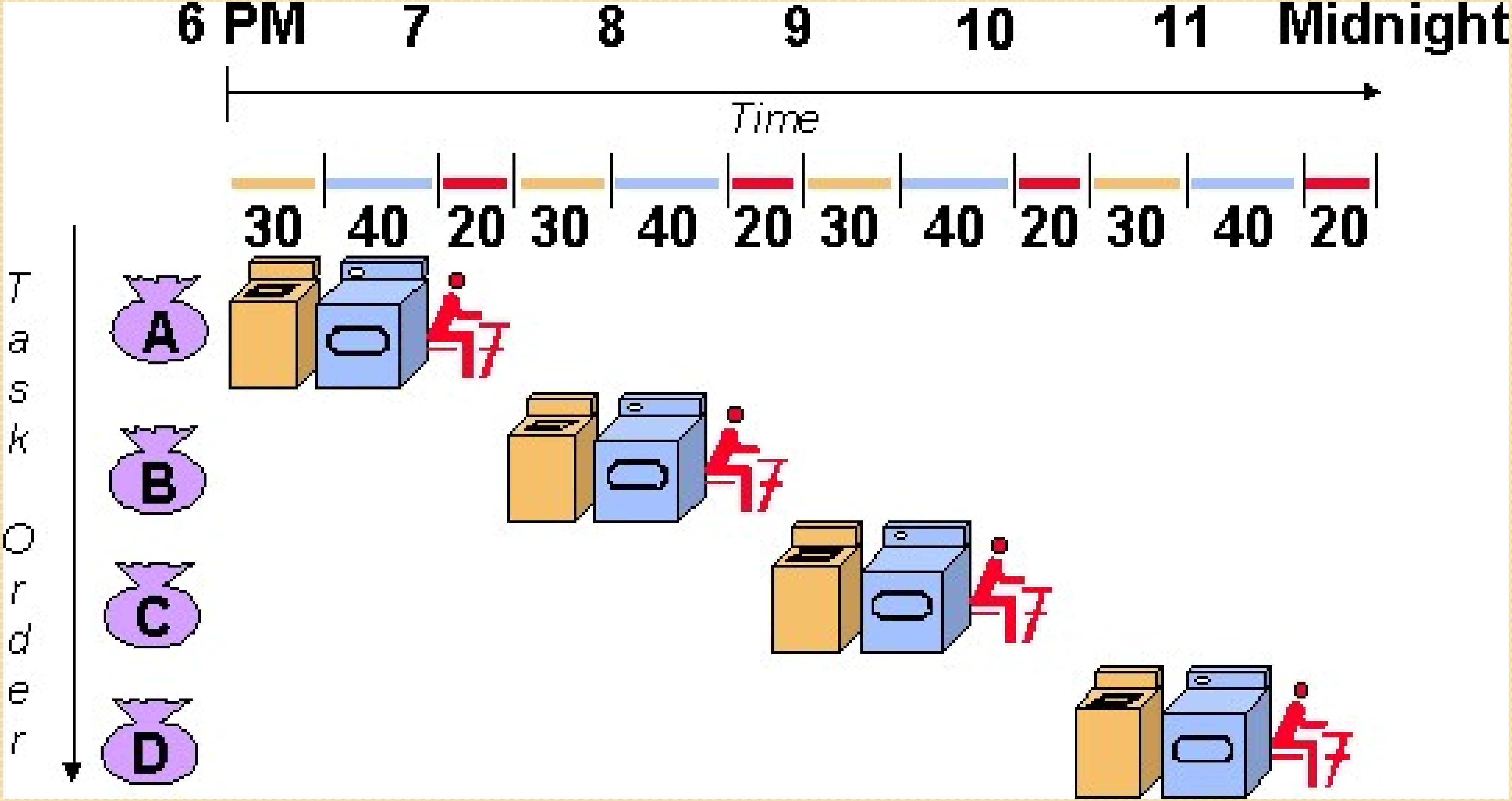
(2 movs × 1 cycle) + (1 mul × 30 cycles) = 32 cycles

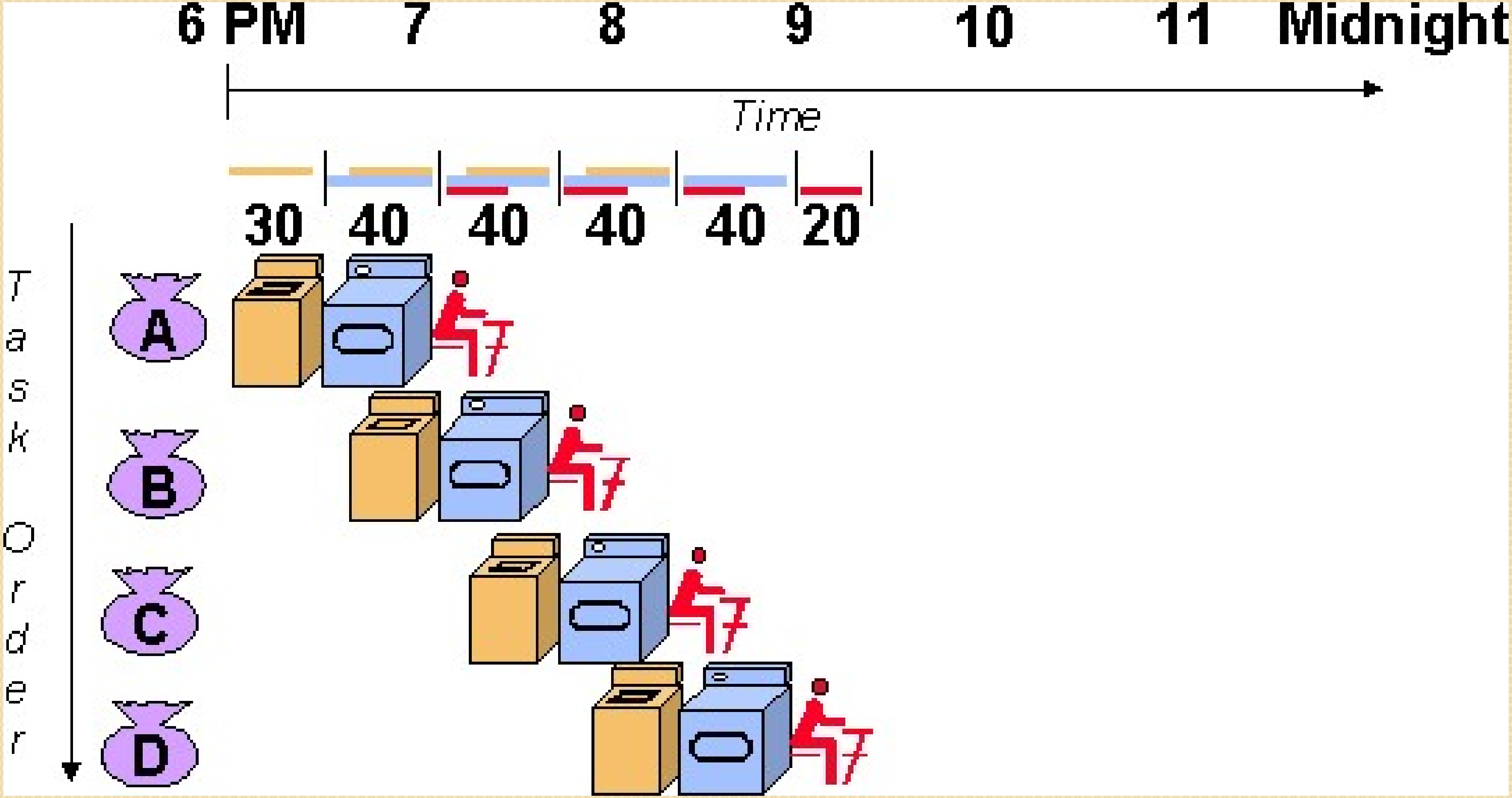
While the clock cycles for the RISC version is:

(3 movs × 1 cycle) + (5 adds × 1 cycle) + (5 loops × 1 cycle) = 13 cycles

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

PIPELINING-Ex Laundry Analogy




6 PM
7
8
9
10
11
Midnight
Time
30
40
40
40
40
20
Task Order
A
B
C
D

Problems with pipeline

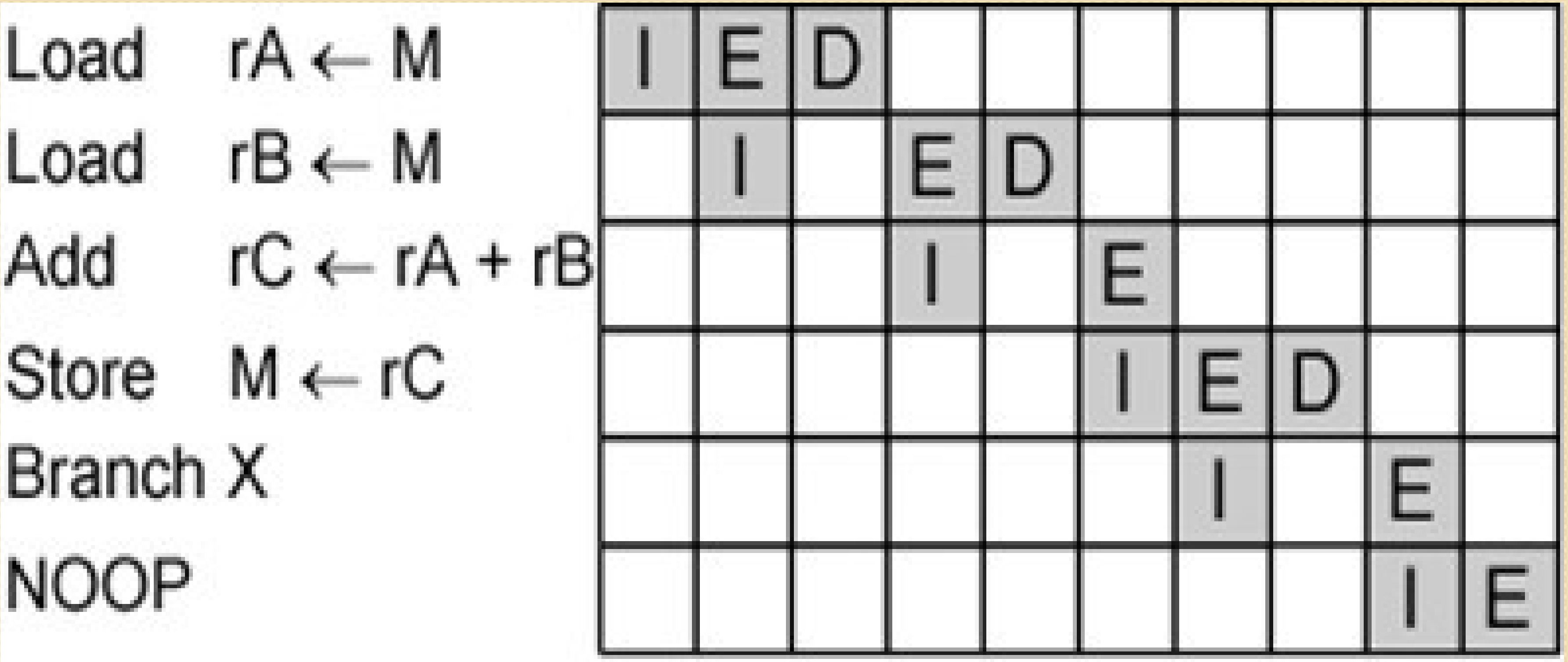
- Stalling of pipeline
  - Data Dependency
  - Branch,etc…

RISC Pipelining

- Most instructions are **register to register**
- Two phases of execution, **I E**
  - I: Instruction fetch
  - E: Execute
    - ALU operation with register input and output
- For **load and store**(**memory**),**I E D**
  - I: Instruction fetch
  - E: Execute
    - Calculate memory address
  - D: Memory
    - Register to memory or memory to register operation

Load	rA ← M	I	E	D										
Load	rB ← M				I	E	D							
Add	rC ← rA + rB							I	E					
Store	M ← rC									I	E	D		
Branch	X												I	E

(a) Sequential execution



(b) Two-stage pipelined timing

NOP is typically used to generate a delay in execution or to reserve space in code memory.

Load	$rA \leftarrow M$	I	E	D					
Load	$rB \leftarrow M$		I	E	D				
NOOP				I	E				
Add	$rC \leftarrow rA + rB$				I	E			
Store	$M \leftarrow rC$					I	E	D	
Branch	X						I	E	
NOOP								I	E

(c) Three-stage pipelined timing

RISC Architecture

- 9 **functional units** interconnected by multiple data paths with width ranging from 32-128 bits
- All internal- external buses are 32 bit wide
- Separate instruction (4KB)and data cache(8KB)
- **MMU-** implements paged virtual memory structure
- **RISC integer unit** executes load,store,fetch etc
- 2 floating point units , multiplier unit and adder unit
- **Graphics unit** to support 3D drawing

CISC Architecture

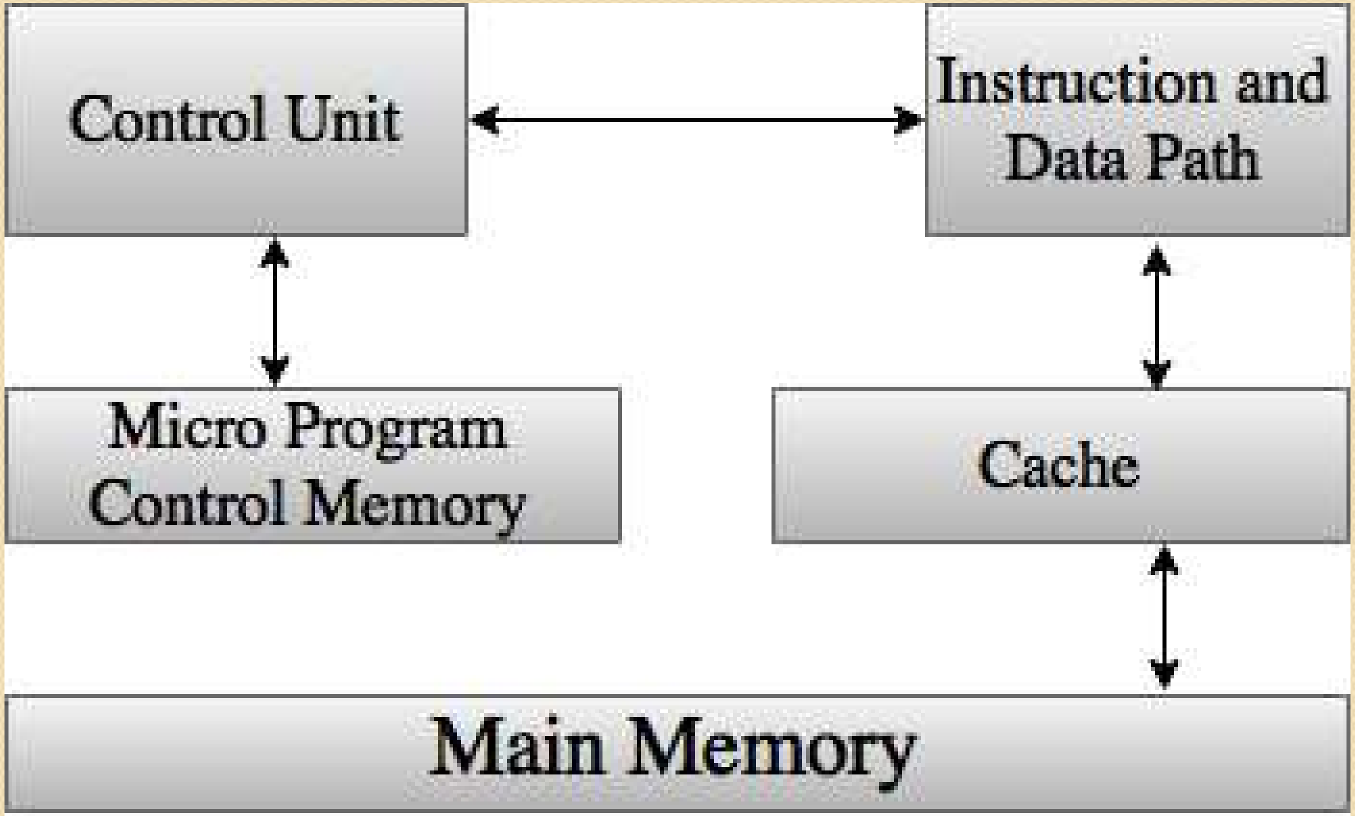


Fig. CISC Architecture