Chapter 1. Basic Structure of Computers

Computer system Architectur

- What is Computer Architecture?
- computer architecture is the science of integrating the components to achieve a level of functionality and performance.

COMPUTER TYPES

Computers are classified based on the parameters like

- Speed of operation
- Cost
- Computational power
- Type of application

DESK TOP COMPUTER

- Processing &storage units, visual display &audio uits, keyboards
- Storage media-Hard disks, CD-ROMs
- Eg: Personal computers which is used in homes and offices
- Advantage: Cost effective, easy to operate, suitable for general purpose educational or business application

NOTEBOOK COMPUTER

- Compact form of personal computer (laptop)
- Advantage is portability

WORK STATIONS

- More computational power than PC
- Costlier
- Used to solve complex problems which arises in engineering application (graphics, CAD/CAM etc)

ENTERPRISE SYSTEM (MAINFRAME)

- More computational power
- Larger storage capacity
- Used for business data processing in large organization
- Commonly referred as servers or super computers

SERVER SYSTEM

- Supports large volumes of data which frequently need to be accessed or to be modified
- Supports request response operation

SUPER COMPUTERS

- Faster than mainframes
- Helps in calculating large scale numerical and algorithm calculation in short span of time
- Used for aircraft design and testing, military application and weather forecasting

HANDHELD

- Also called a PDA (Personal Digital Assistant).
- A computer that fits into a pocket, runs on batteries, and is used while holding the unit in your hand.
- Typically used as an appointment book, address book, calculator, and notepad.
- Can be synchronized with a personal microcomputer as a backup.

Basic Terminology

- Computer
  - A device that accepts input, processes data, stores data, and produces output, all according to a series of stored instructions.
- Hardware
  - Includes the electronic and mechanical devices that process the data; refers to the computer as well as peripheral devices.
- Software
  - A computer program that tells the computer how to perform particular tasks.
- Network
  - Two or more computers and other devices that are connected, for the purpose of sharing data and programs.
- Peripheral devices
  - Used to expand the computer’s input, output and storage capabilities.

Basic Terminology

- Input
  - Whatever is put into a computer system.
- Data
  - Refers to the symbols that represent facts, objects, or ideas.
- Information
  - The results of the computer storing data as bits and bytes; the words, numbers, sounds, and graphics.
- Output
  - Consists of the processing results produced by a computer.
- Processing
  - Manipulation of the data in many ways.
- Memory
  - Area of the computer that temporarily holds data waiting to be processed, stored, or output.
- Storage
  - Area of the computer that holds data on a permanent basis when it is not immediately needed for processing.

Basic Terminology

- Assembly language program (ALP) – Programs are written using mnemonics

- Mnemonic – Instruction will be in the form of English like form

- Assembler – is a software which converts ALP to MLL (Machine Level Language)

- HLL (High Level Language) – Programs are written using English like statements

- Compiler - Convert HLL to MLL, does this job by reading source program at once

Basic Terminology

- Interpreter – Converts HLL to MLL, does this job statement by statement

- System software – Program routines which aid the user in the execution of programs eg: Assemblers, Compilers

- Operating system – Collection of routines responsible for controlling and coordinating all the activities in a computer system

Computing Systems

Computers have two kinds of components:

- *Hardware*, consisting of its physical devices (CPU, memory, bus, storage devices, ...)
- *Software*, consisting of the programs it has (Operating system, applications, utilities, ...)

What is a computer?

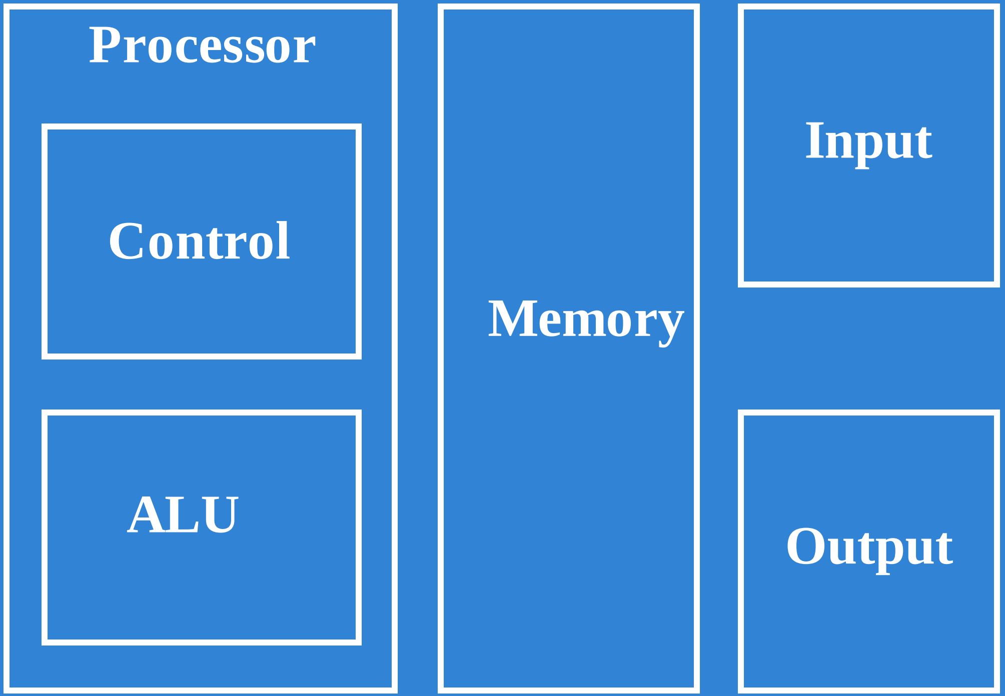
- Simply put, a computer is a sophisticated electronic calculating machine that:
  - Accepts input information,
  - Processes the information according to a list of internally stored instructions and
  - Produces the resulting output information.
- Functions performed by a computer are:
  - Accepting information to be processed as input.
  - Storing a list of instructions to process the information.
  - Processing the information according to the list of instructions.
  - Providing the results of the processing as output.
- What are the functional units of a computer?

Functional Units

FUNCTIONAL UNITS OF COMPUTER

- Input Unit
- Output Unit
- Central processing Unit (ALU and Control Units)
- Memory
- Bus Structure

The Big Picture

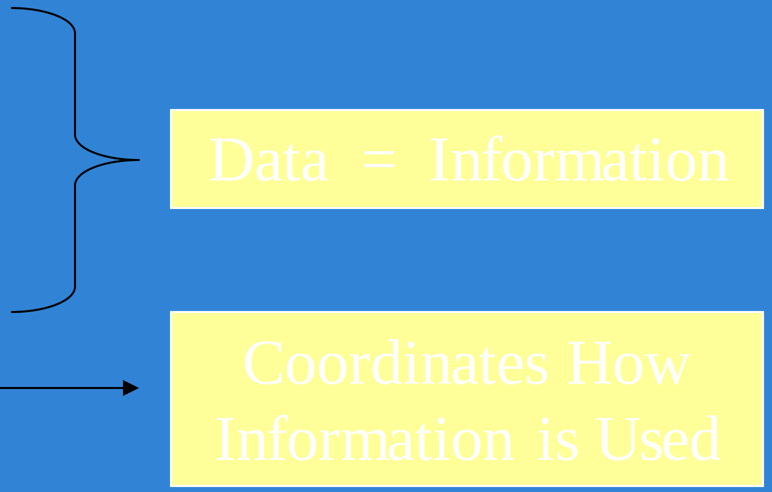


Since 1946 all computers have had 5 components!!!

IMPORTANT SLIDE !

Function

- ALL computer functions are:
  - Data PROCESSING
  - Data STORAGE
  - Data MOVEMENT
  - CONTROL



Data = Information

Coordinates How Information is Used

- NOTHING ELSE!

INPUT UNIT:

- Converts the external world data to a binary format, which can be understood by CPU

- Eg: Keyboard, Mouse, Joystick etc

OUTPUT UNIT:

- Converts the binary format data to a format that a common man can understand

- Eg: Monitor, Printer, LCD, LED etc

CPU

- The “brain” of the machine

- Responsible for carrying out computational task

- Contains ALU, CU, Registers

- ALU Performs Arithmetic and logical operations

- CU Provides control signals in accordance with some timings which in turn controls the execution process

- Register Stores data and result and speeds up the operation

Example

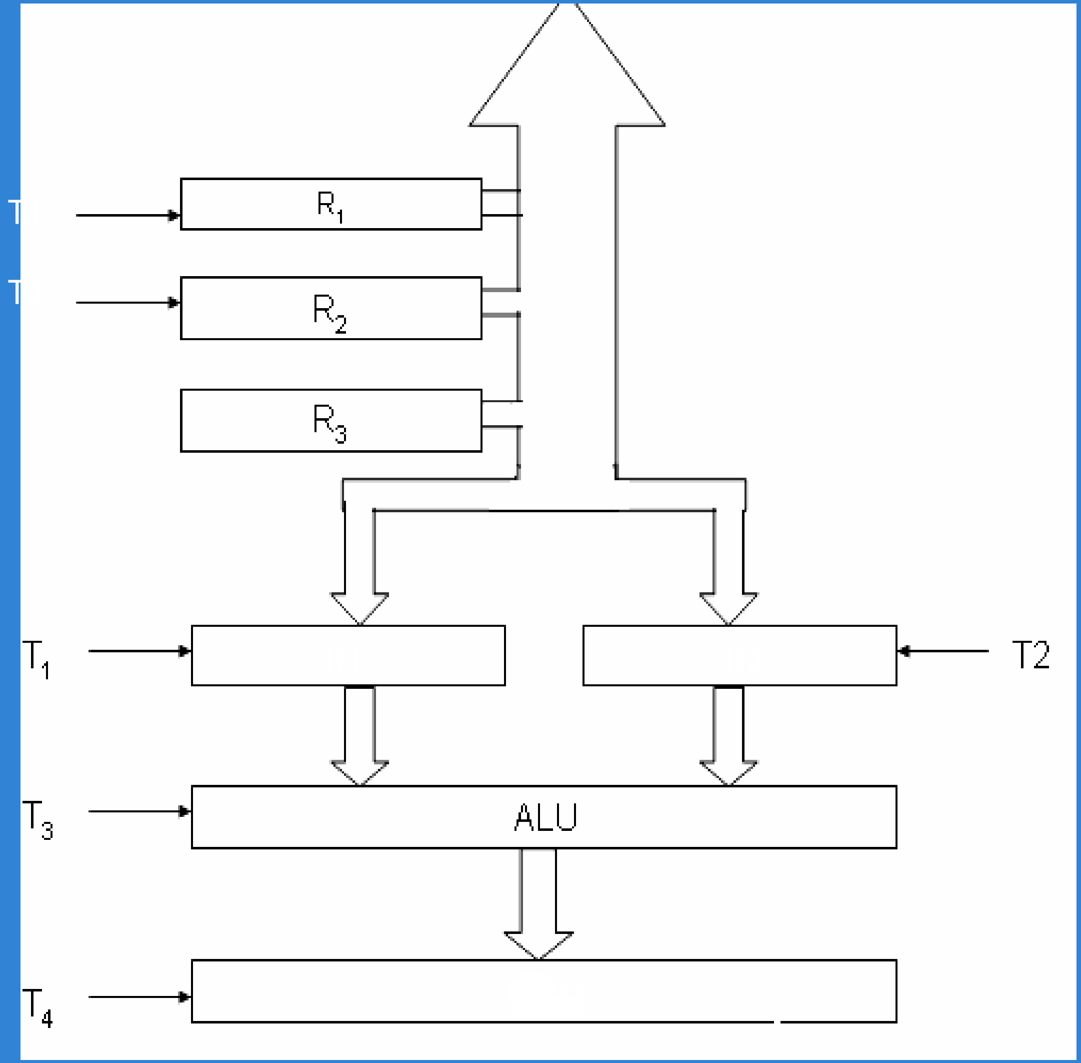
Add R1, R2

T1 → Enable R1

T2 → Enable R2

T3 → Enable ALU for addition operation

T4 → Enable out put of ALU to store result of the operation



- Control unit works with a reference signal called processor clock

- Processor divides the operations into basic steps

- Each basic step is executed in one clock cycle

MEMORY

- Stores data, results, programs

- Two class of storage

(i) Primary (ii) Secondary

- Two types are RAM or R/W memory and ROM read only memory

- ROM is used to store data and program which is not going to change.

- Secondary storage is used for bulk storage or mass storage

Basic Operational Concepts

Basic Function of Computer

- To Execute a given task as per the appropriate program
- Program consists of list of instructions stored in memory

Review

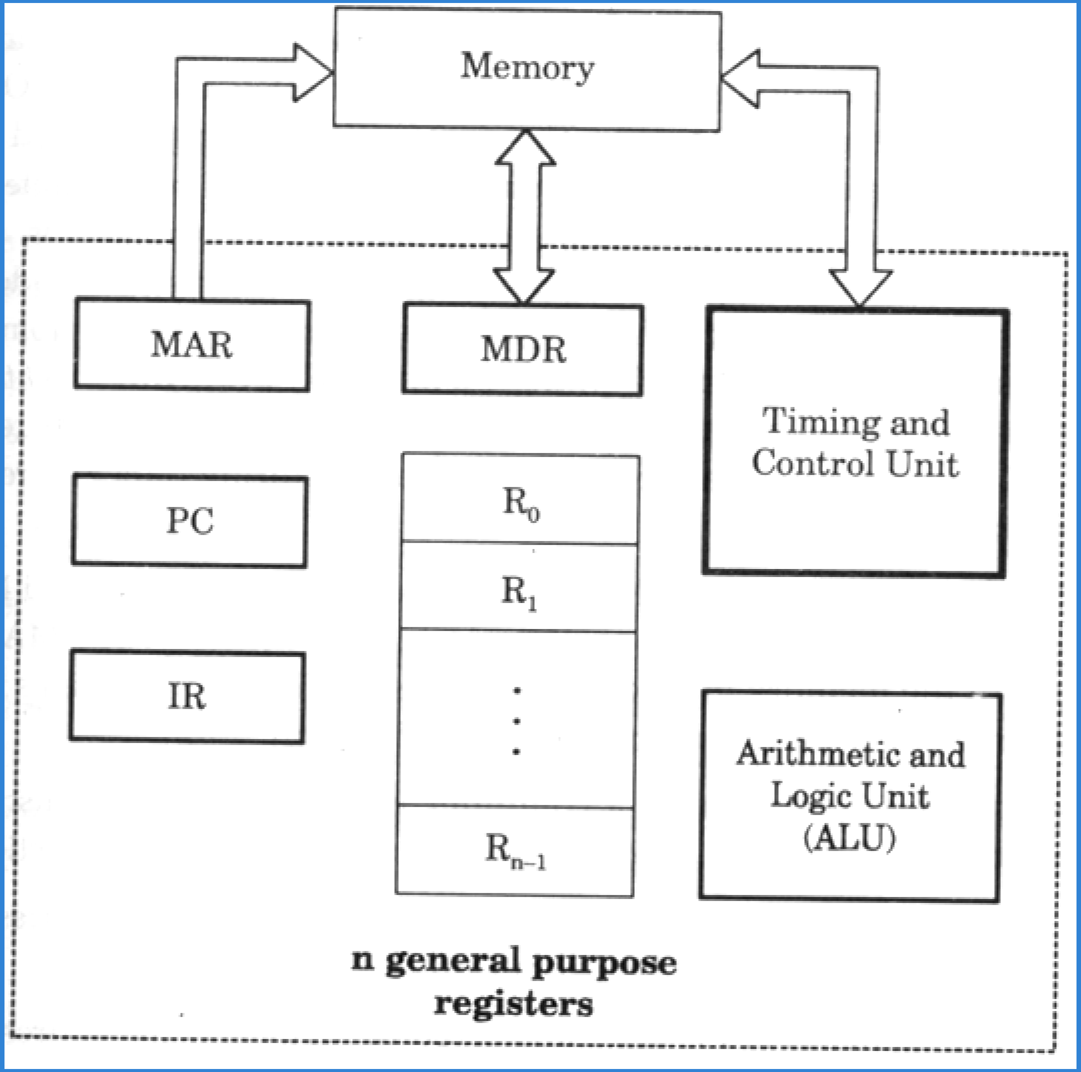
- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.

A Typical Instruction

- Add LOCA, R0
- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

Separate Memory Access and ALU Operation

- Load LOCA, R1
- Add R1, R0
- Whose contents will be overwritten?



Interconnection between Processor and Memory

Registers

Registers are fast stand-alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU. Some of these registers are

- ❑ **Two registers-MAR (Memory Address Register) and MDR (Memory Data Register) : To handle the data transfer between main memory and processor. MAR-Holds addresses, MDR-Holds data**

- ❑ **Instruction register (IR) : Hold the Instructions that is currently being executed**

- ❑ **Program counter: Points to the next instructions that is to be fetched from memory**

- (PC) → (MAR)(the contents of PC transferred to MAR)

- (MAR) → (Address bus) Select a particular memory location

- Issues RD control signals

- Reads instruction present in memory and loaded into MDR

- Will be placed in IR (Contents transferred from MDR to IR)

- Instruction present in IR will be decoded by which processor understand what operation it has to perform

- Increments the contents of PC by 1, so that it points to the next instruction address

- If data required for operation is available in register, it performs the operation

- If data is present in memory following sequence is performed

- Address of the data → MAR

- MAR → Address bus → select memory location where is issued RD signal

- Reads data via data bus → MDR

- From MDR data can be directly routed to ALU or it can be placed in register and then operation can be performed

- Results of the operation can be directed towards output device, memory or register

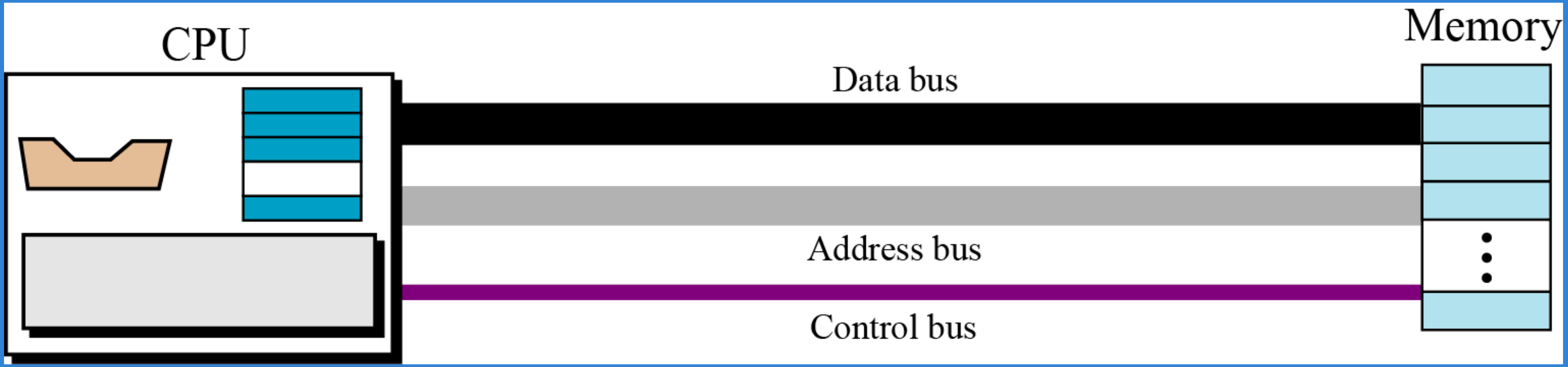
- Normal execution preempted (interrupt)

Interrupt

- An interrupt is a request from I/O device for service by processor
- Processor provides requested service by executing interrupt service routine (ISR)
- Contents of PC, general registers, and some control information are stored in memory .
- When ISR completed, processor restored, so that interrupted program may continue

BUS STRUCTURE
Connecting CPU and memory

The CPU and memory are normally connected by three groups of connections, each called a **bus**: *data bus*, *address bus* and *control bus*

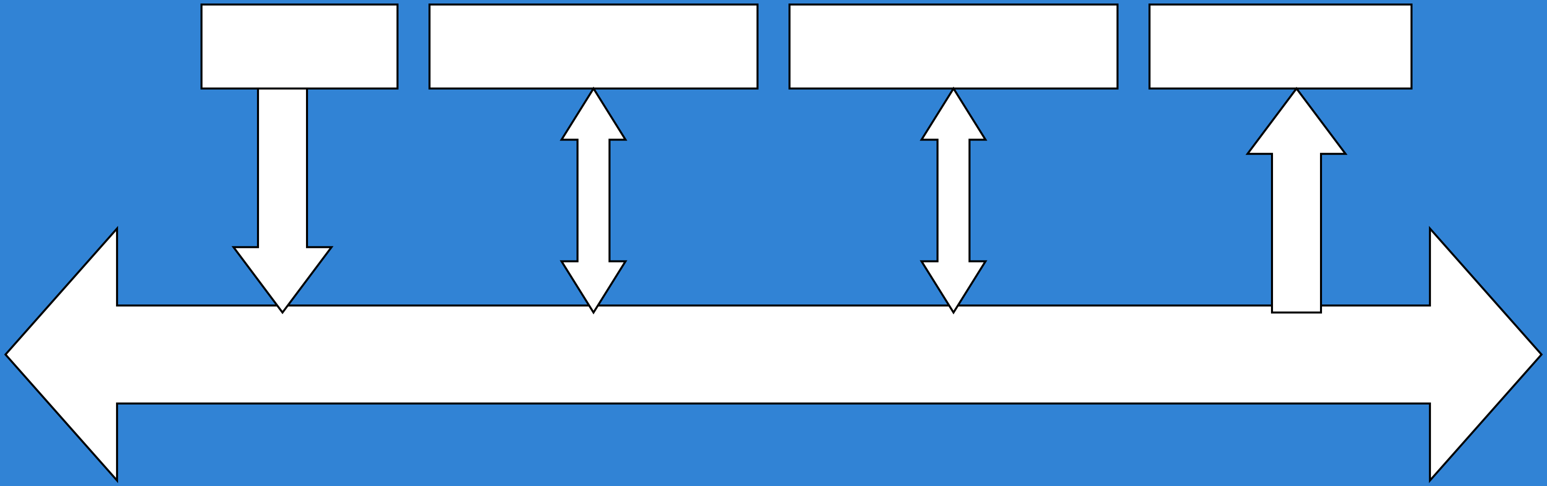


Connecting CPU and memory using three buses

BUS STRUCTURE

- Group of wires which carries information form CPU to peripherals or vice – versa

- **Single bus structure**: Common bus used to communicate between peripherals and microprocessor



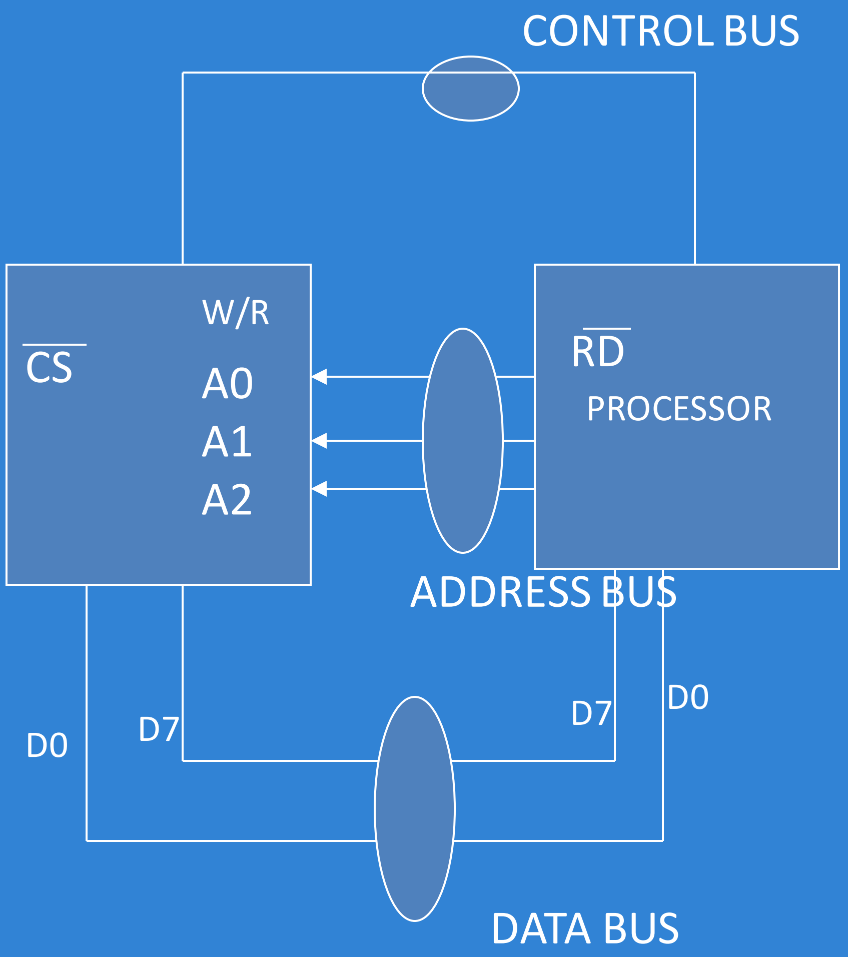
SINGLE BUS STRUCTURE

Continued:-

- To improve performance **multibus** structure can be used

- In two – bus structure : One bus can be used to fetch instruction other can be used to fetch data, required for execution.

- Thus improving the performance ,but cost increases



A_2	A_1	A_0	Selected location
0	0	0	0th Location
0	0	1	1st Location
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Cont:-

- 2^3 = 8 i.e. 3 address line is required to select 8 location

- In general 2^x = n where x number of address lines (address bit) and n is number of location

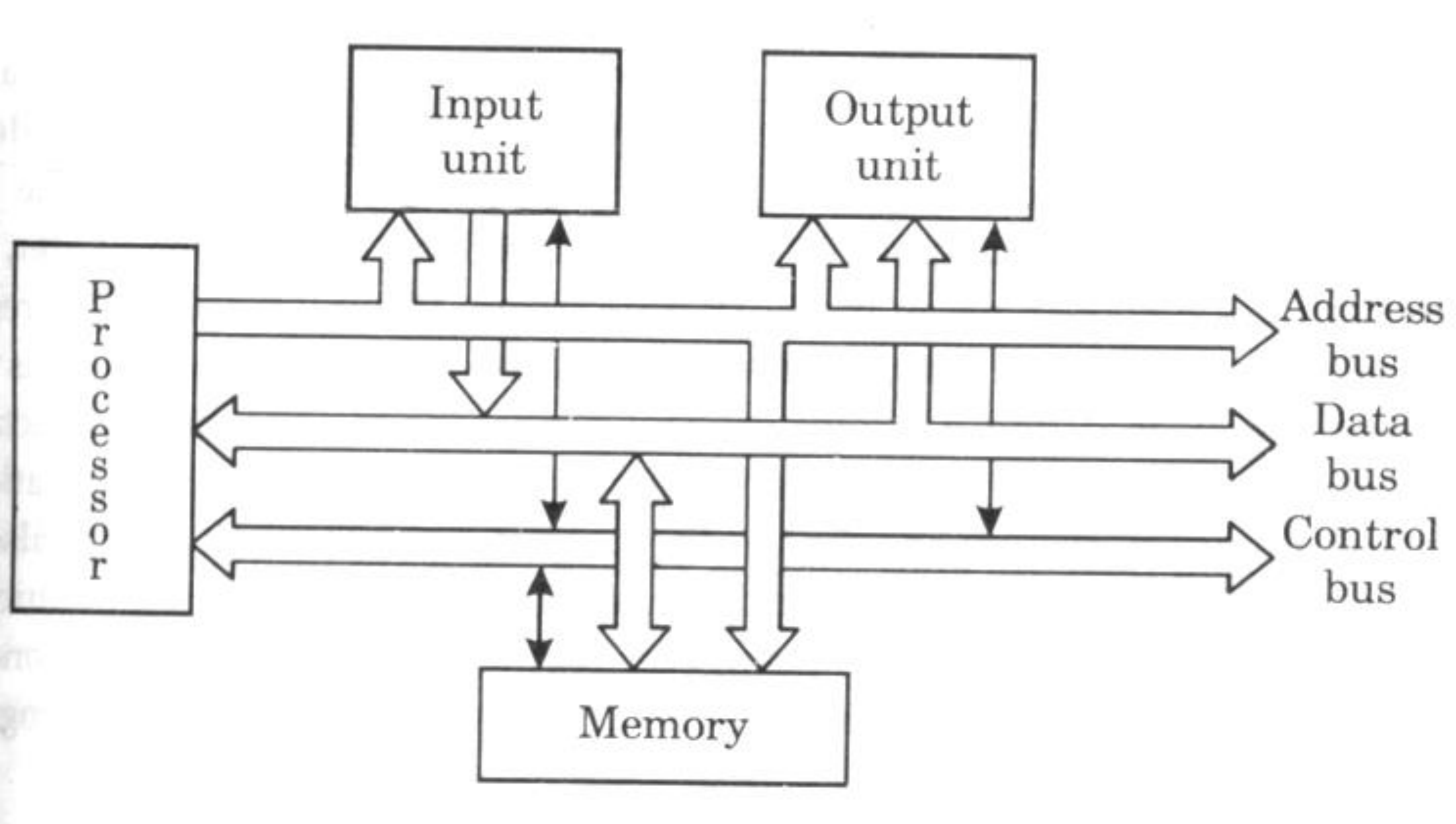
- **Address bus** : unidirectional : group of wires which carries address information bits form processor to peripherals (16,20,24 or more parallel signal lines)

Cont:-

- **Databus**: bidirectional : group of wires which carries data information bit form processor to peripherals and vice – versa

- **Controlbus**: bidirectional: group of wires which carries control signals form processor to peripherals and vice – versa

- Figure below shows address, data and control bus and their connection with peripheral and microprocessor



Single bus structure showing the details of connection

Memory Locations, Addresses, and Operations

Memory Location, Addresses, and Operation

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups. *n* is called word length.

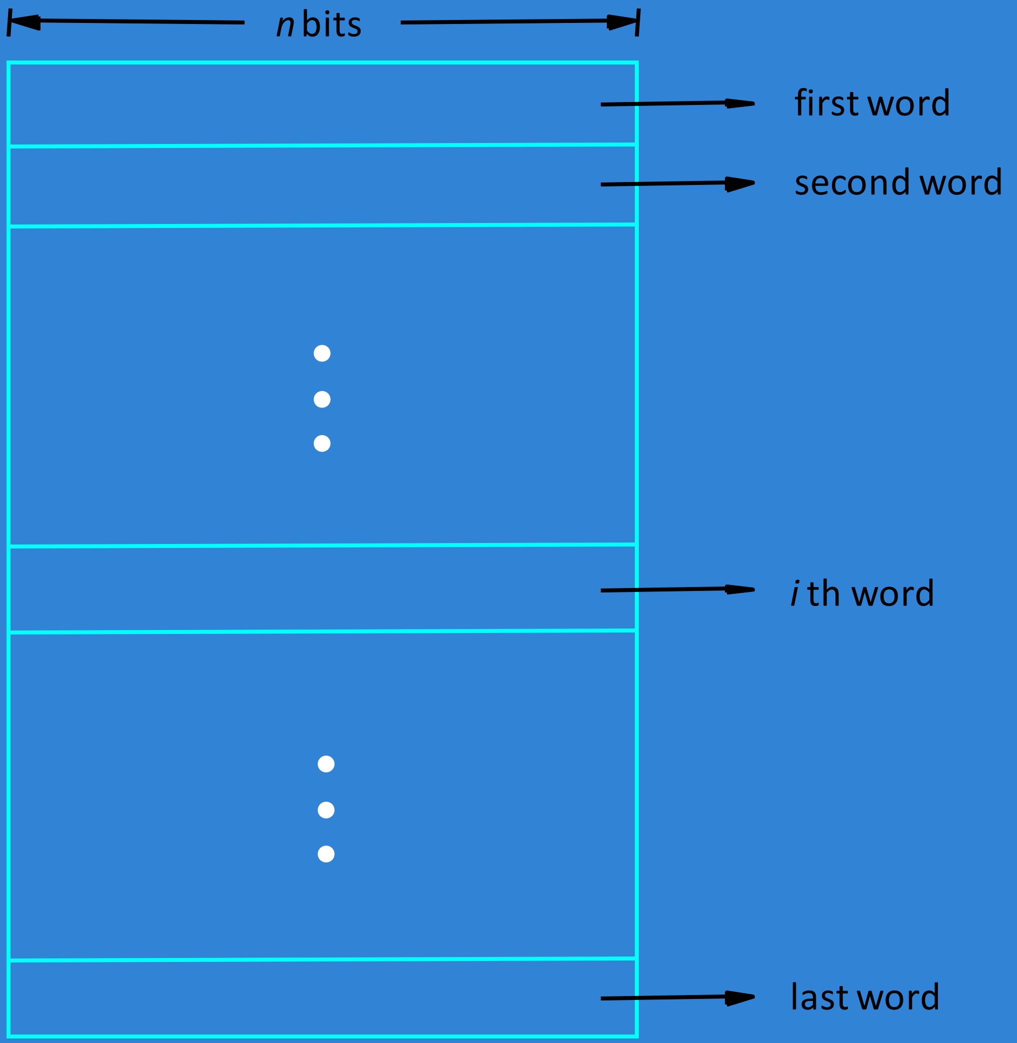


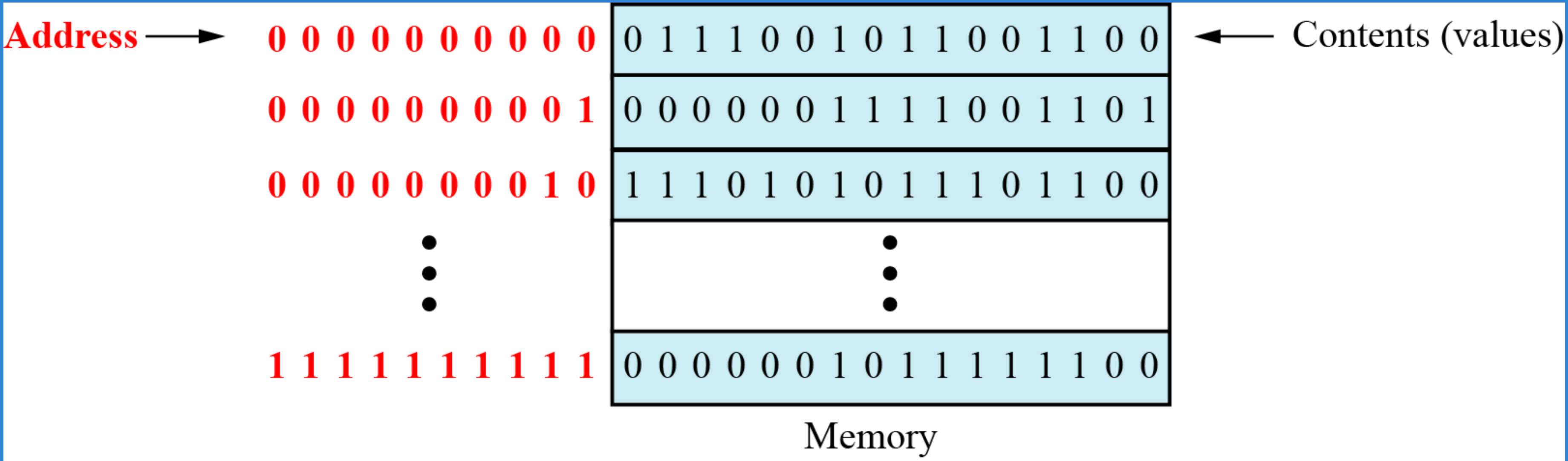
Figure 2.5. Memory words.

MEMORY LOCATIONS AND ADDRESSES

•**Main memory** is the second major subsystem in a computer. It consists of a collection of storage locations, each with a unique identifier, called an **address**.

•Data is transferred to and from memory in groups of bits called **words**. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing).

•If the word is 8 bits, it is referred to as a **byte**. The term "byte" is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word, or a 32-bit word is referred to as a 4-byte word.



Address	Contents (values)
0 0 0 0 0 0 0 0 0 0	0 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 1
0 0 0 0 0 0 0 0 1 0	1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0
⋮	⋮
1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0

Memory

Figure 5.3 **Main memory**

Address space

- To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address.

- The total number of uniquely identifiable locations in memory is called the **address space**.

- For example, a memory with 64 kilobytes (16 address line required) and a word size of 1 byte has an address space that ranges from 0 to 65,535.

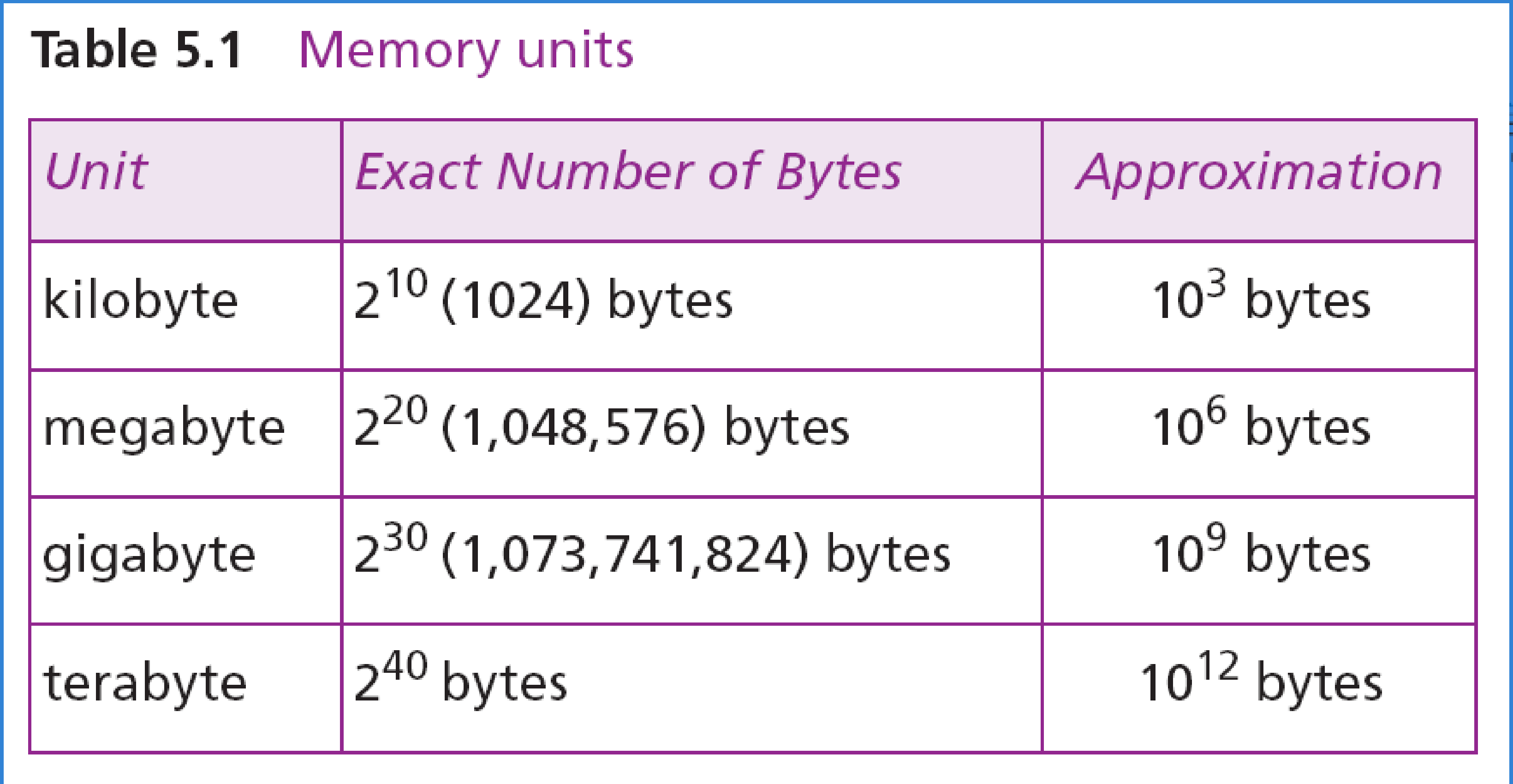


Table 5.1 Memory units

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^{3} bytes
megabyte	2^{20} (1,048,576) bytes	10^{6} bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^{9} bytes
terabyte	2^{40} bytes	10^{12} bytes

Memory addresses are defined using unsigned binary integers.

Example 1

A computer has 32 MB (megabytes) of memory. How many bits are needed to address any single byte in memory?

Solution

The memory address space is 32 MB, or 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$, or **25 bits**, to address each byte.

Example 2

A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?

Solution

The memory address space is 128 MB, which means 2^{27}. However, each word is eight (2^3) bytes, which means that we have 2^{24} words. This means that we need $\log_2 2^{24}$, or **24 bits**, to address each word.

MEMORY OPERATIONS

- Today, **general-purpose computers** use a set of instructions called a **program** to process data.
- A computer executes the program to create output data from input data
- Both program instructions and data operands are stored in memory
- Two basic operations requires in memory access
  - Load operation (Read or Fetch)-Contents of specified memory location are read by processor
  - Store operation (Write)- Data from the processor is stored in specified memory location

Assignment of Byte Address

- **Big**-**endian** and **little**-**endian** are terms that describe the order in which a sequence of bytes are stored in computer **memory**. **Big-endian** is an order in which the "**big**end" (most significant value in the sequence) is stored first (at the lowest storage address).
-

Assignment of byte addresses

- Little Endian (e.g., in DEC, Intel)
  » low order byte stored at lowest address
  » byte0 byte1 byte2 byte3

- Eg: 46,78,96,54 (32 bit data)
- H BYTE ← L BYTE

- 8000
- 8001
- 8002
- 8003
- 8004

54
96
78
46
|

Big Endian

- Big Endian (e.g., in IBM, Motorolla, Sun, HP)

  » high order byte stored at lowest address

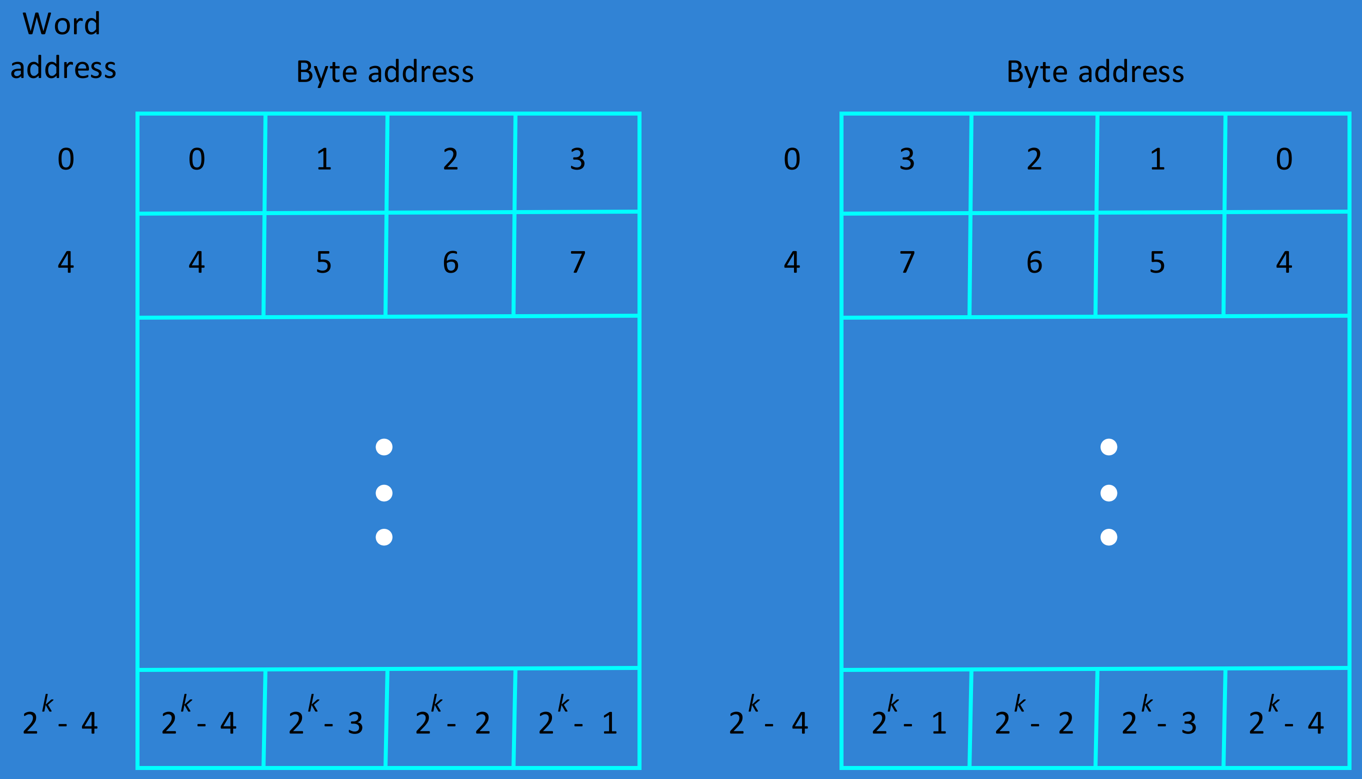
  » byte3 byte2 byte1 byte0

- Programmers/protocols should be careful when transferring binary data between Big Endian and Little Endian machines

Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word



Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
• • •				
$2^k - 4$	$2^k - 4$	$2^k - 3$	$2^k - 2$	$2^k - 1$

(a) Big-endian assignment

	Byte address			
0	3	2	1	0
4	7	6	5	4
• • •				
$2^k - 4$	$2^k - 1$	$2^k - 2$	$2^k - 3$	$2^k - 4$

(b) Little-endian assignment

Figure 2.7. Byte and word addressing.

- In case of 16 bit data, aligned words begin at byte addresses of 0,2,4,...............................
- In case of 32 bit data, aligned words begin at byte address of 0,4,8,...............................
- In case of 64 bit data, aligned words begin at byte addresses of 0,8,16,.............................
- In some cases words can start at an arbitrary byte address also then, we say that word locations are unaligned

Instruction and Instruction Sequencing

- INSTRUCTION SET ARCHITECTURE:-Complete instruction set of the processor

- BASIC 4 TYPES OF OPERATION:-
    - Data transfer between memory and processor register
    - Arithmetic and logic operation
    - Program sequencing and control
    - I/O transfer

Register transfer notation (RTN)

Transfer between processor registers & memory, between processor register & I/O devices

Memory locations, registers and I/O register names are identified by a symbolic name in uppercase alphabets

- LOC,PLACE,MEM are the address of memory location
- R1 , R2,… are processor registers
- DATA_IN, DATA_OUT are I/O registers

- Contents of location is indicated by using square brackets [ ]
- RHS of RTN always denotes a values, and is called Source
- LHS of RTN always denotes a symbolic name where value is to be stored and is called destination
- Source contents are not modified
- Destination contents are overwritten

Examples of RTN statements

- R2 ← [LOCN]
- R4 ← [R3] +[R2]

ASSEMBLY LANGUAGE NOTATION (ALN)

- RTN is easy to understand and but cannot be used to represent machine instructions
- Mnemonics can be converted to machine language, which processor understands using assembler

Eg:

1. MOVE LOCN, R2
2. ADD   R3, R2, R4

TYPES OF INSTRUCTION

- Three address instruction

  - Syntax: Operation source 1, source 2, destination
  - Eg: ADD D,E,F   where D,E,F are memory location
  - Advantage: Single instruction can perform the complete operation
  - Disadvantage : Instruction code will be too large to fit in one word location in memory

TWO ADDRESS INSTRUCTION

- Syntax : Operation source, destination
- Eg: MOVE E,F MOVE D,F

ADD D,F OR ADD E,F

Perform ADD A,B,C using 2 instructions
MOVE B,C
ADD A,C

❖Disadvantage: Single instruction is not sufficient to perform the entire operation.

ONE ADDRESS INSTRUCTION

- Syntax- Operation source/destination
- In this type either a source or destination operand is mentioned in the instruction
- Other operand is implied to be a processor register called Accumulator
- Eg: ADD B (general)
- Load D; ACC ← [memlocation _D]
- ADD E; ACC ← (ACC) +(E)
- STORE F; memlocation_ F ← (ACC)

Zero address instruction

- Location of all operands are defined implicitly
- Operands are stored in a structure called pushdown stack

Continued

- If processor supports ALU operations one data in memory and other in register then the instruction sequence is
- MOVE D, Ri
- ADD E, Ri
- MOVE Ri, F

- If processor supports ALU operations only with registers then one has to follow the instruction sequence given below
- LOAD D, Ri
- LOAD E, Rj
- ADD Ri, Rj
- MOVE Rj, F

Instruction Formats

Example: Evaluate (A+B) ∗ (C+D)

- Three-Address

1. ADD R1, A, B ; R1 ← M[A] + M[B]
2. ADD R2, C, D ; R2 ← M[C] + M[D]
3. MUL X, R1, R2 ; M[X] ← R1 ∗ R2

Instruction Formats

Example: Evaluate (A+B) ∗ (C+D)

- Two-Address

```
1. MOV   R1, A          ; R1 ← M[A]
2. ADD   R1, B          ; R1 ← R1 + M[B]
3. MOV   R2, C          ; R2 ← M[C]
4. ADD   R2, D          ; R2 ← R2 + M[D]
5. MUL   R1, R2         ; R1 ← R1 ∗ R2
6. MOV   X, R1          ; M[X] ← R1
```

Instruction Formats

Example: Evaluate (A+B) ∗ (C+D)

- One-Address

```
1.  LOAD    A        ; AC ← M[A]
2.  ADD     B        ; AC ← AC + M[B]
3.  STORE   T        ; M[T] ← AC
4.  LOAD    C        ; AC ← M[C]
5.  ADD     D        ; AC ← AC + M[D]
6.  MUL     T        ; AC ← AC ∗ M[T]
7.  STORE   X        ; M[X] ← AC
```

Instruction Formats

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

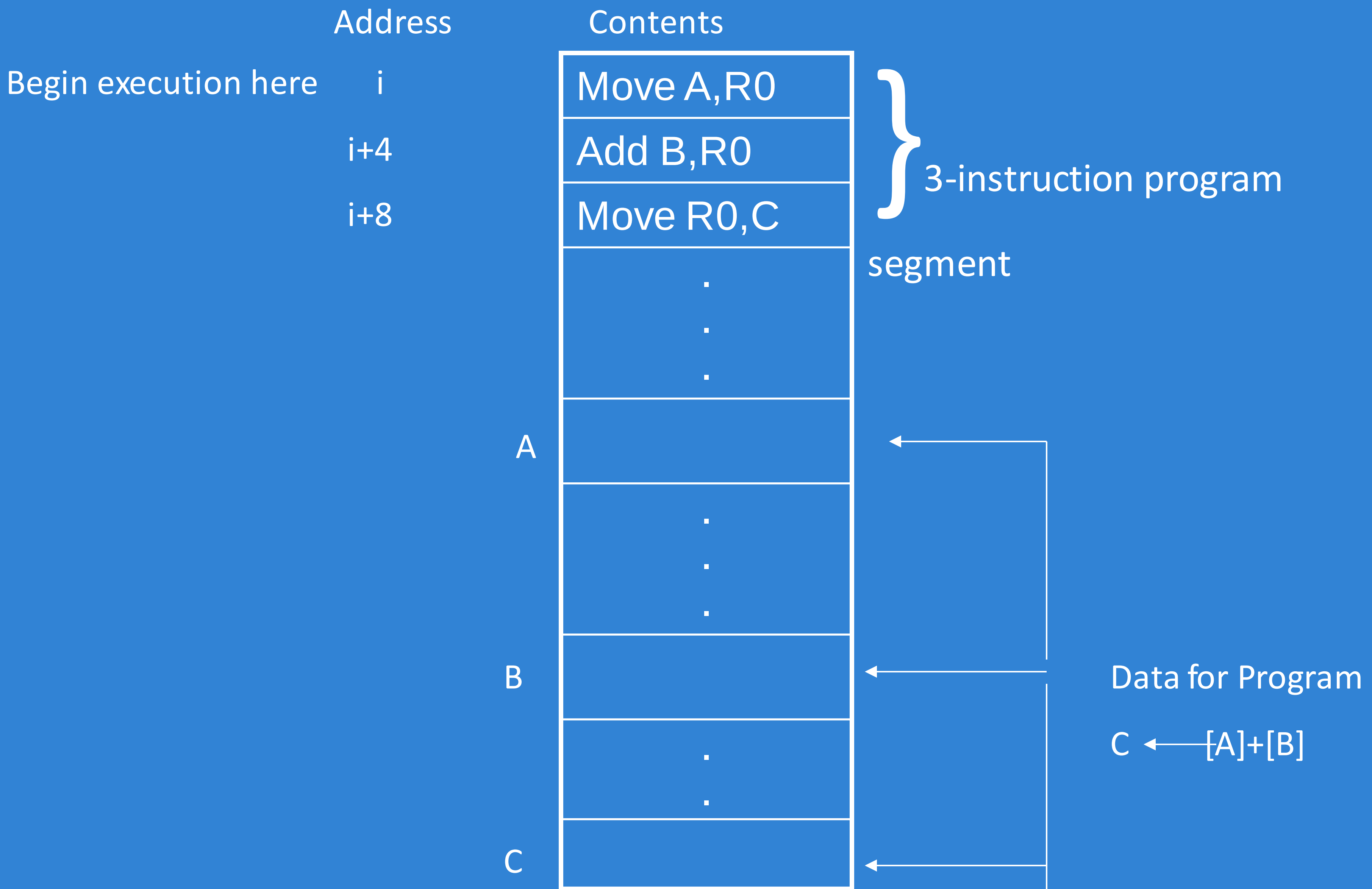
- Zero-Address

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

Basic Instruction Cycle

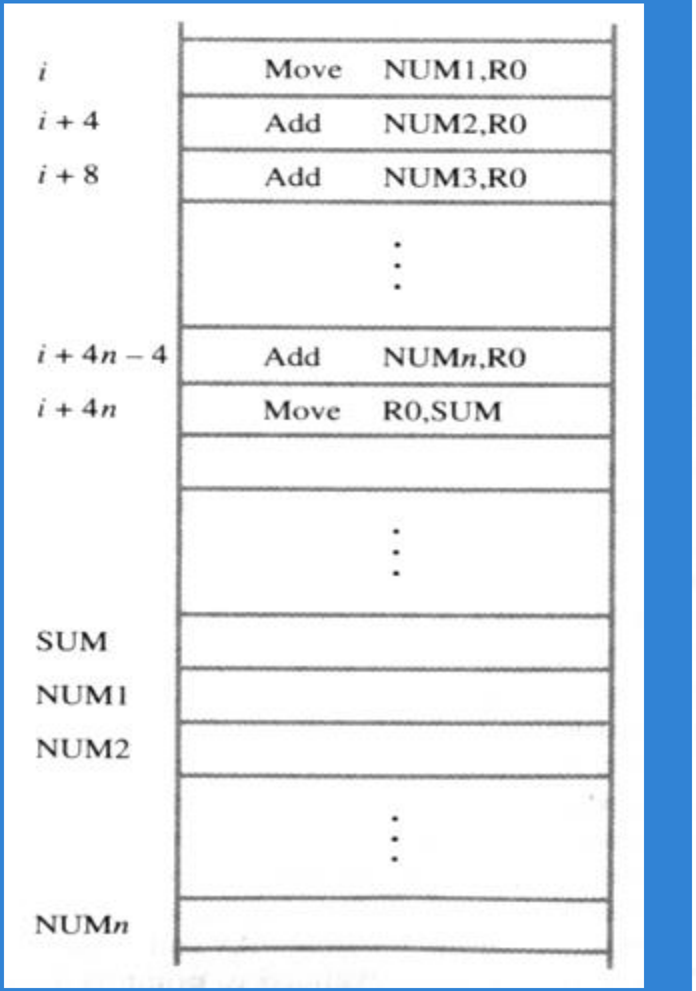
- Basic computer operation cycle
  - Fetch the instruction from memory into a control register (PC)
  - Decode the instruction
  - Locate the operands used by the instruction
  - Fetch operands from memory (if necessary)
  - Execute the operation in processor registers
  - Store the results in the proper place
  - Go back to step 1 to fetch the next instruction

INSTRUCTION EXECUTION & STRIAGHT LINE SEQUENCING



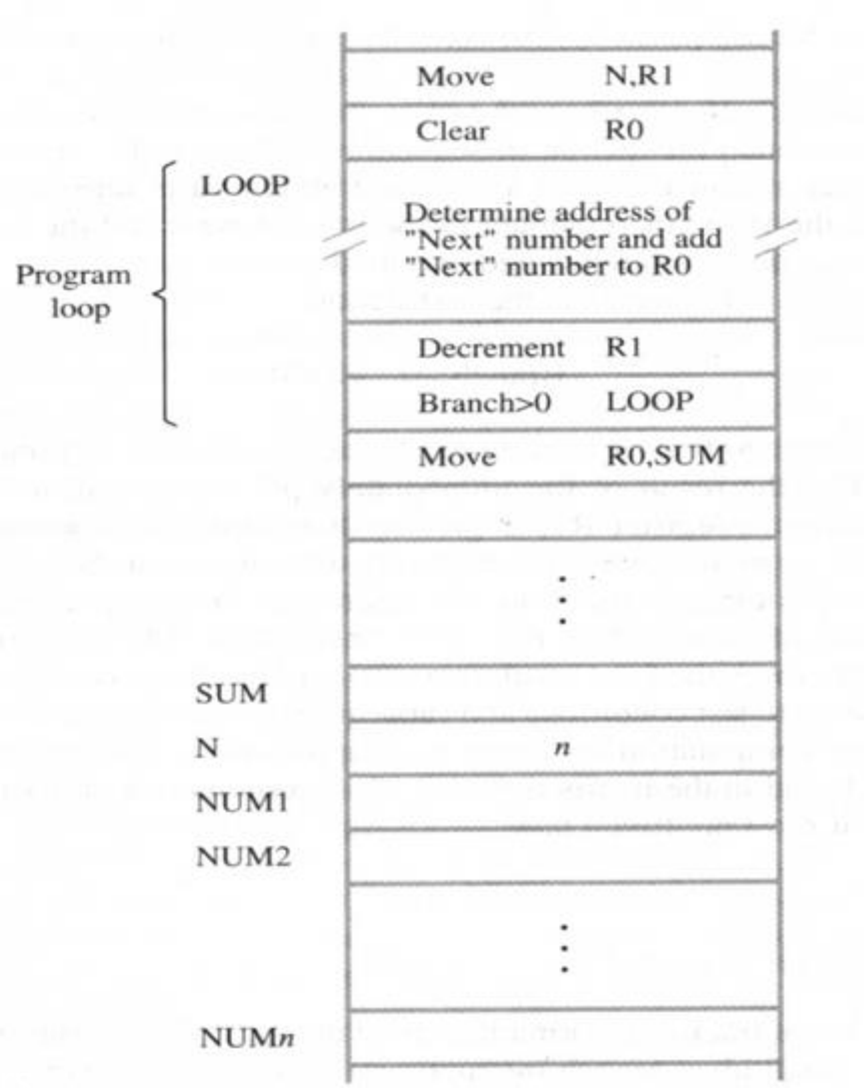
- PC – Program counter: hold the address of the next instruction to be executed
- Straight line sequencing: If fetching and executing of instructions is carried out one by one from successive addresses of memory, it is called straight line sequencing.
- Major two phase of instruction execution
- Instruction fetch phase: Instruction is fetched form memory and is placed in instruction register IR
- Instruction execute phase: Contents of IR is decoded and processor carries out the operation either by reading data from memory or registers.

BRANCHING



i	Move	NUM1,R0
$i+4$	Add	NUM2,R0
$i+8$	Add	NUM3,R0
	⋮	
$i+4n-4$	Add	NUMn,R0
$i+4n$	Move	R0,SUM
	⋮	
SUM		
NUM1		
NUM2		
	⋮	
NUMn		

A straight line program for adding n numbers



		Move	N,R1
		Clear	R0
Program loop	LOOP	Determine address of "Next" number and add "Next" number to R0	
		Decrement	R1
		Branch>0	LOOP
		Move	R0,SUM
		⋮	
	SUM		
	N	n	
	NUM1		
	NUM2		
		⋮	
	NUMn		

Using a loop to add n numbers

BRANCHING

- Branch instruction are those which changes the normal sequence of execution.
- Sequence can be changed either conditionally or unconditionally.
- Accordingly we have conditional branch instructions and unconditional branch instruction.
- Conditional branch instruction changes the sequence only when certain conditions are met.
- Unconditional branch instruction changes the sequence of execution irrespective of condition of the results.

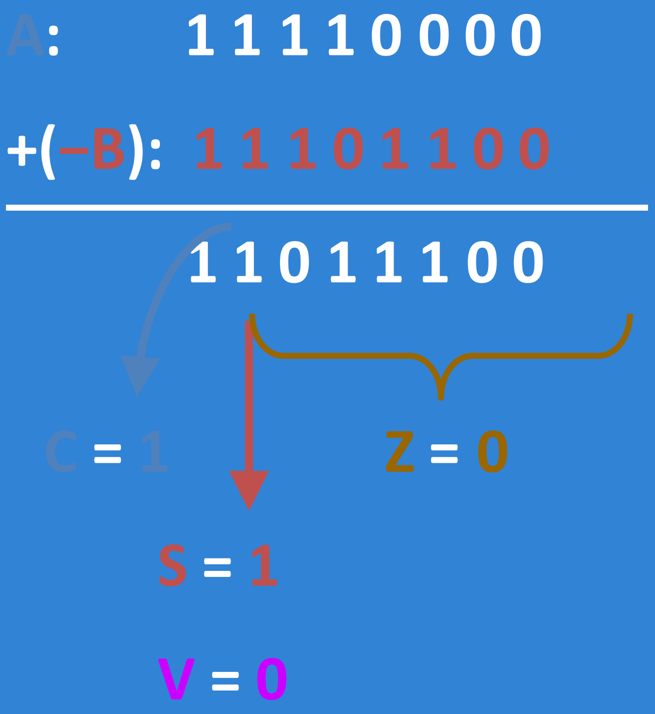
CONDITION CODES

➢ CONDITIONAL CODE FLAGS: The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions

- N – Negative 1 if results are Negative
  0 if results are Positive
- Z – Zero 1 if results are Zero
  0 if results are Non zero
- V – Overflow 1 if arithmetic overflow occurs
  0 non overflow occurs
- C – Carry 1 if carry and from MSB bit
  0 if there is no carry from MSB bit

Conditional Branch Instructions

- Example:
  - A: 1 1 1 1 0 0 0 0
  - B: 0 0 0 1 0 1 0 0



A: 1 1 1 1 0 0 0 0

+(−B): 1 1 1 0 1 1 0 0

1 1 0 1 1 1 0 0

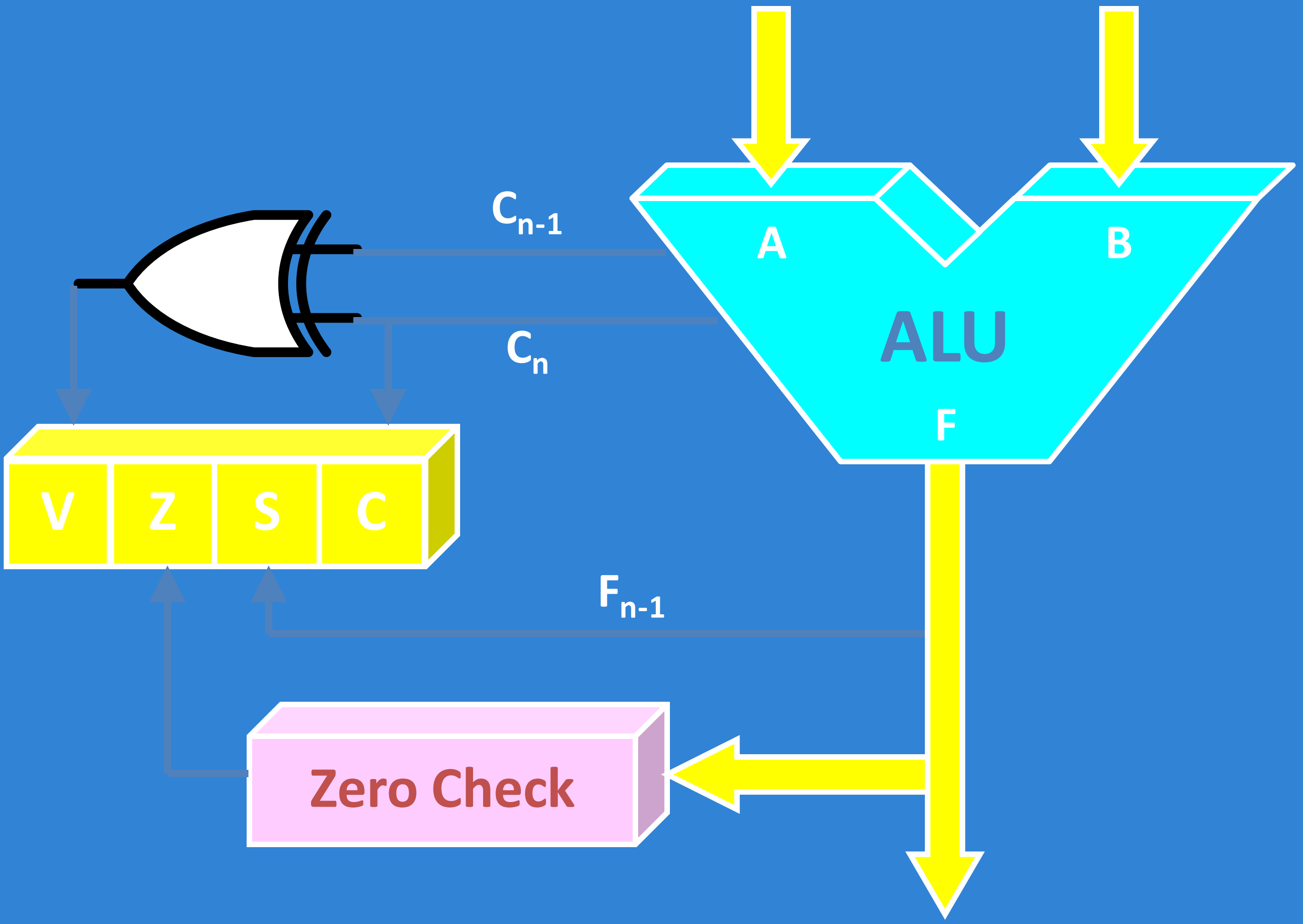
C = 1

Z = 0

S = 1

V = 0

Status Bits



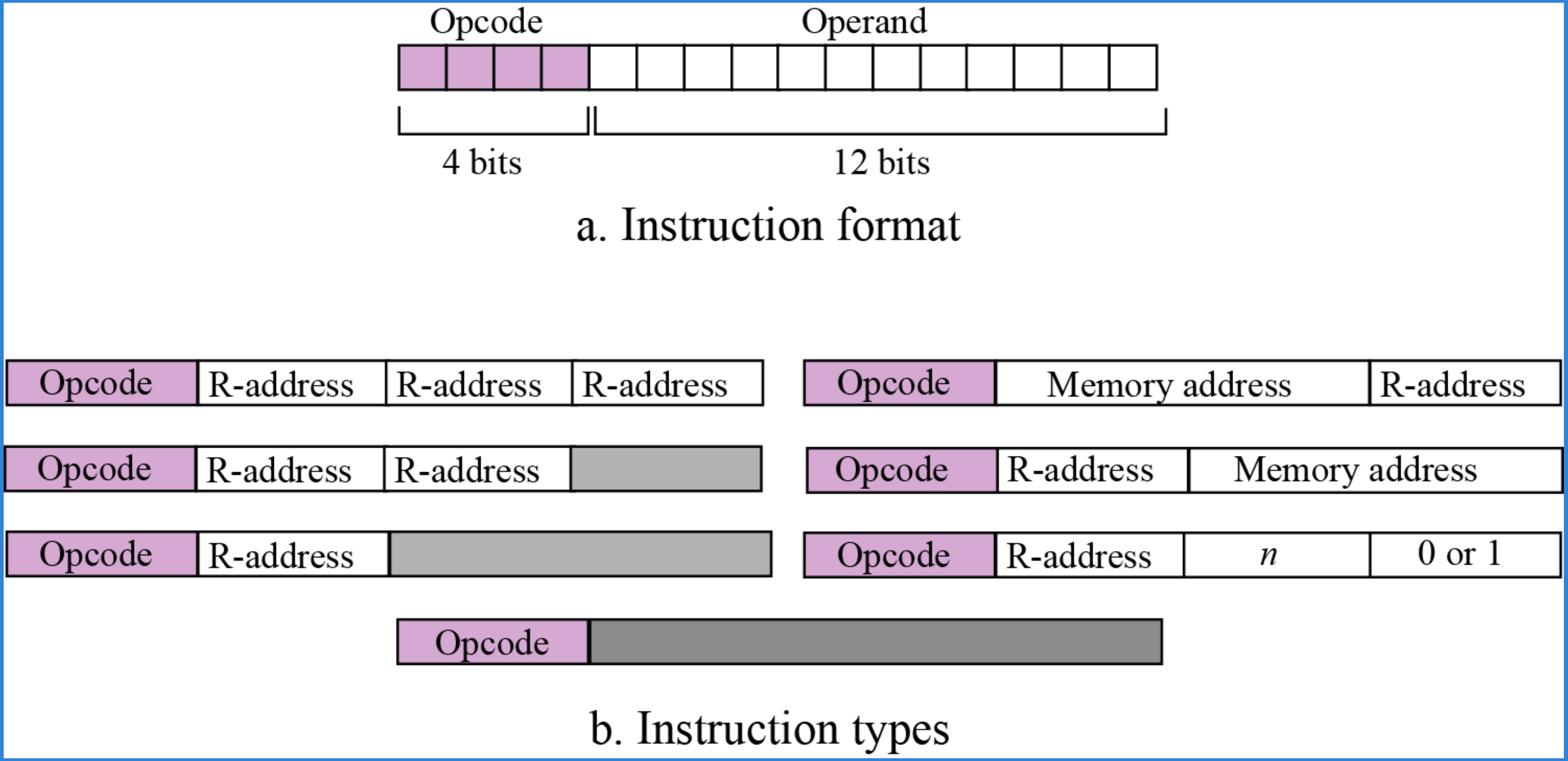


Figure **Format and different instruction types**

Processing the instructions

Simple computer, like most computers, uses machine cycles.

A cycle is made of three phases: fetch, decode and execute.

During the fetch phase, the instruction whose address is determined by the PC is obtained from the memory and loaded into the IR. The PC is then incremented to point to the next instruction.

During the decode phase, the instruction in IR is decoded and the required operands are fetched from the register or from memory.

During the execute phase, the instruction is executed and the results are placed in the appropriate memory location or the register.

Once the third phase is completed, the control unit starts the cycle again, but now the PC is pointing to the next instruction.

The process continues until the CPU reaches a HALT instruction.

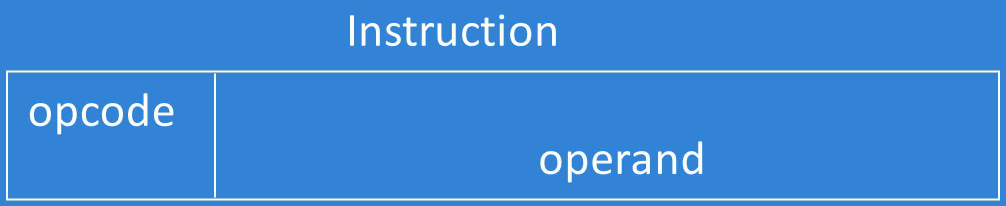
Types of Addressing Modes

The different ways in which the location of the operand is specified in an instruction are referred to as addressing modes

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Relative Addressing
- Indexed Addressing

Immediate Addressing

- Operand is given explicitly in the instruction
- Operand = Value
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



Instruction

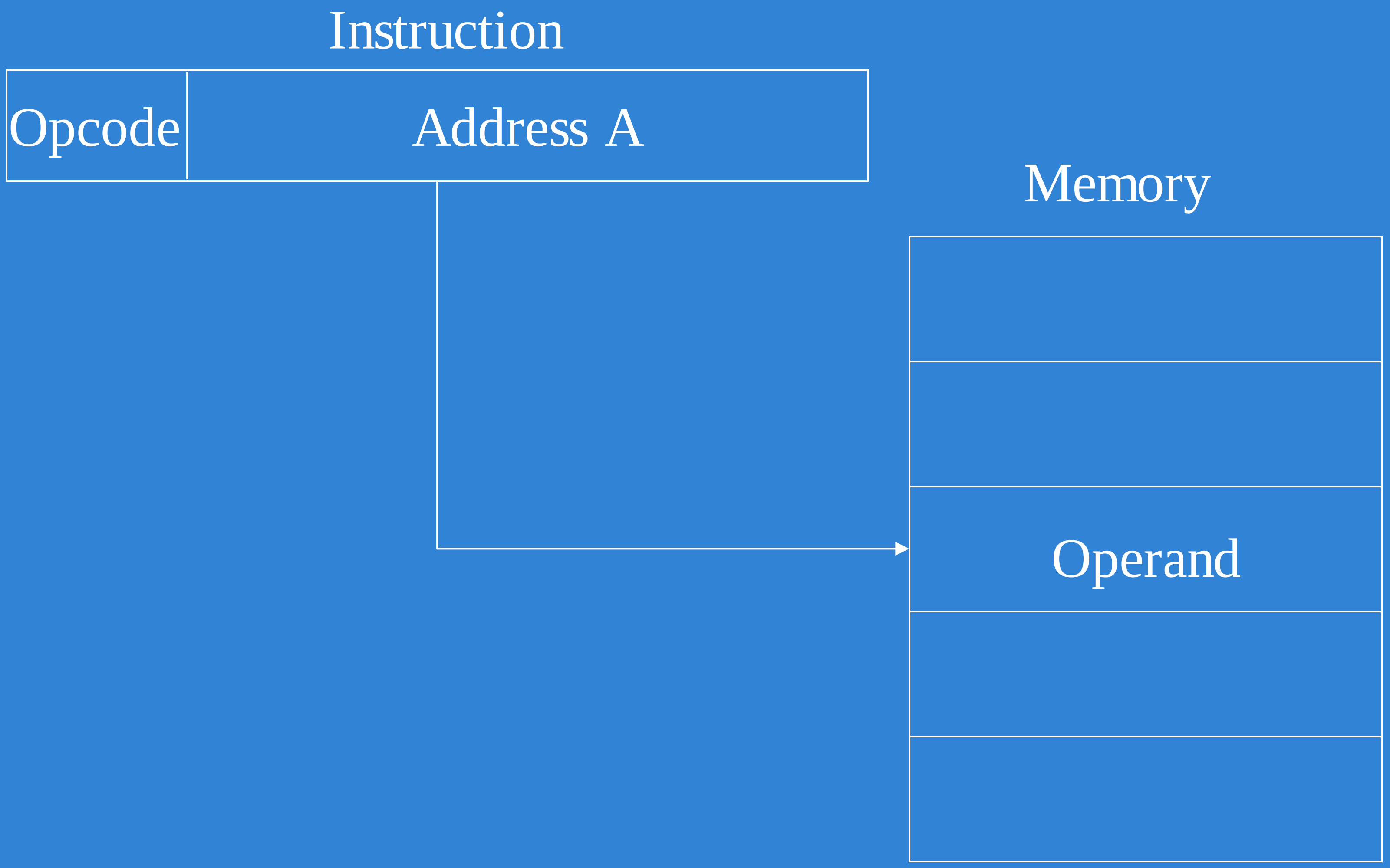
opcode

operand

Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Direct Addressing Diagram



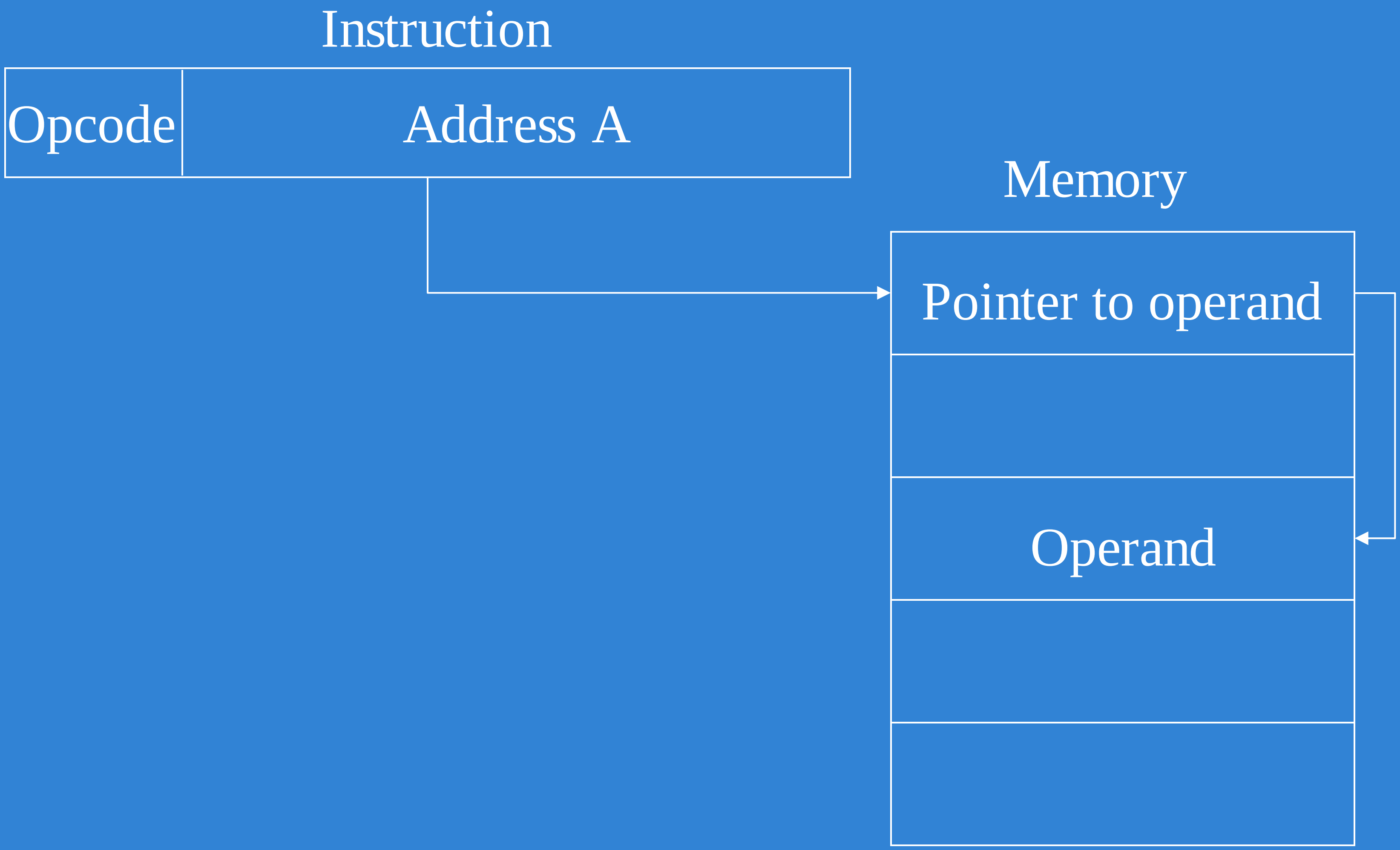
Indirect Addressing (1)

- 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 (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (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

Indirect Addressing Diagram



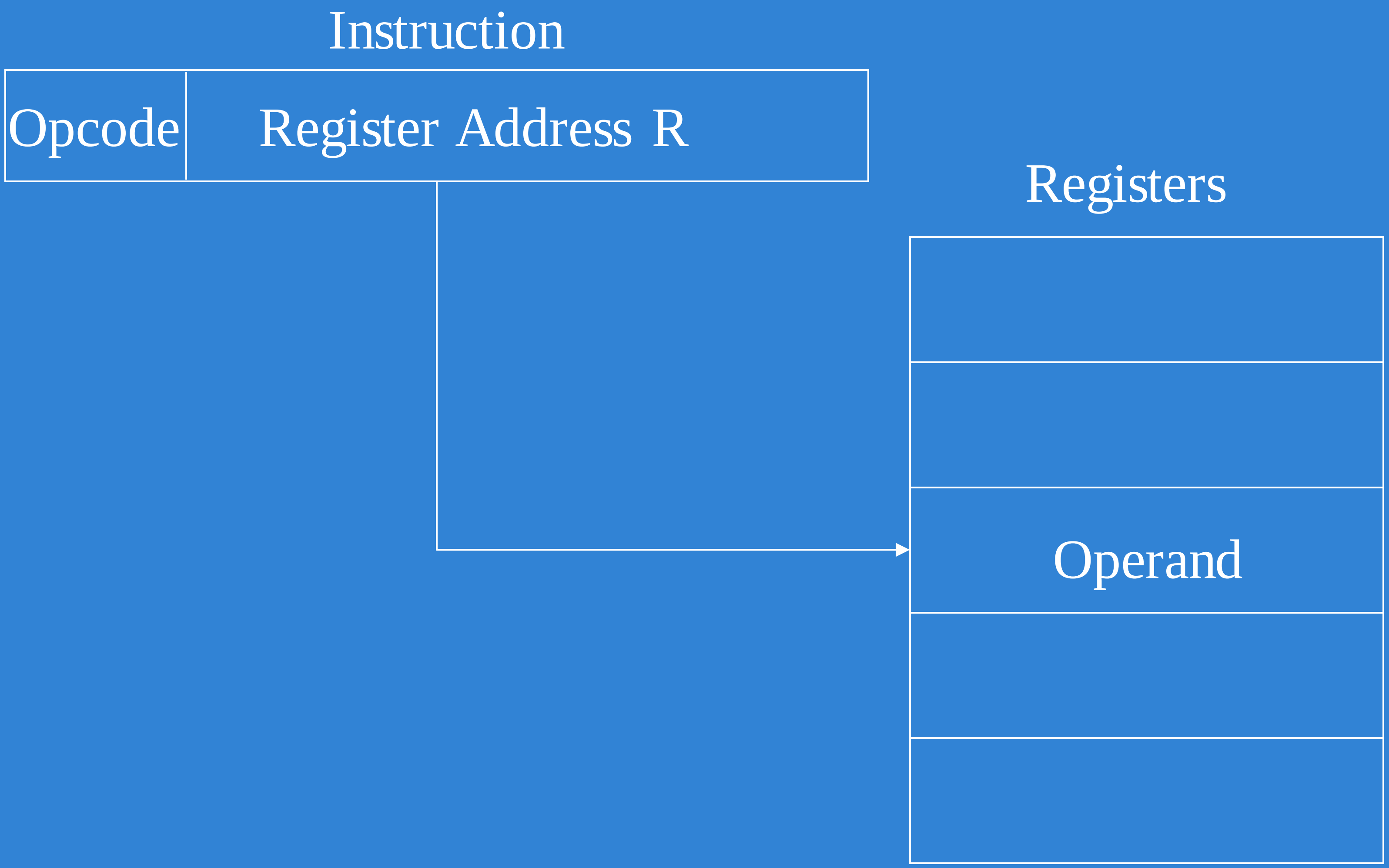
Register Addressing (1)

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

Register Addressing (2)

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

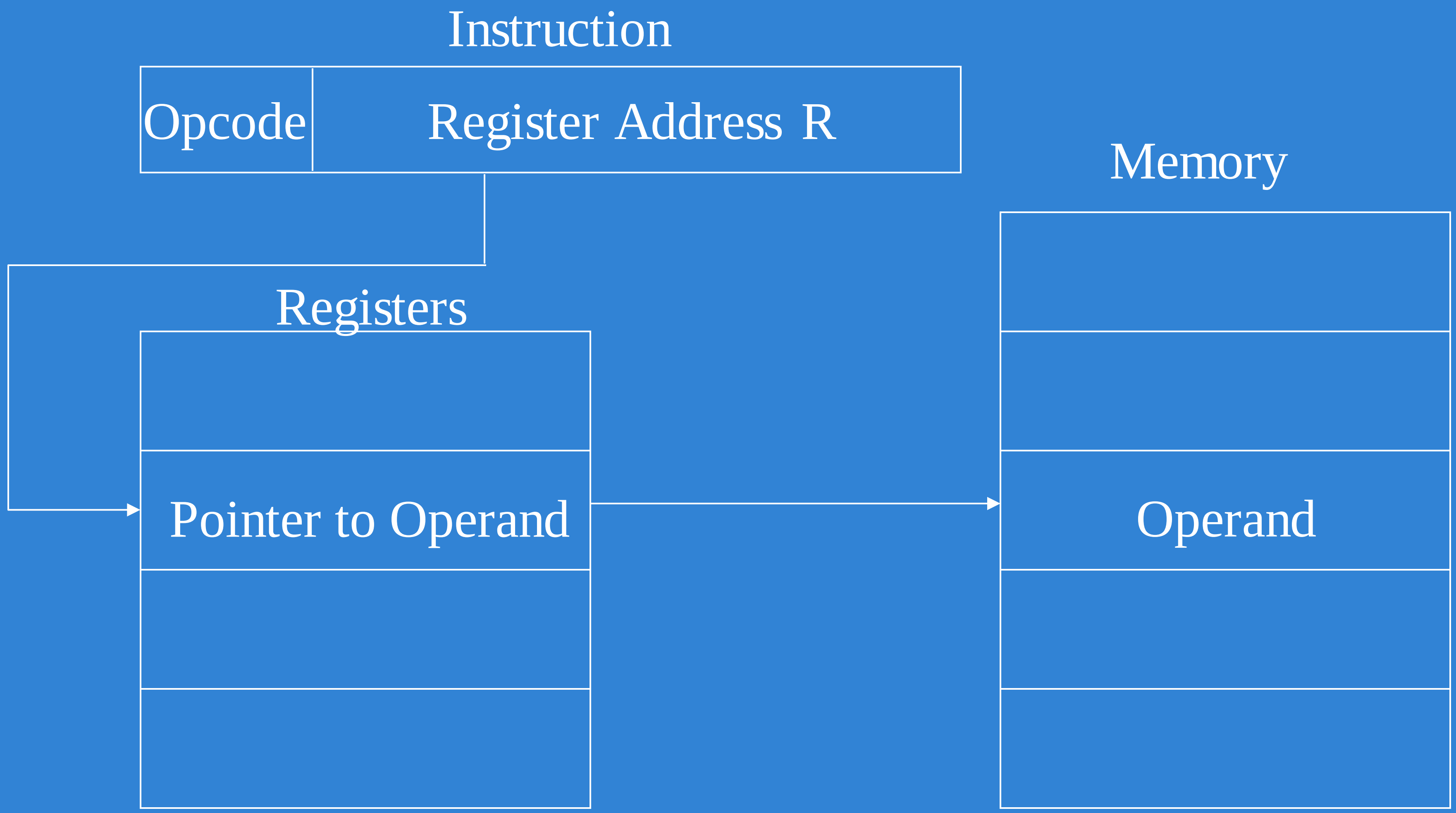
Register Addressing Diagram



Register Indirect Addressing

- C.f. indirect addressing
- EA = [R]
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

Register Indirect Addressing Diagram



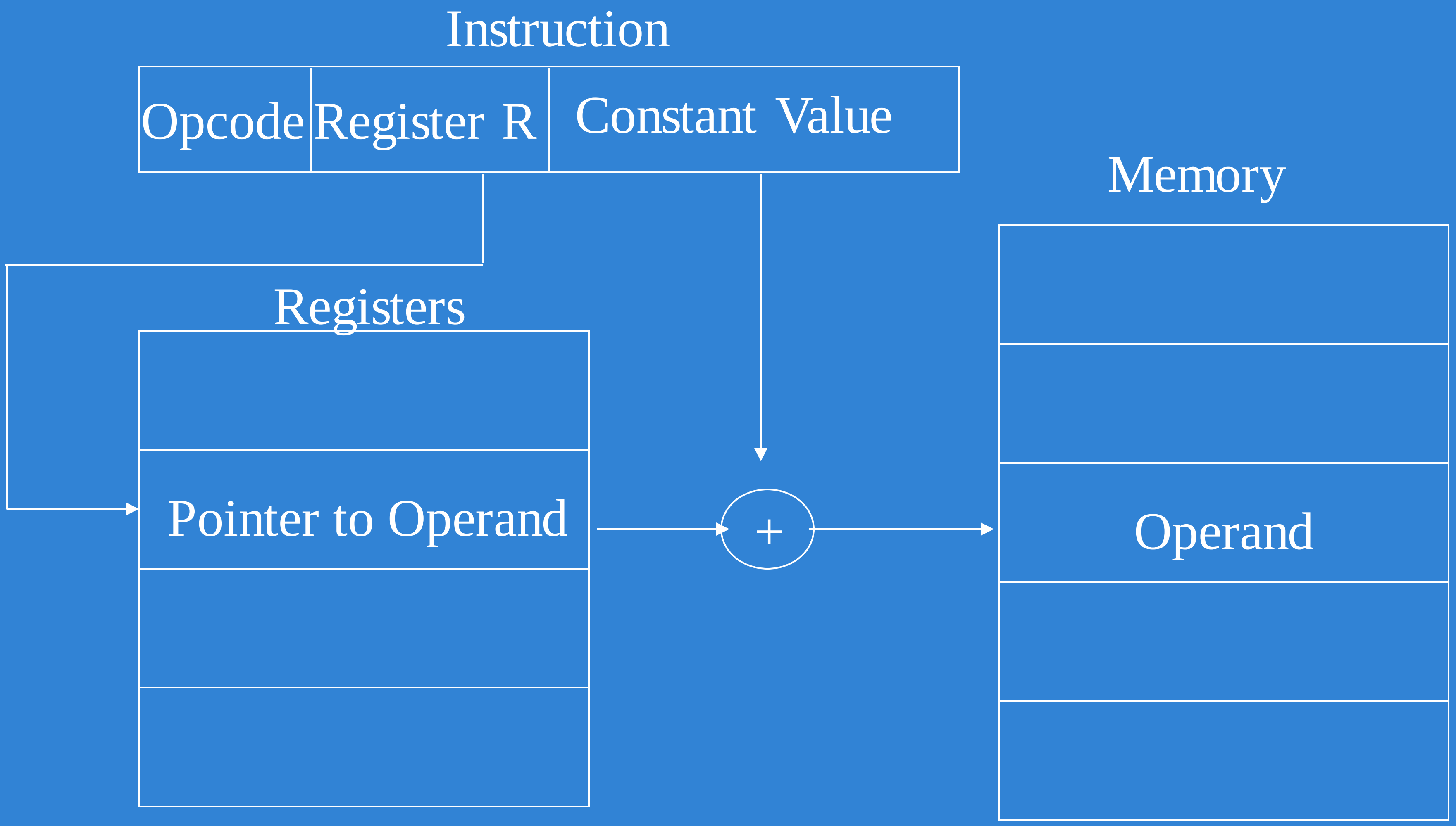
Indexed Addressing

- EA = X + [R]
- Address field hold two values
  - X = constant value (offset)
  - R = register that holds address of memory locations
  - or vice versa

□ (Offset given as constant or in the index register)

Add 20(R1),R2 or Add 1000(R1),R2

Indexed Addressing Diagram



Relative Addressing

- A version of displacement addressing
- R = Program counter, PC
- $EA = X + (PC)$
- i.e. get operand from X bytes away from current location pointed to by PC
- c.f locality of reference & cache usage

Auto increment mode

- The effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in the list
- EA=[Ri]; Increment Ri ---- (Ri)+

Eg: Add (R2)+,R0

Auto decrement mode

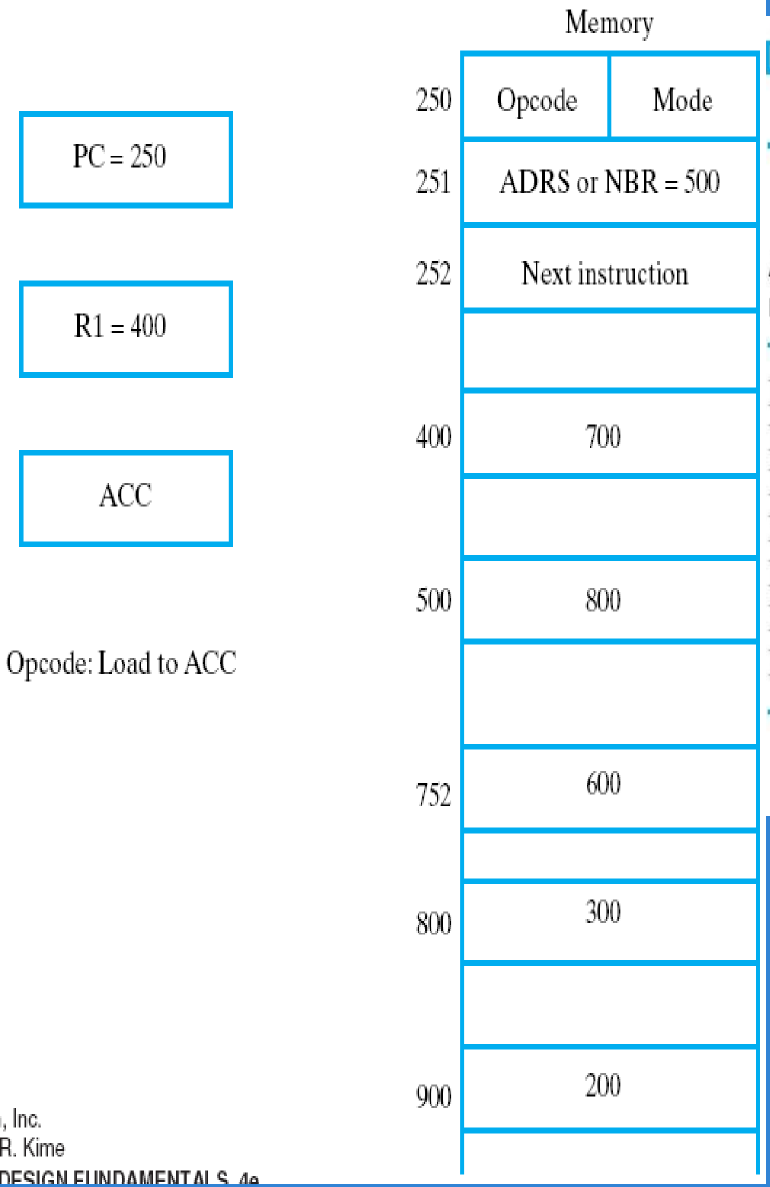
- The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand

- Decrement Ri; EA= [Ri] ----- -(Ri)

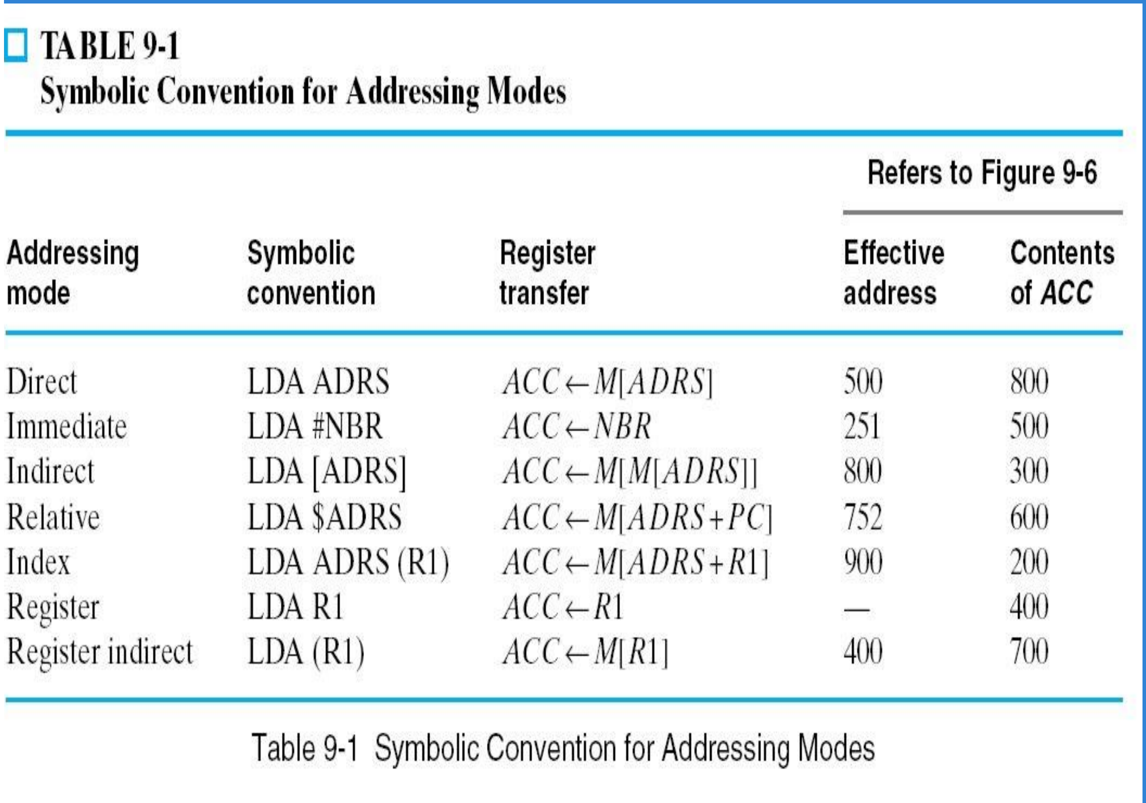
Addressing Modes

- Implied mode
  - The operand is specified implicitly in the definition of the opcode.
- Immediate mode
  - The actual operand is specified in the instruction itself.

Addressing Modes (Summary)



, Inc.
R. Kime
DESIGN FUNDAMENTALS 4e



□ **TABLE 9-1**
Symbolic Convention for Addressing Modes

Refers to Figure 9-6

Addressing mode	Symbolic convention	Register transfer	Effective address	Contents of *ACC*
Direct	LDA ADRS	$ACC \leftarrow M[ADRS]$	500	800
Immediate	LDA #NBR	$ACC \leftarrow NBR$	251	500
Indirect	LDA [ADRS]	$ACC \leftarrow M[M[ADRS]]$	800	300
Relative	LDA $ADRS	$ACC \leftarrow M[ADRS+PC]$	752	600
Index	LDA ADRS (R1)	$ACC \leftarrow M[ADRS+R1]$	900	200
Register	LDA R1	$ACC \leftarrow R1$	—	400
Register indirect	LDA (R1)	$ACC \leftarrow M[R1]$	400	700

Table 9-1 Symbolic Convention for Addressing Modes

Base register LDA #ADRS(R1) ACC <- M[R1+ADRS]

Assembly Language

- Mnemonics
- Assembly Language
- Assembler
- Assembler Directives

Assembly language is mostly a thin layer above the machine structure. An assembly language is a low-level programming language for a computer, microcontroller, or other programmable device, in which each statement corresponds to a single machine code instruction. Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

An assembler directive is a message to the assembler that tells the assembler something it needs to know in order to carry out the assembly process; for example, an assembler directive tells the assembler where a program is to be located in memory.
Examples of common assembler directives are ORG (origin), EQU (equate), and DS.B (define space for a byte).

Equate

The EQU assembler directive simply equates a symbolic name to a numeric value. Consider:

```
Sunday EQU 1
Monday EQU 2
```

The assembler substitutes the equated value for the symbolic name; for example, if you write the instruction ADD.B #Sunday,D2, the assembler treats it as if it were ADD.B #1,D2.

You could also write

```
Sunday EQU 1
Monday EQU Sunday + 1
```

In this case, the assembler evaluates "Sunday + 1" as 1 + 1 and assigns the value 2 to the symbolic name "Monday".

ORG

sets the current origin to a new value. This is used to set the program or register address during assembly. For example, ORG 0100h tells the assembler to assemble all s

Subsequent code starting at address 0100h.

DS

Defines an amount of free space. No code is generated. This is sometimes used for allocating variable space.

Basic Input/Output Operations

I/O

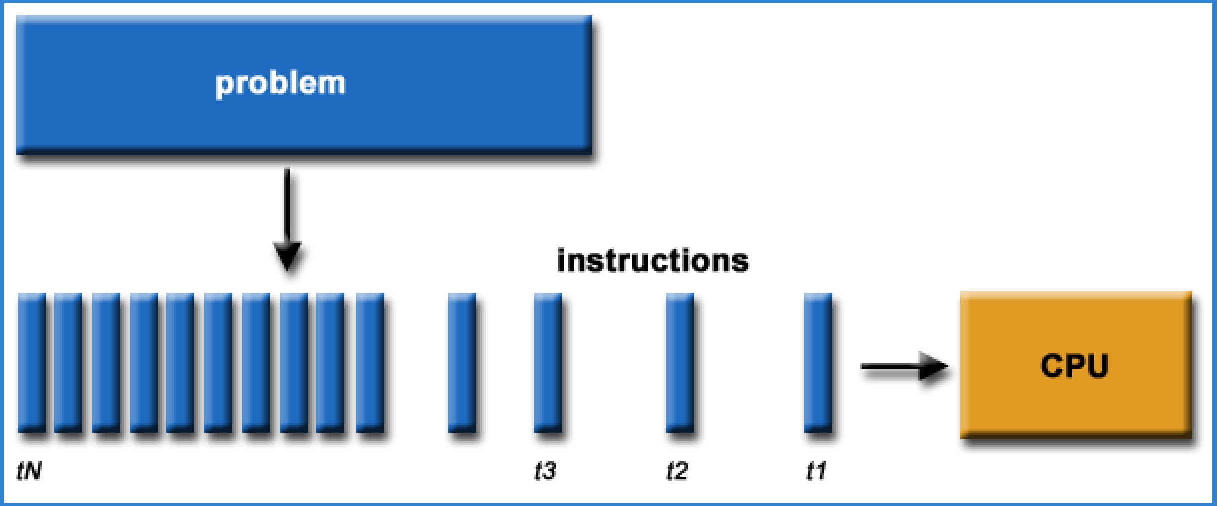
- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

Evolution of parallel computers

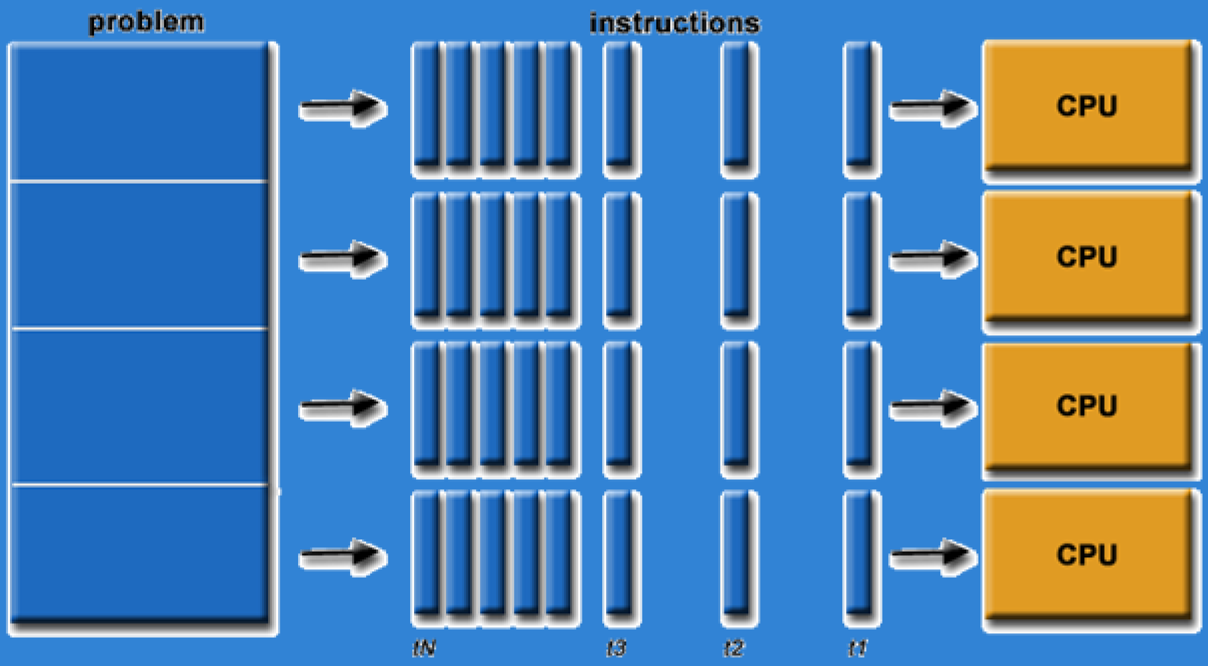
What is Parallel Computing?

– **Parallel Processing Systems** are designed to speed up the execution of programs by dividing the program into multiple fragments and processing these fragments simultaneously



What is Parallel Computing?

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem.
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs

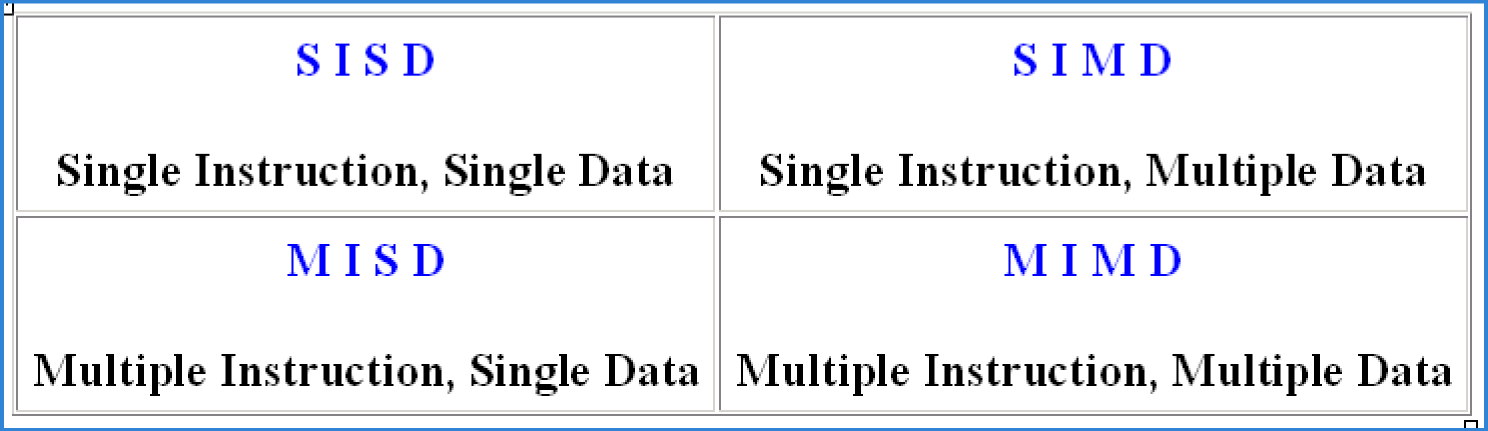


Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.

Flynn Matrix

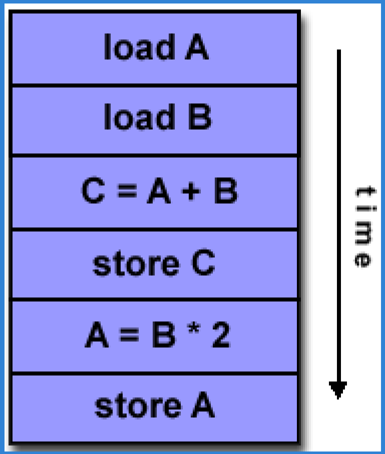
- The matrix below defines the 4 possible classifications according to Flynn



SISD Single Instruction, Single Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

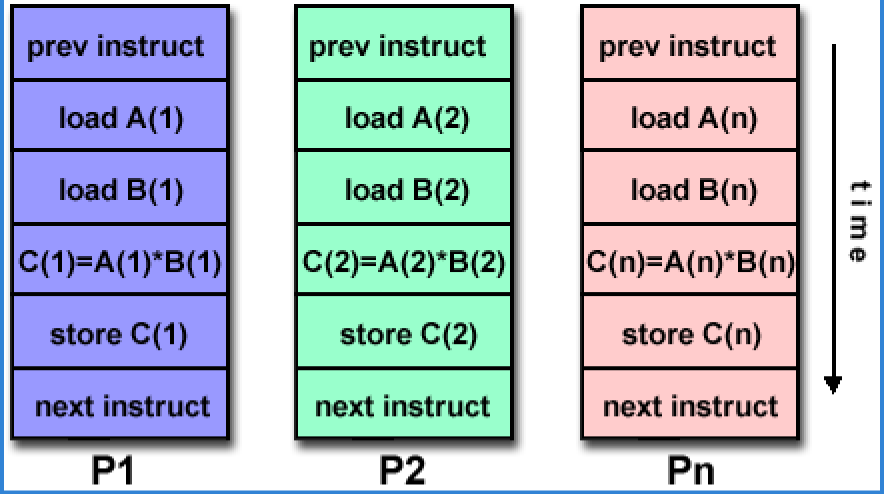
Single Instruction, Single Data (SISD)

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



Single Instruction, Multiple Data (SIMD)

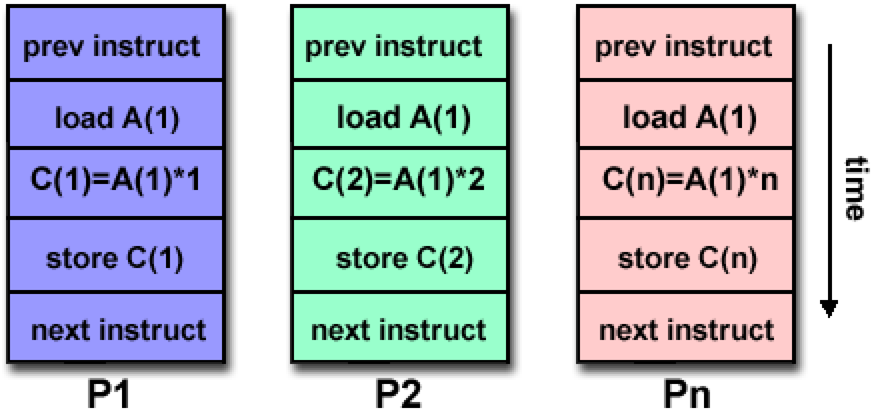
- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity,such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820



P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	load A(2)	load A(n)
load B(1)	load B(2)	load B(n)
C(1)=A(1)*B(1)	C(2)=A(2)*B(2)	C(n)=A(n)*B(n)
store C(1)	store C(2)	store C(n)
next instruct	next instruct	next instruct

Multiple Instruction, Single Data (MISD)

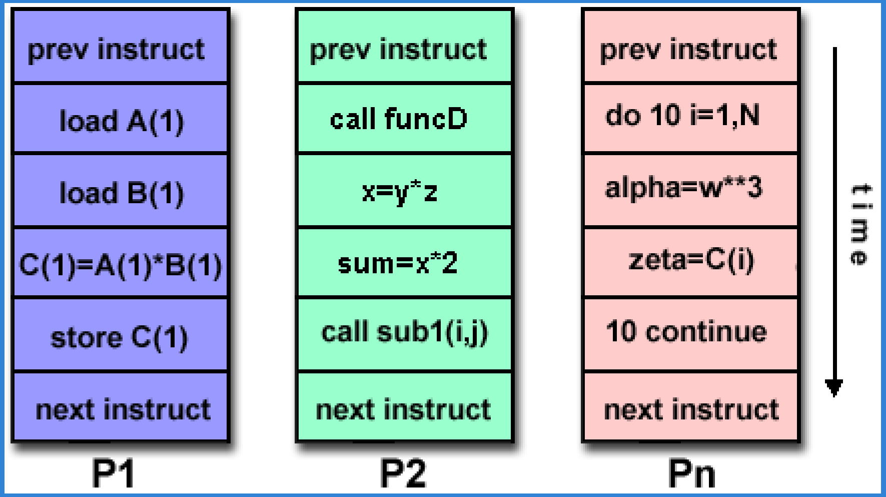
- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
- multiple cryptography algorithms attempting to crack a single coded message.



P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	load A(1)	load A(1)
C(1)=A(1)*1	C(2)=A(1)*2	C(n)=A(1)*n
store C(1)	store C(2)	store C(n)
next instruct	next instruct	next instruct

Multiple Instruction, Multiple Data (MIMD)

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	call funcD	do 10 i=1,N
load B(1)	x=y*z	alpha=w**3
C(1)=A(1)*B(1)	sum=x*2	zeta=C(i)
store C(1)	call sub1(i,j)	10 continue
next instruct	next instruct	next instruct

Performance

Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:

➢ Hardware design

➢ Instruction set

➢ Compiler

Performance

- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.

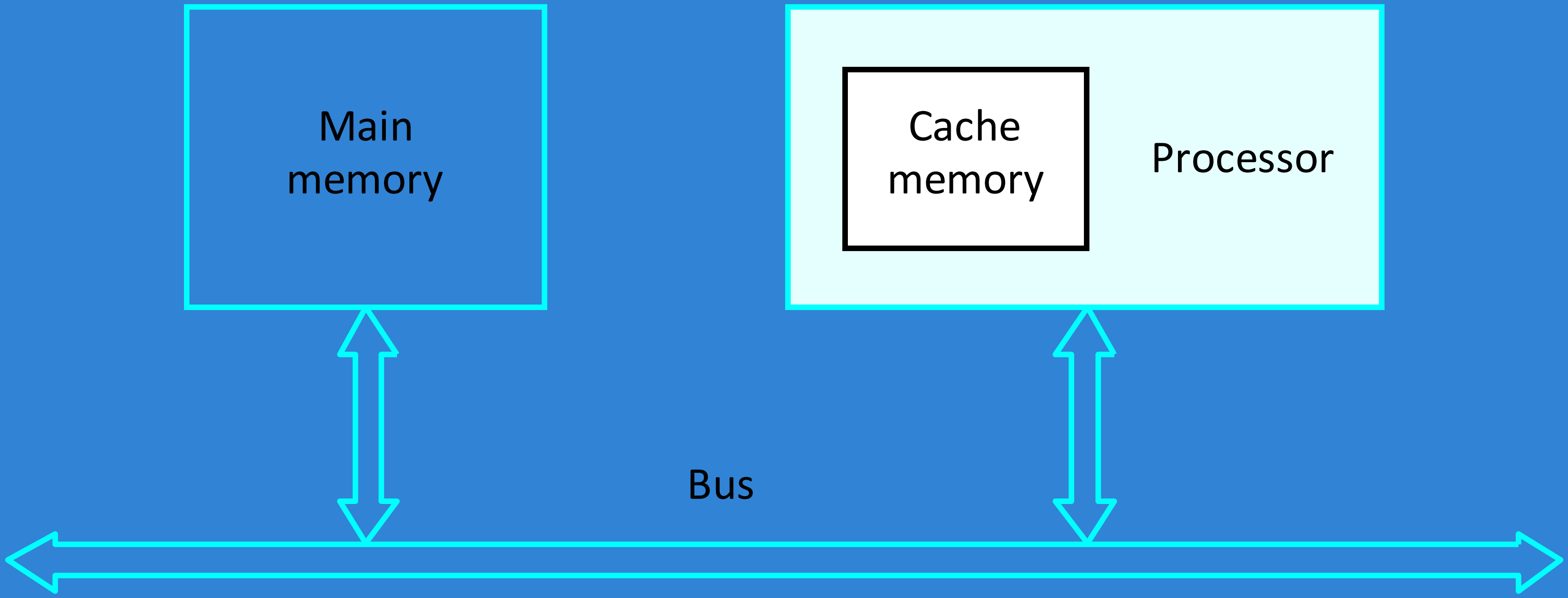


Figure 1.5. The processor cache.

Performance

- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- Speed
- Cost
- Memory management

Processor Clock

- Processor Ckts are controlled by a timing signal called Clock.
- clock cycle, and clock rate
- The execution of each instruction is divided into several steps, each of which completes in one clock cycle.
- Hertz – cycles per second
- 500 million cycles per second-500MHz
- 1250- million cycles per second-1.25GHz

Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

$$T = \frac{N \times S}{R}$$

How to improve T?

Pipeline and Superscalar Operation

- Instructions are not necessarily executed one after another.
- The value of S doesn’t have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- **Pipelining** is an implementation technique where multiple instructions are overlapped in execution. The **computer pipeline** is divided in stages. Each stage completes a part of an instruction in parallel.
- Add R1, R2, R3
- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce S (could become <1!)

Clock Rate

- Increase clock rate
  - Improve the integrated-circuit (IC) technology to make the circuits faster
  - Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor’s operation equally except the time to access the main memory.

CISC and RISC

- Tradeoff between N and S
- A key consideration is the use of pipelining
  - ➢ S is close to 1 even though the number of basic steps per instruction may be considerably larger
  - ➢ It is much easier to implement efficient pipelining in processor with simple instruction sets
- Reduced Instruction Set Computers (RISC)
- Complex Instruction Set Computers (CISC)

Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce N×S
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\,rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\,rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Multiprocessors and Multicomputers

- Multiprocessor computer
- ➢ Execute a number of different application tasks in parallel
- ➢ Execute subtasks of a single large task in parallel
- ➢ All processors have access to all of the memory – shared-memory multiprocessor
- ➢ Cost – processors, memory units, complex interconnection networks
- Multicomputers
- ➢ Each computer only have access to its own memory
- ➢ Exchange message via a communication network – message-passing multicomputers

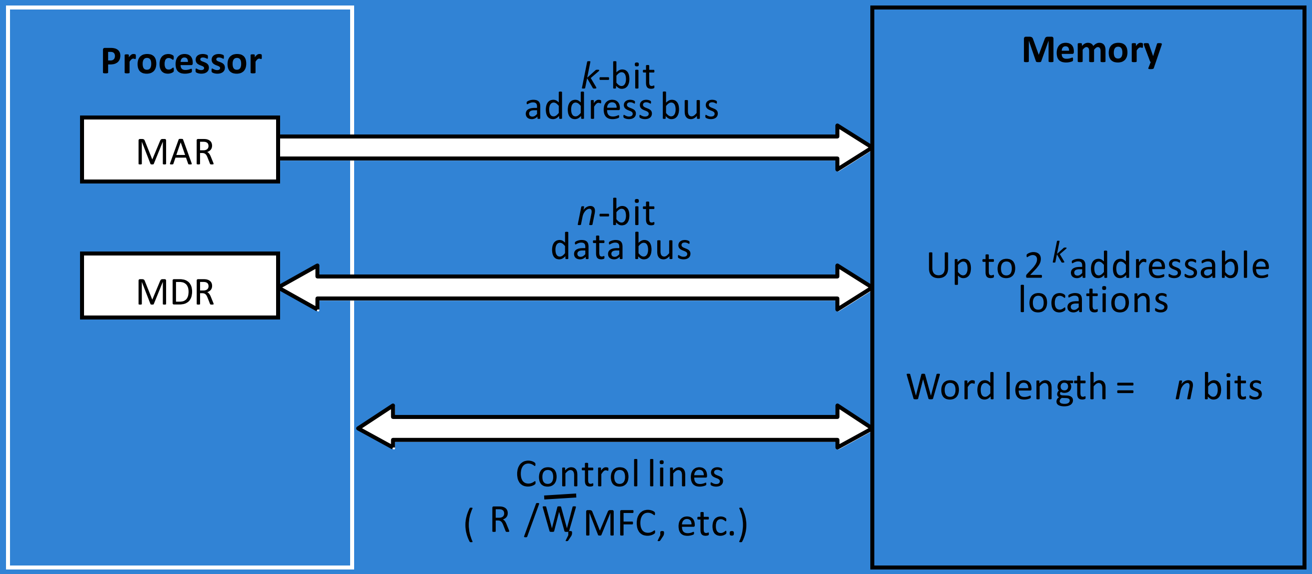
UNIT IV

The Memory System

The Memory System

Some basic concepts

- Maximum size of the Main Memory
- byte-addressable
- CPU-Main Memory Connection



Processor

MAR

MDR

k-bit
address bus

n-bit
data bus

Control lines
($R/\overline{W}$, MFC, etc.)

Memory

Up to 2^k addressable
locations

Word length = *n* bits

Some basic concepts(Contd..)

- Measures for the speed of a memory:
  - memory access time.
  - memory cycle time.
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

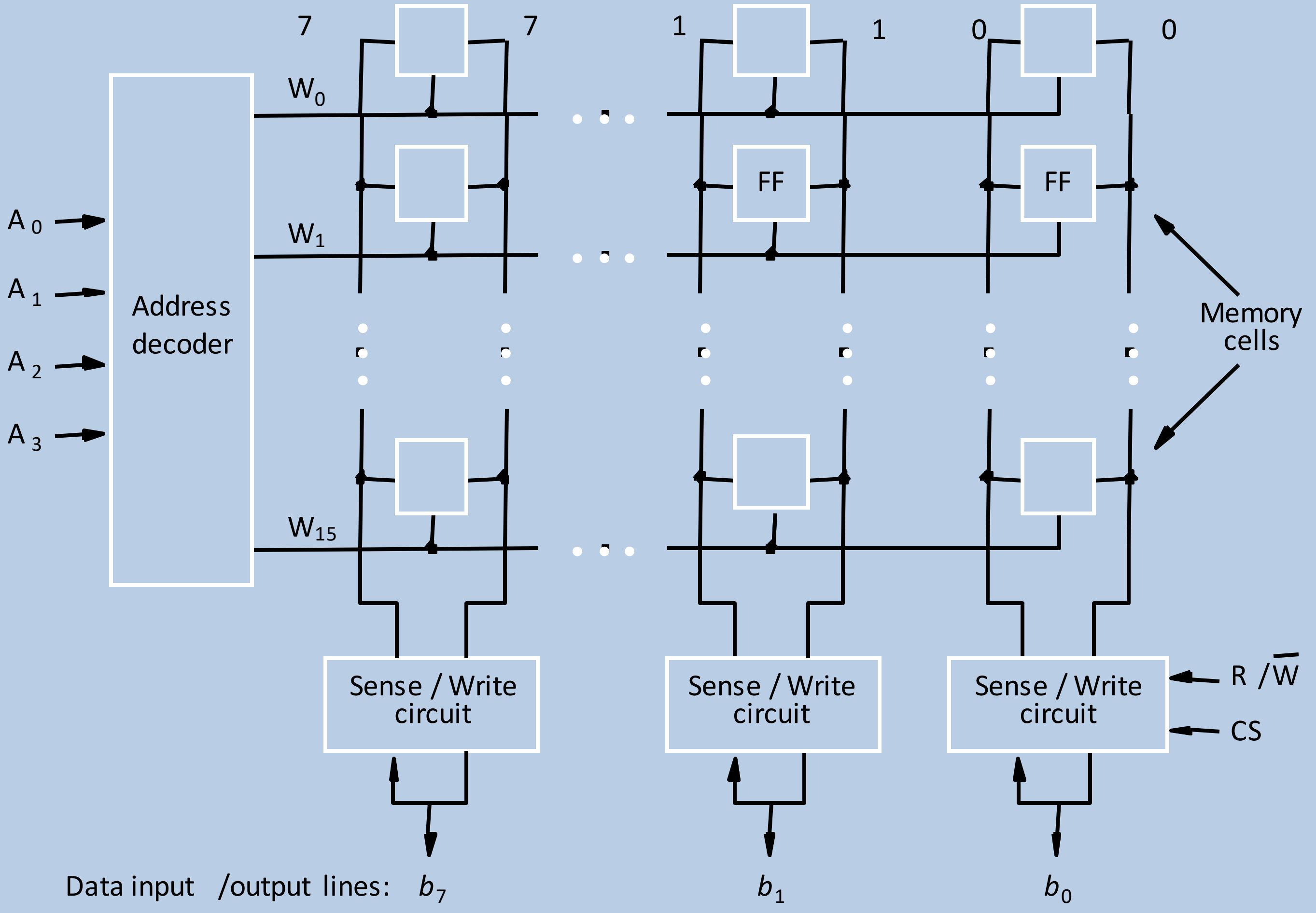
The Memory System

Semiconductor RAM memories

Internal organization of memory chips

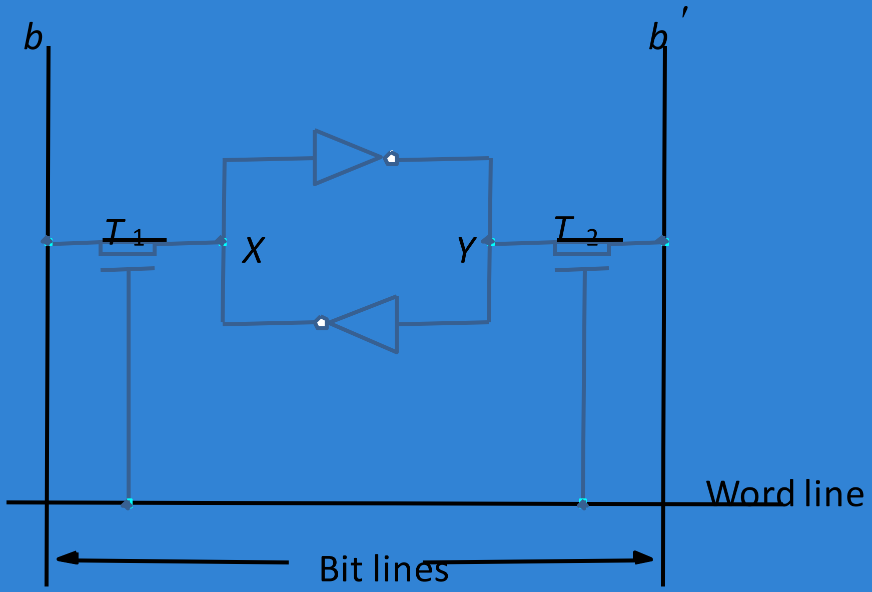
- Each memory cell can hold one bit of information.
- Memory cells are organized in the form of an array.
- One row is one memory word.
- All cells of a row are connected to a common line, known as the “word line”.
- Word line is connected to the address decoder.
- Sense/write circuits are connected to the data input/output lines of the memory chip.

Internal organization of memory chips (Contd.,)



SRAM Cell

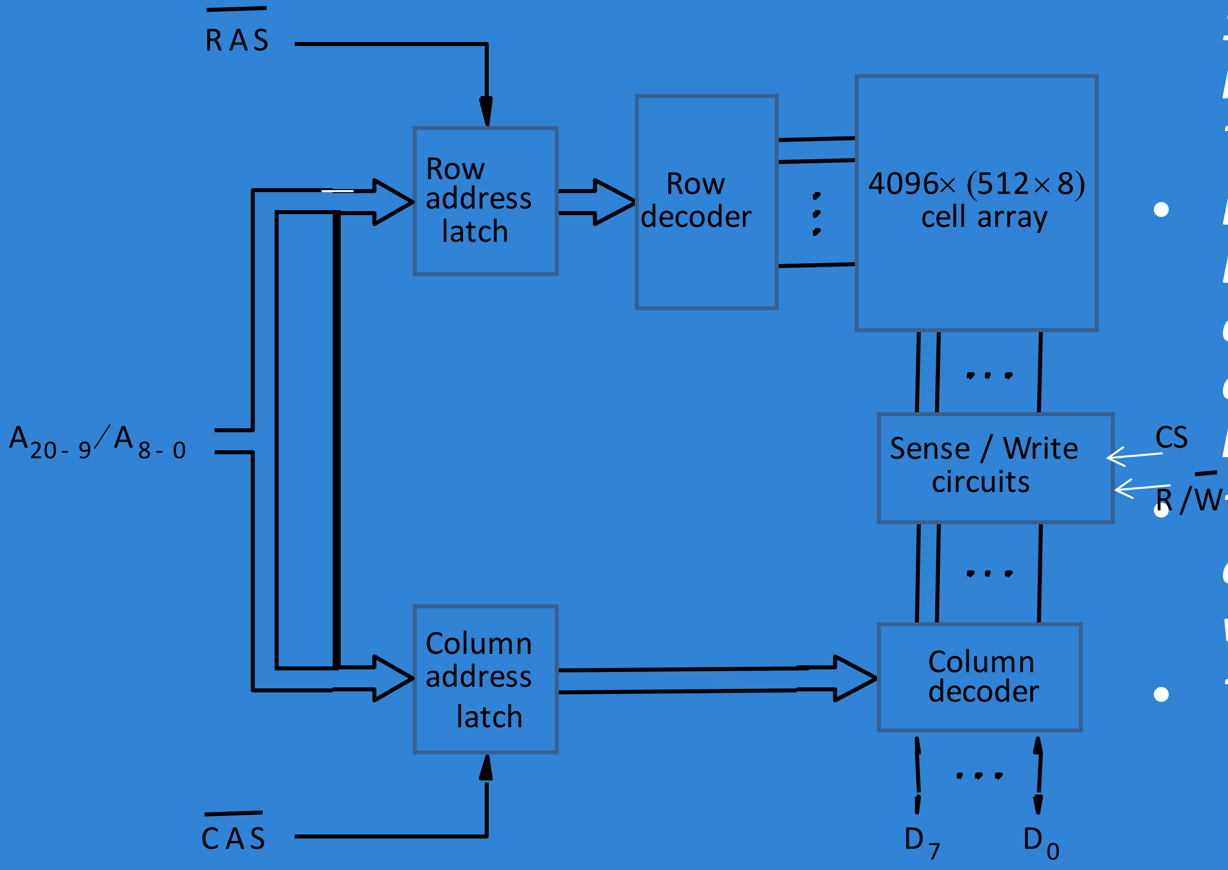
- Two transistor inverters are cross connected to implement a basic flip-flop.
- The cell is connected to one word line and two bits lines by transistors T1 and T2
- When word line is at ground level, the transistors are turned off and the latch retains its state
- Read operation: In order to read state of SRAM cell, the word line is activated to close switches T1 and T2. Sense/Write circuits at the bottom monitor the state of b and b’



Asynchronous DRAMs

- Static RAMs (SRAMs):
  - Consist of circuits that are capable of retaining their state as long as the power is applied.
  - Volatile memories, because their contents are lost when power is interrupted.
  - Access times of static RAMs are in the range of few nanoseconds.
  - However, the cost is usually high.
- Dynamic RAMs (DRAMs):
  - Do not retain their state indefinitely.
  - Contents must be periodically refreshed.
  - Contents may be refreshed while accessing them for reading.

Asynchronous DRAMs

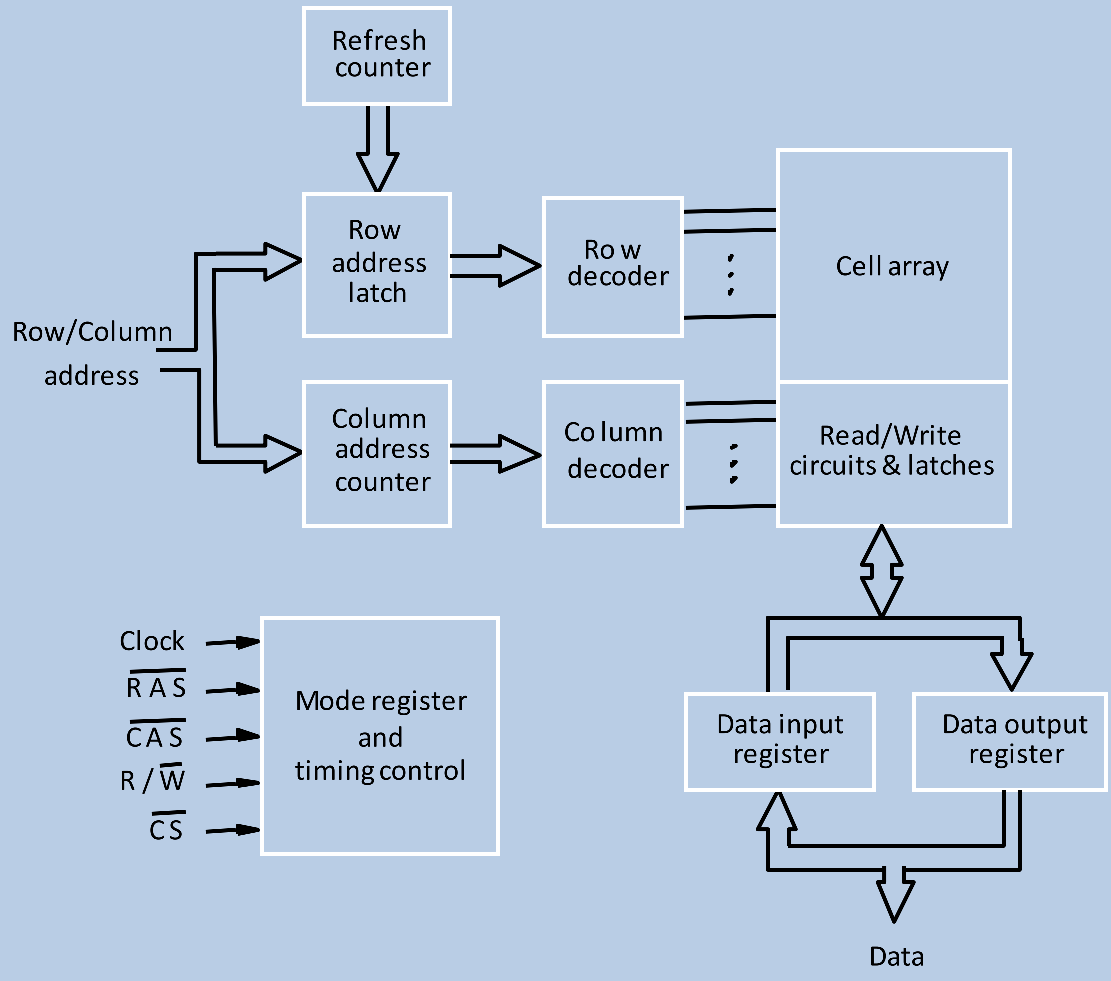


- *Each row can store 512 bytes. 12 bits to select a row, and 9 bits to select a group in a row. Total of 21 bits.*
- *First apply the row address, RAS signal latches the row address. Then apply the column address, CAS signal latches the address.*
- *Timing of the memory unit is controlled by a specialized unit which generates RAS and CAS.*
- *This is asynchronous DRAM*

Fast Page Mode

- Suppose if we want to access the consecutive bytes in the selected row.
- This can be done without having to reselect the row.
  - Add a latch at the output of the sense circuits in each row.
  - All the latches are loaded when the row is selected.
  - Different column addresses can be applied to select and place different bytes on the data lines.
- Consecutive sequence of column addresses can be applied under the control signal CAS, without reselecting the row.
  - Allows a block of data to be transferred at a much faster rate than random accesses.
  - A small collection/group of bytes is usually referred to as a block.
- This transfer capability is referred to as the fast page mode feature.

Synchronous DRAMs

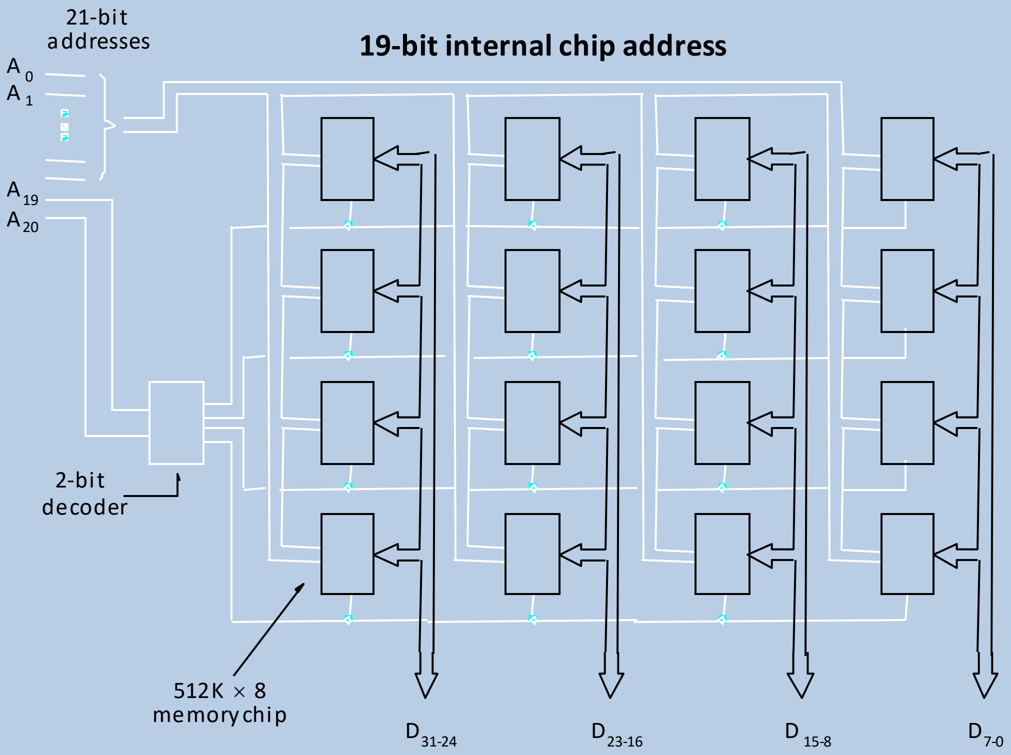


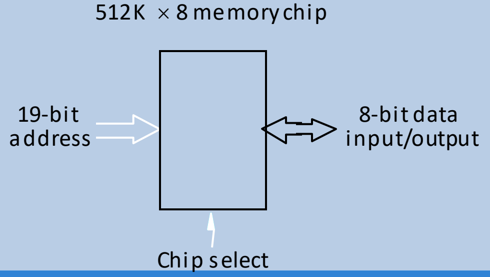
- *Operation is directly synchronized with processor clock signal.*
- *The outputs of the sense circuits are connected to a latch.*
- *During a Read operation, the contents of the cells in a row are loaded onto the latches.*
- *During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches.*
- *Data held in the latches correspond to the selected columns are transferred to the output.*
- *For a burst mode of operation, successive columns are selected using column address counter and clock. $\overline{CAS}$ signal need not be generated externally. A new data is placed during raising edge of the clock*

Latency, Bandwidth, and DDRSDRAMs

- Memory latency is the time it takes to transfer a word of data to or from memory
- Memory bandwidth is the number of bits or bytes that can be transferred in one second.
- DDRSDRAMs
  - Cell array is organized in two banks

Static memories





Implement a memory unit of 2M words of 32 bits each.
Use 512x8 static memory chips.
Each column consists of 4 chips.
Each chip implements one byte position.
A chip is selected by setting its chip select control line to 1.
Selected chip places its data on the data output line, outputs of other chips are in high impedance state.
21 bits to address a 32-bit word.
High order 2 bits are needed to select the row, by activating the four Chip Select signals.
19 bits are used to access specific byte locations inside the selected chip.

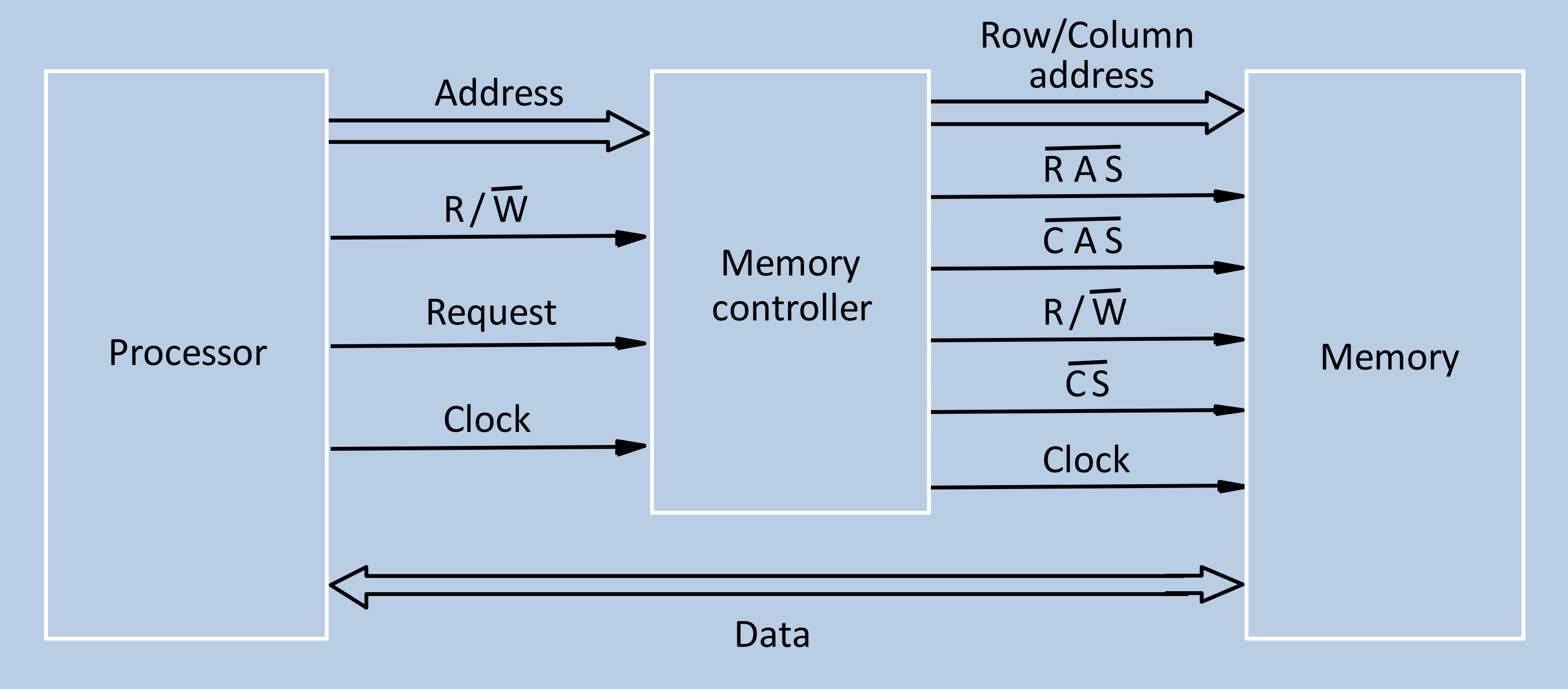
Dynamic memories

- Large dynamic memory systems can be implemented using DRAM chips in a similar way to static memory systems.
- Placing large memory systems directly on the motherboard will occupy a large amount of space.
  - Also, this arrangement is inflexible since the memory system cannot be expanded easily.
- Packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules).
- Memory modules are an assembly of memory chips on a small board that plugs vertically onto a single socket on the motherboard.
  - Occupy less space on the motherboard.
  - Allows for easy expansion by replacement.

Memory controller

- Recall that in a dynamic memory chip, to reduce the number of pins, multiplexed addresses are used.
- Address is divided into two parts:
  - High-order address bits select a row in the array.
  - They are provided first, and latched using RAS signal.
  - Low-order address bits select a column in the row.
  - They are provided later, and latched using CAS signal.
- However, a processor issues all address bits at the same time.
- In order to achieve the multiplexing, memory controller circuit is inserted between the processor and memory.

Memory controller (contd..)



The Memory System

Read-Only Memories (ROMs)

Read-Only Memories (ROMs)

- SRAM and SDRAM chips are volatile:
  - Lose the contents when the power is turned off.
- Many applications need memory devices to retain contents after the power is turned off.
  - For example, computer is turned on, the operating system must be loaded from the disk into the memory.
  - Store instructions which would load the OS from the disk.
  - Need to store these instructions so that they will not be lost after the power is turned off.
  - We need to store the instructions into a non-volatile memory.
- Non-volatile memory is read in the same manner as volatile memory.
  - Separate writing process is needed to place information in this memory.
  - Normal operation involves only reading of data, this type of memory is called Read-Only memory (ROM).

Read-Only Memories (Contd..)

- Read-Only Memory:
  - Data are written into a ROM when it is manufactured.
- Programmable Read-Only Memory (PROM):
  - Allow the data to be loaded by a user.
  - Process of inserting the data is irreversible.
  - Storing information specific to a user in a ROM is expensive.
  - Providing programming capability to a user may be better.
- Erasable Programmable Read-Only Memory (EPROM):
  - Stored data to be erased and new data to be loaded.
  - Flexibility, useful during the development phase of digital systems.
  - Erasable, reprogrammable ROM.
  - Erasure requires exposing the ROM to UV light.

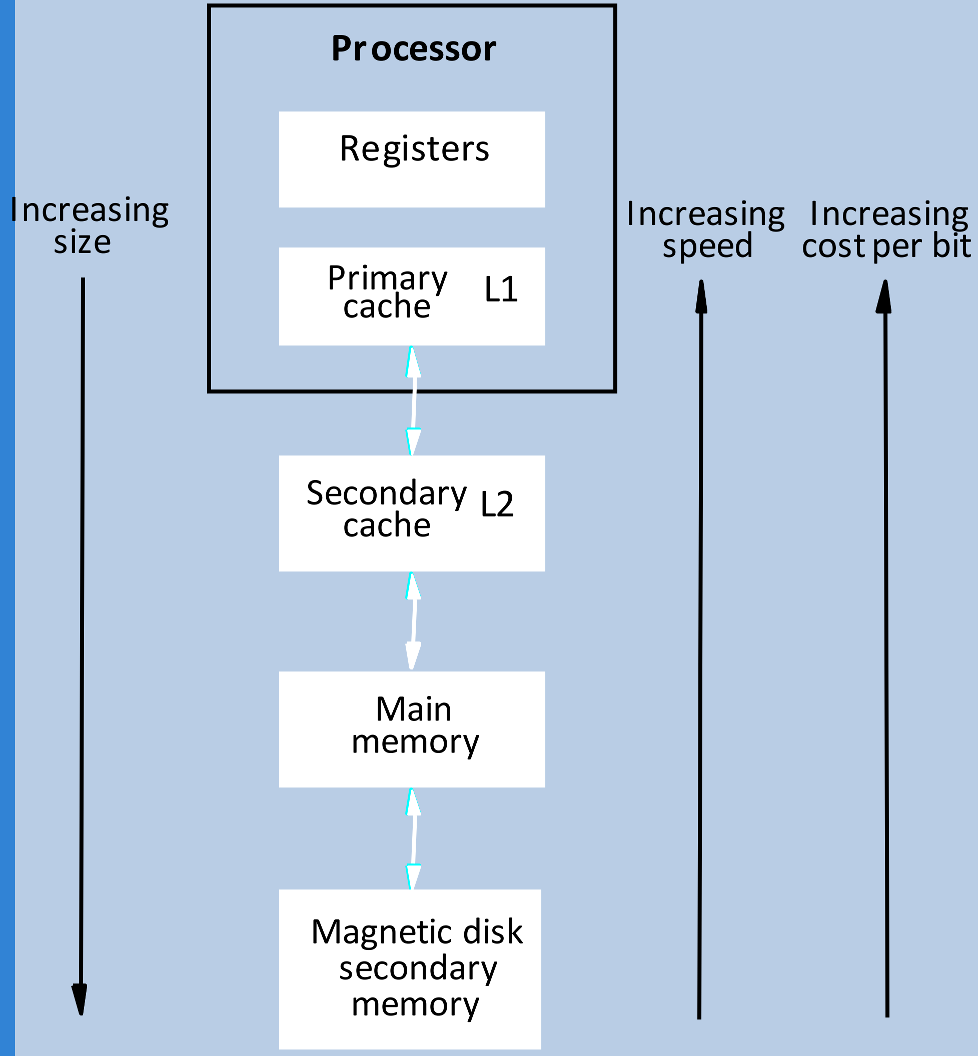
Read-Only Memories (Contd..)

- Electrically Erasable Programmable Read-Only Memory (EEPROM):
  - To erase the contents of EPROMs, they have to be exposed to ultraviolet light.
  - Physically removed from the circuit.
  - EEPROMs the contents can be stored and erased electrically.
- Flash memory:
  - Has similar approach to EEPROM.
  - Read the contents of a single cell, but write the contents of an entire block of cells.
  - Flash devices have greater density.
    - Higher capacity and low storage cost per bit.
  - Power consumption of flash memory is very low, making it attractive for use in equipment that is battery-driven.
  - Single flash chips are not sufficiently large, so larger memory modules are implemented using flash cards and flash drives.

Speed, Size, and Cost

- A big challenge in the design of a computer system is to provide a sufficiently large memory, with a reasonable speed at an affordable cost.
- Static RAM:
  - Very fast, but expensive, because a basic SRAM cell has a complex circuit making it impossible to pack a large number of cells onto a single chip.
- Dynamic RAM:
  - Simpler basic cell circuit, hence are much less expensive, but significantly slower than SRAMs.
- Magnetic disks:
  - Storage provided by DRAMs is higher than SRAMs, but is still less than what is necessary.
  - Secondary storage such as magnetic disks provide a large amount of storage, but is much slower than DRAMs.

Memory Hierarchy



- *Fastest access is to the data held in processor registers. Registers are at the top of the memory hierarchy.*
- *Relatively small amount of memory that can be implemented on the processor chip. This is processor cache.*
- *Two levels of cache. Level 1 (L1) cache is on the processor chip. Level 2 (L2) cache is in between main memory and processor.*
- *Next level is main memory, implemented as SIMMs. Much larger, but much slower than cache memory.*
- *Next level is magnetic disks. Huge amount of inexepensive storage.*
- *Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.*

The Memory System

Cache Memories

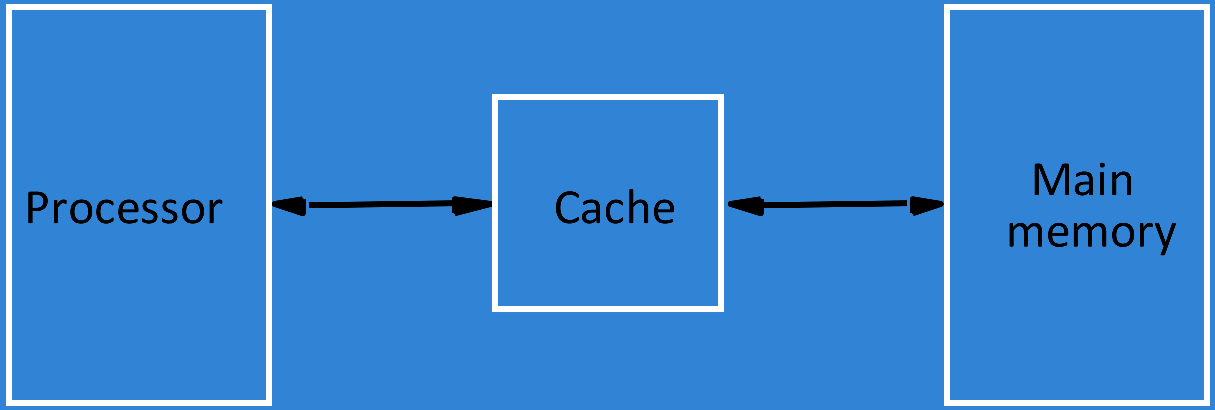
Cache Memories

- Processor is much faster than the main memory.
  - As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
  - Major obstacle towards achieving good performance.
- Speed of the main memory cannot be increased beyond a certain point.
- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- The effectiveness of Cache mechanism is based on the property of computer programs known as "locality of reference". Cache block –cache line

locality of Reference

- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
  - These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.
  - This is called "locality of reference".
- Temporal locality of reference:
  - Recently executed instruction is likely to be executed again very soon.
- Spatial locality of reference:
  - Instructions with addresses close to a recently instruction are likely to be executed soon.

Cache memories



- *Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.*
- *Subsequent references to the data in this block of words are found in the cache.*
- *At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a “mapping function”.*
- *When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a “replacement algorithm”.*

Cache hit

- *Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.*

- *If the data is in the cache it is called a* <u>*Read or Write hit*</u>*.*

- *Read hit:*
  - *The data is obtained from the cache.*

- *Write hit:*
  - *Cache has a replica of the contents of the main memory.*
  - *Contents of the cache and the main memory may be updated simultaneously. This is the* <u>*write-through*</u> *protocol.*
  - *Update the contents of the cache, and mark it as updated by setting a bit known as the* <u>*dirty bit or modified*</u> *bit. The contents of the main memory are updated when this block is replaced. This is* <u>*write-back or copy-back*</u> *protocol.*

Cache miss

- *If the data is not present in the cache, then a Read miss or Write miss occurs.*

- *Read miss:*
  - *Block of words containing this requested word is transferred from the memory.*
  - *After the block is transferred, the desired word is forwarded to the processor.*
  - *The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.*

- *Write-miss:*
  - *Write-through protocol is used, then the contents of the main memory are updated directly.*
  - *If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.*

Cache Coherence Problem

- *A bit called as “valid bit” is provided for each block.*
- *If the block contains valid data, then the bit is set to 1, else it is 0.*
- *Valid bits are set to 0, when the power is just turned on.*
- *When a block is loaded into the cache for the first time, the valid bit is set to 1.*

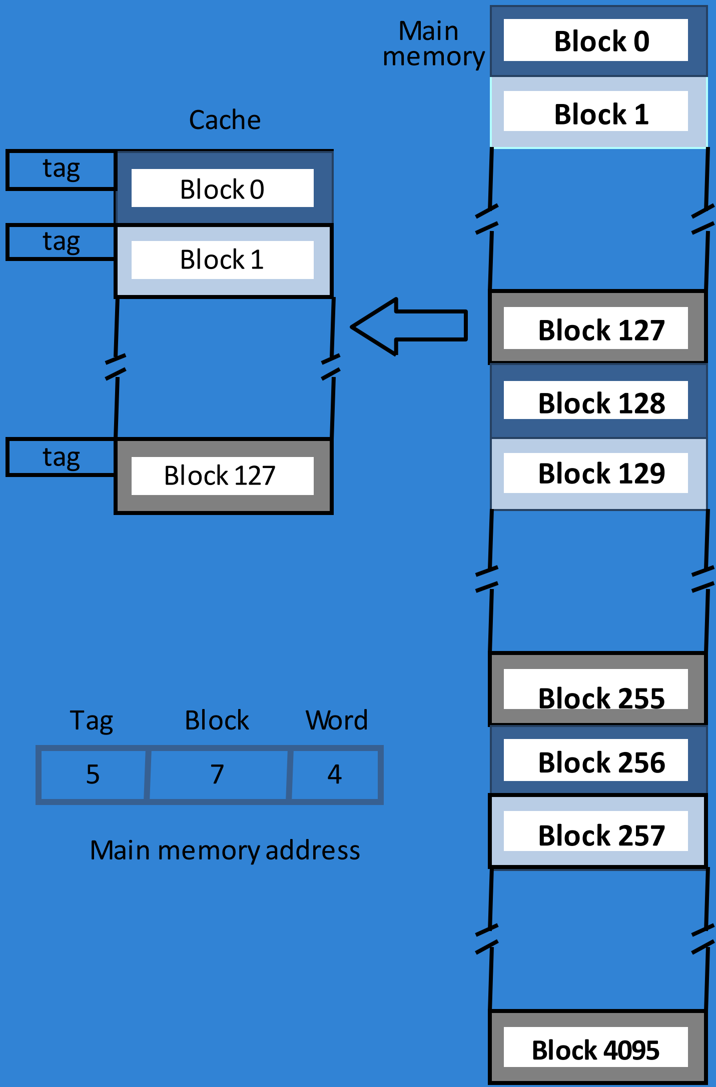
- *Data transfers between main memory and disk occur directly bypassing the cache.*
- *When the data on a disk changes, the main memory block is also updated.*
- *However, if the data is also resident in the cache, then the valid bit is set to 0.*

- *What happens if the data in the disk and main memory changes and the write-back protocol is being used?*
- *In this case, the data in the cache may also have changed and is indicated by the dirty bit.*
- *The copies of the data in the cache, and the main memory are different. This is called the cache coherence problem.*
- *One option is to force a write-back before the main memory is updated from the disk.*

Mapping functions

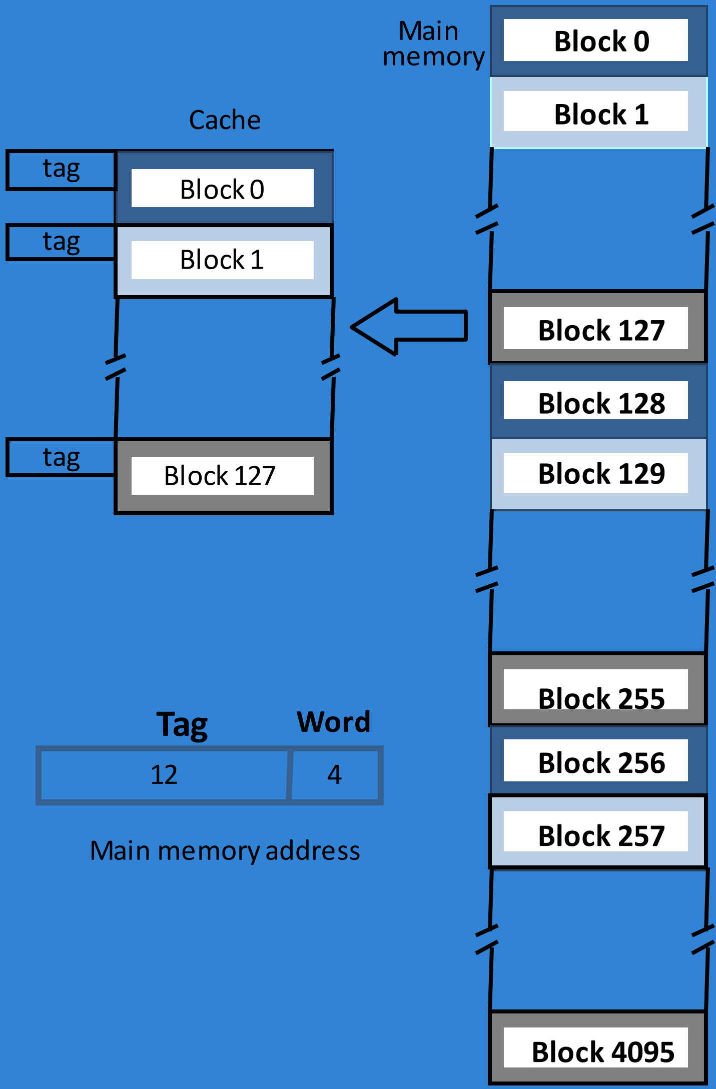
- Mapping functions determine how memory blocks are placed in the cache.
- A simple processor example:
  - Cache consisting of 128 blocks of 16 words each.
  - Total size of cache is 2048 (2K) words.
  - Main memory is addressable by a 16-bit address.
  - Main memory has 64K words.
  - Main memory has 4K blocks of 16 words each.
- Three mapping functions:
  - Direct mapping
  - Associative mapping
  - Set-associative mapping.

Direct mapping



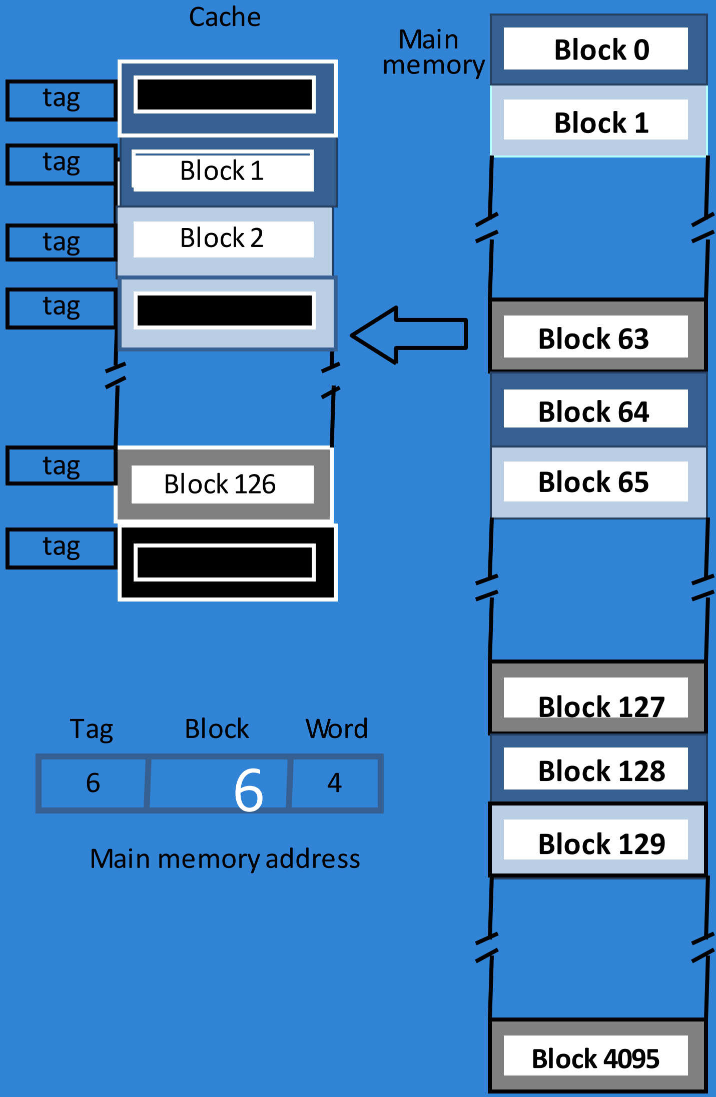
- *Block j of the main memory maps to j modulo 128 of the cache. 0 ,128,256..maps to block 0, 1,129,257 maps to Block1 and so on.*
- *More than one memory block is mapped onto the same position in the cache.*
- *May lead to contention for cache blocks even if the cache is not full.*
- *Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.*
- *Memory address is divided into three fields:*
  - *Low order 4 bits determine one of the 16 words in a block.*
  - *When a new block is brought into the cache, then the next 7 bits determines cache position in which this   block must be stored.*
  - *High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.*
- *Simple to implement but not very flexible.*

Associative mapping



- Main memory block can be placed into any cache position.
- Memory address is divided into two fields:
  - Low order 4 bits identify the word within a block.
  - High order 12 bits or tag bits identify a memory block when it is resident in the cache.
- Flexible, and uses cache space efficiently.
- Replacement algorithms can be used to replace an existing block in the cache when the cache is full.
- Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.

Set-Associative mapping



Blocks of cache are grouped into sets.
Mapping function allows a block of the main memory to reside in any block of a specific set.
Divide the cache into 64 sets, with two blocks per set.
Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.
Memory address is divided into three fields:

- *6 bit field determines the set number.*
- *High order 6 bit fields are compared to the tag fields of the two blocks in a set.*

Set-associative mapping combination of direct and associative mapping.
Number of blocks per set is a design parameter.

- *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
- *Other extreme is to have one block per set, is the same as direct mapping.*

The Memory System

Performance considerations

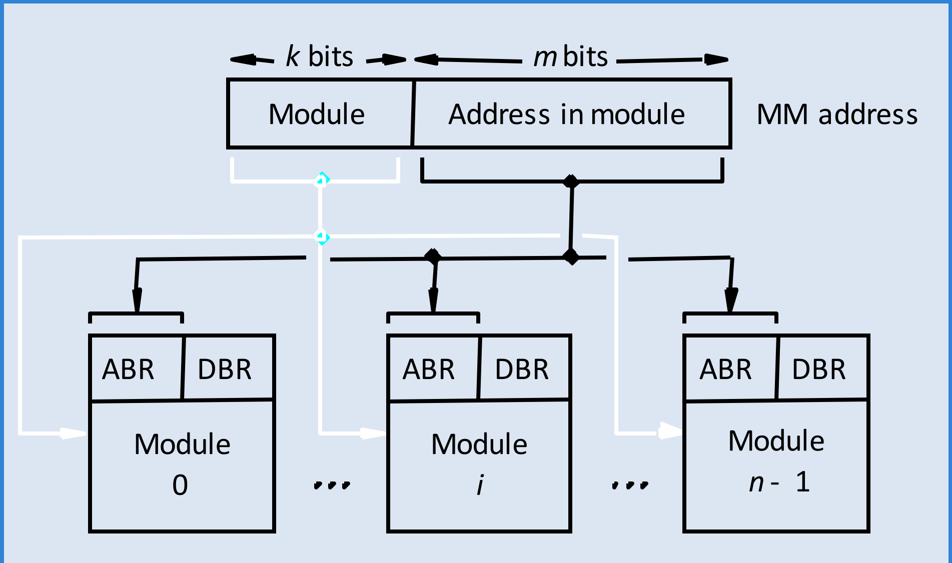
Performance considerations

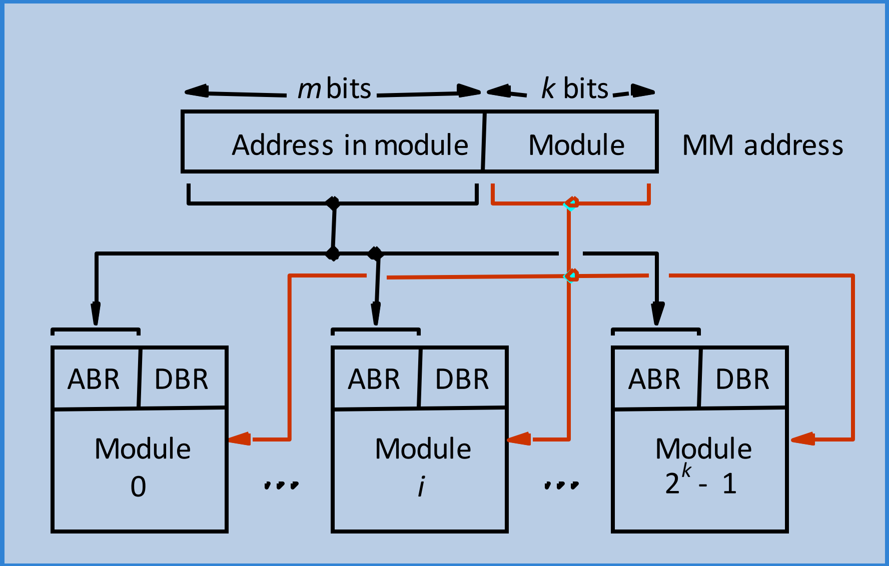
- A key design objective of a computer system is to achieve the best possible performance at the lowest possible cost.
  - Price/performance ratio is a common measure of success.
- Performance of a processor depends on:
  - How fast machine instructions can be brought into the processor for execution.
  - How fast the instructions can be executed.

Interleaving

- Divides the memory system into a number of memory modules. Each module has its own address buffer register (ABR) and data buffer register (DBR).
- Arranges addressing so that successive words in the address space are placed in different modules.
- When requests for memory access involve consecutive addresses, the access will be to different modules.
- Since parallel access to these modules is possible, the average rate of fetching words from the Main Memory can be increased.

Methods of address layouts





- *Consecutive words are placed in a module.*
- *High-order k bits of a memory address determine the module.*
- *Low-order m bits of a memory address determine the word within a module.*
- *When a block of words is transferred from main memory to cache, only one module is busy at a time.*

- ***Consecutive words are located in consecutive modules.***
- ***Consecutive addresses can be located in consecutive modules.***
- ***While transferring a block of data, several memory modules can be kept busy at the same time.***

Hit Rate and Miss Penalty

- Hit rate
- Miss penalty
- Hit rate can be improved by increasing block size, while keeping cache size constant
- Block sizes that are neither very small nor very large give best results.
- Miss penalty can be reduced if load-through approach is used when loading new blocks into cache.

Caches on the processor chip

- In high performance processors 2 levels of caches are normally used.
- Avg access time in a system with 2 levels of caches is

$$T_{ave} = h1c1+(1-h1)h2c2+(1-h1)(1-h2)M$$

Other Performance Enhancements

Write buffer

- *Write-through:*
- *Each write operation involves writing to the main memory.*
- *If the processor has to wait for the write operation to be complete, it slows down the processor.*
- *Processor does not depend on the results of the write operation.*
- *Write buffer can be included for temporary storage of write requests.*
- *Processor places each write request into the buffer and continues execution.*
- *If a subsequent Read request references data which is still in the write buffer, then this data is referenced in the write buffer.*

- *Write-back:*
- *Block is written back to the main memory when it is replaced.*
- *If the processor waits for this write to complete, before reading the new block, it is slowed down.*
- *Fast write buffer can hold the block to be written, and the new block can be read first.*

Other Performance Enhancements (Contd.,)

Prefetching

- *New data are brought into the processor when they are first needed.*
- *Processor has to wait before the data transfer is complete.*
- *Prefetch the data into the cache before they are actually needed, or a before a Read miss occurs.*
- *Prefetching can be accomplished through software by including a special instruction in the machine language of the processor.*
    - ***Inclusion of prefetch instructions increases the length of the programs.***
- *Prefetching can also be accomplished using hardware:*
    - ***Circuitry that attempts to discover patterns in memory references and then prefetches according to this pattern.***

Other Performance Enhancements (Contd..)

Lockup-Free Cache

- *Prefetching scheme does not work if it stops other accesses to the cache until the prefetch is completed.*
- *A cache of this type is said to be "locked" while it services a miss.*
- *Cache structure which supports multiple outstanding misses is called a lockup free cache.*
- *Since only one miss can be serviced at a time, a lockup free cache must include circuits that keep track of all the outstanding misses.*
- *Special registers may hold the necessary information about these misses.*

The Memory System

Virtual Memory

Virtual memories

- Recall that an important challenge in the design of a computer system is to provide a large, fast memory system at an affordable cost.
- Architectural solutions to increase the effective speed and size of the memory system.
- Cache memories were developed to increase the effective speed of the memory system.
- Virtual memory is an architectural solution to increase the effective size of the memory system.

Virtual memories (contd..)

- Recall that the addressable memory space depends on the number of address bits in a computer.
  - For example, if a computer issues 32-bit addresses, the addressable memory space is 4G bytes.
- Physical main memory in a computer is generally not as large as the entire possible addressable space.
  - Physical memory typically ranges from a few hundred megabytes to 1G bytes.
- Large programs that cannot fit completely into the main memory have their parts stored on secondary storage devices such as magnetic disks.
  - Pieces of programs must be transferred to the main memory from secondary storage before they can be executed.

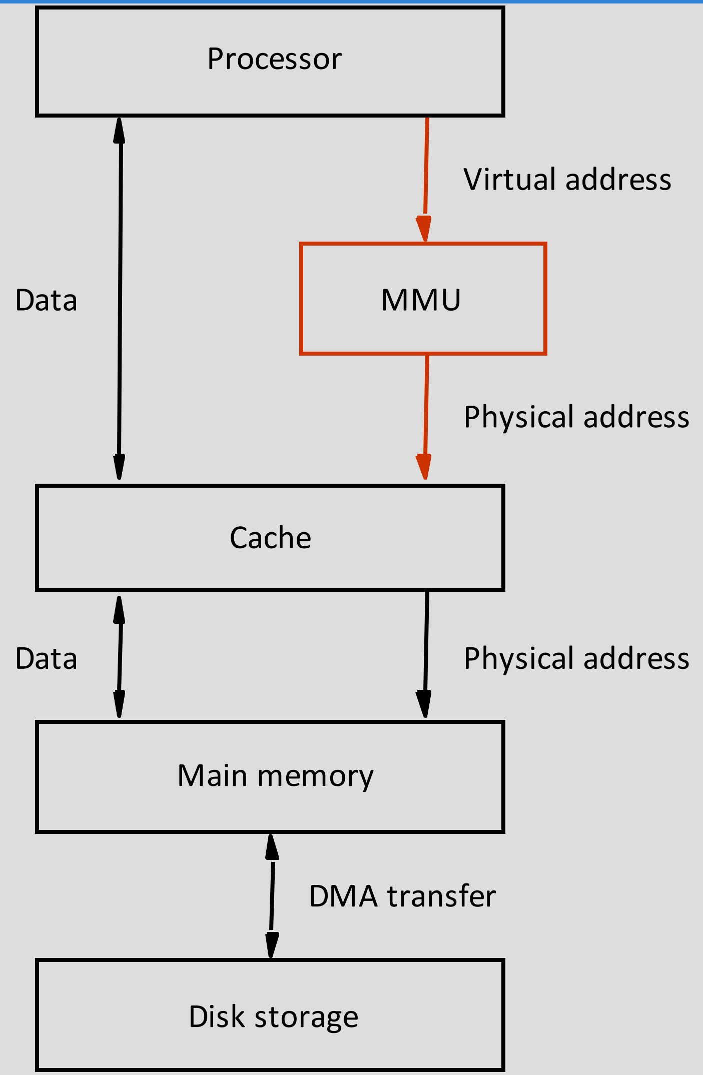
Virtual memories (contd..)

- When a new piece of a program is to be transferred to the main memory, and the main memory is full, then some other piece in the main memory must be replaced.
  - Recall this is very similar to what we studied in case of cache memories.
- Operating system automatically transfers data between the main memory and secondary storage.
  - Application programmer need not be concerned with this transfer.
  - Also, application programmer does not need to be aware of the limitations imposed by the available physical memory.

Virtual memories (contd..)

- Techniques that automatically move program and data between main memory and secondary storage when they are required for execution are called virtual-memory techniques.
- Programs and processors reference an instruction or data independent of the size of the main memory.
- Processor issues binary addresses for instructions and data.
  - These binary addresses are called logical or virtual addresses.
- Virtual addresses are translated into physical addresses by a combination of hardware and software subsystems.
  - If virtual address refers to a part of the program that is currently in the main memory, it is accessed immediately.
  - If the address refers to a part of the program that is not currently in the main memory, its contents must be brought to the main memory before it can be used.

Virtual memory organization



- *Memory management unit (MMU) translates virtual addresses into physical addresses.*
- *If the desired data or instructions are in the main memory they are fetched as described previously.*
- *If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.*
- *MMU causes the operating system to bring the data from the secondary storage into the main memory.*

Address translation

- Assume that program and data are composed of fixed-length units called pages.
- A page consists of a block of words that occupy contiguous locations in the main memory.
- Page is a basic unit of information that is transferred between secondary storage and main memory.
- Size of a page commonly ranges from 2K to 16K bytes.
  - Pages should not be too small, because the access time of a secondary storage device is much larger than the main memory.
  - Pages should not be too large, else a large portion of the page may not be used, and it will occupy valuable space in the main memory.

Address translation (contd..)

- Concepts of virtual memory are similar to the concepts of cache memory.
- Cache memory:
  - Introduced to bridge the speed gap between the processor and the main memory.
  - Implemented in hardware.
- Virtual memory:
  - Introduced to bridge the speed gap between the main memory and secondary storage.
  - Implemented in part by software.

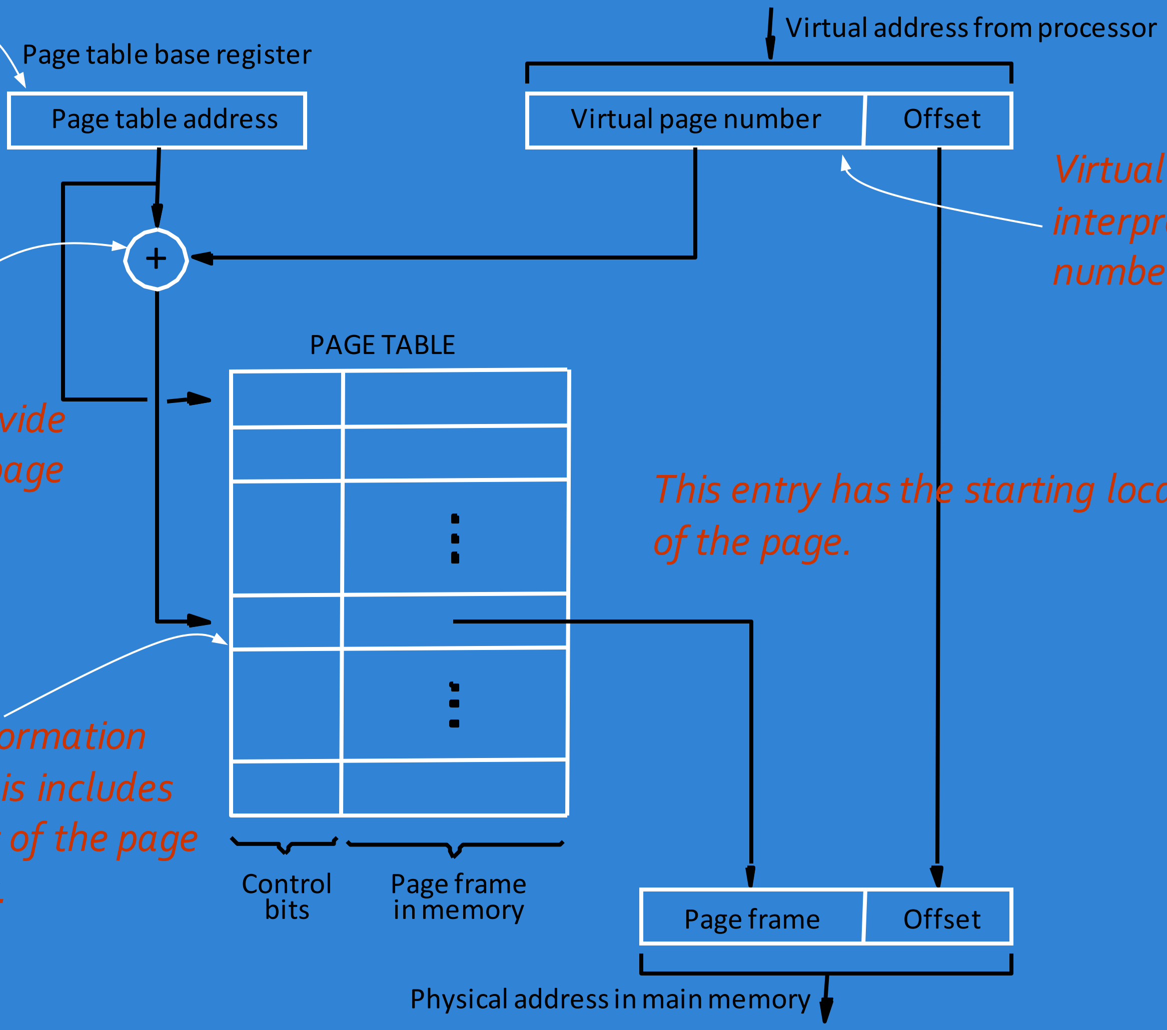
Address translation (contd..)

- Each virtual or logical address generated by a processor is interpreted as a virtual page number (high-order bits) plus an offset (low-order bits) that specifies the location of a particular byte within that page.
- Information about the main memory location of each page is kept in the page table.
  - Main memory address where the page is stored.
  - Current status of the page.
- Area of the main memory that can hold a page is called as page frame.
- Starting address of the page table is kept in a page table base register.

Address translation (contd..)

- Virtual page number generated by the processor is added to the contents of the page table base register.
  - This provides the address of the corresponding entry in the page table.
- The contents of this location in the page table give the starting address of the page if the page is currently in the main memory.

Address translation (contd..)



PTBR holds the address of the page table.

Virtual address is interpreted as page number and offset.

PTBR + virtual page number provide the entry of the page in the page table.

This entry has the starting location of the page.

Page table holds information about each page. This includes the starting address of the page in the main memory.

Address translation (contd..)

- Page table entry for a page also includes some control bits which describe the status of the page while it is in the main memory.
- One bit indicates the validity of the page.
  - Indicates whether the page is actually loaded into the main memory.
  - Allows the operating system to invalidate the page without actually removing it.
- One bit indicates whether the page has been modified during its residency in the main memory.
  - This bit determines whether the page should be written back to the disk when it is removed from the main memory.
  - Similar to the dirty or modified bit in case of cache memory.

Address translation (contd..)

- Other control bits for various other types of restrictions that may be imposed.
  - For example, a program may only have read permission for a page, but not write or modify permissions.

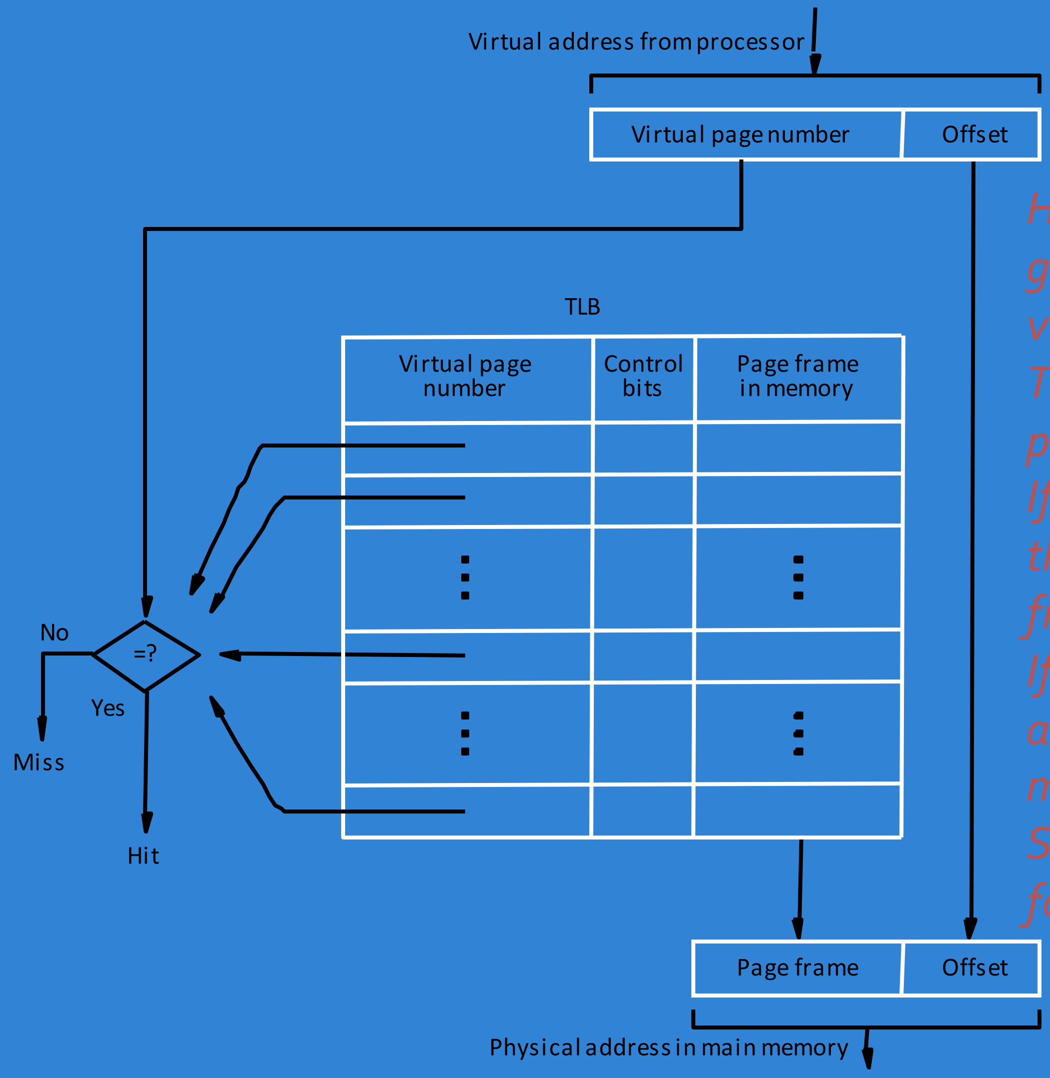
Address translation (contd..)

- Where should the page table be located?
  - Page table information is used by the MMU for every Read and Write access.
  - Ideal location for the page table is within the MMU.
- Page table is quite large.
- MMU is implemented as part of the processor chip.
- Impossible to include a complete page table on the chip.
- Page table is kept in the main memory.
- A copy of a small portion of the page table can be accommodated within the MMU.
  - This Portion consists of page table entries that correspond to the most recently accessed pages.

Address translation (contd..)

- A small cache called as Translation Lookaside Buffer (TLB) is included in the MMU.
  - TLB holds page table entries of the most recently accessed pages.
- Recall that cache memory holds most recently accessed blocks from the main memory.
  - Operation of the TLB and page table in the main memory is similar to the operation of the cache and main memory.
- Page table entry for a page includes:
  - Address of the page frame where the page resides in the main memory.
  - Some control bits.
- In addition to the above for each page, TLB must hold the virtual page number for each page.

Address translation (contd..)



Associative-mapped TLB

High-order bits of the virtual address generated by the processor select the virtual page.

These bits are compared to the virtual page numbers in the TLB.

If there is a match, a hit occurs and the corresponding address of the page frame is read.

If there is no match, a miss occurs and the page table within the main memory must be consulted.

Set-associative mapped TLBs are found in commercial processors.

Address translation (contd..)

- How to keep the entries of the TLB coherent with the contents of the page table in the main memory?
- Operating system may change the contents of the page table in the main memory.
  - Simultaneously it must also invalidate the corresponding entries in the TLB.
- A control bit is provided in the TLB to invalidate an entry.
- If an entry is invalidated, then the TLB gets the information for that entry from the page table.
  - Follows the same process that it would follow if the entry is not found in the TLB or if a “miss” occurs.

Address translation (contd..)

- What happens if a program generates an access to a page that is not in the main memory?
- In this case, a page fault is said to occur.
  - Whole page must be brought into the main memory from the disk, before the execution can proceed.
- Upon detecting a page fault by the MMU, following actions occur:
  - MMU asks the operating system to intervene by raising an exception.
  - Processing of the active task which caused the page fault is interrupted.
  - Control is transferred to the operating system.
  - Operating system copies the requested page from secondary storage to the main memory.
  - Once the page is copied, control is returned to the task which was interrupted.

Address translation (contd..)

- Servicing of a page fault requires transferring the requested page from secondary storage to the main memory.
- This transfer may incur a long delay.
- While the page is being transferred the operating system may:
  - Suspend the execution of the task that caused the page fault.
  - Begin execution of another task whose pages are in the main memory.
- Enables efficient use of the processor.

Address translation (contd..)

- How to ensure that the interrupted task can continue correctly when it resumes execution?
- There are two possibilities:
  - Execution of the interrupted task must continue from the point where it was interrupted.
  - The instruction must be restarted.
- Which specific option is followed depends on the design of the processor.

Address translation (contd..)

- When a new page is to be brought into the main memory from secondary storage, the main memory may be full.
  - Some page from the main memory must be replaced with this new page.
- How to choose which page to replace?
  - This is similar to the replacement that occurs when the cache is full.
  - The principle of locality of reference (?) can also be applied here.
  - A replacement strategy similar to LRU can be applied.
- Since the size of the main memory is relatively larger compared to cache, a relatively large amount of programs and data can be held in the main memory.
  - Minimizes the frequency of transfers between secondary storage and main memory.

Address translation (contd..)

- A page may be modified during its residency in the main memory.
- When should the page be written back to the secondary storage?
- Recall that we encountered a similar problem in the context of cache and main memory:
  - Write-through protocol(?)
  - Write-back protocol(?)
- Write-through protocol cannot be used, since it will incur a long delay each time a small amount of data is written to the disk.

The Memory System

Memory Management

Memory management

- Operating system is concerned with transferring programs and data between secondary storage and main memory.
- Operating system needs memory routines in addition to the other routines.
- Operating system routines are assembled into a virtual address space called system space.
- System space is separate from the space in which user application programs reside.
  - This is user space.
- Virtual address space is divided into one

  system space + several user spaces.

Memory management (contd..)

- Recall that the Memory Management Unit (MMU) translates logical or virtual addresses into physical addresses.
- MMU uses the contents of the page table base register to determine the address of the page table to be used in the translation.
  - Changing the contents of the page table base register can enable us to use a different page table, and switch from one space to another.
- At any given time, the page table base register can point to one page table.
  - Thus, only one page table can be used in the translation process at a given time.
  - Pages belonging to only one space are accessible at any given time.

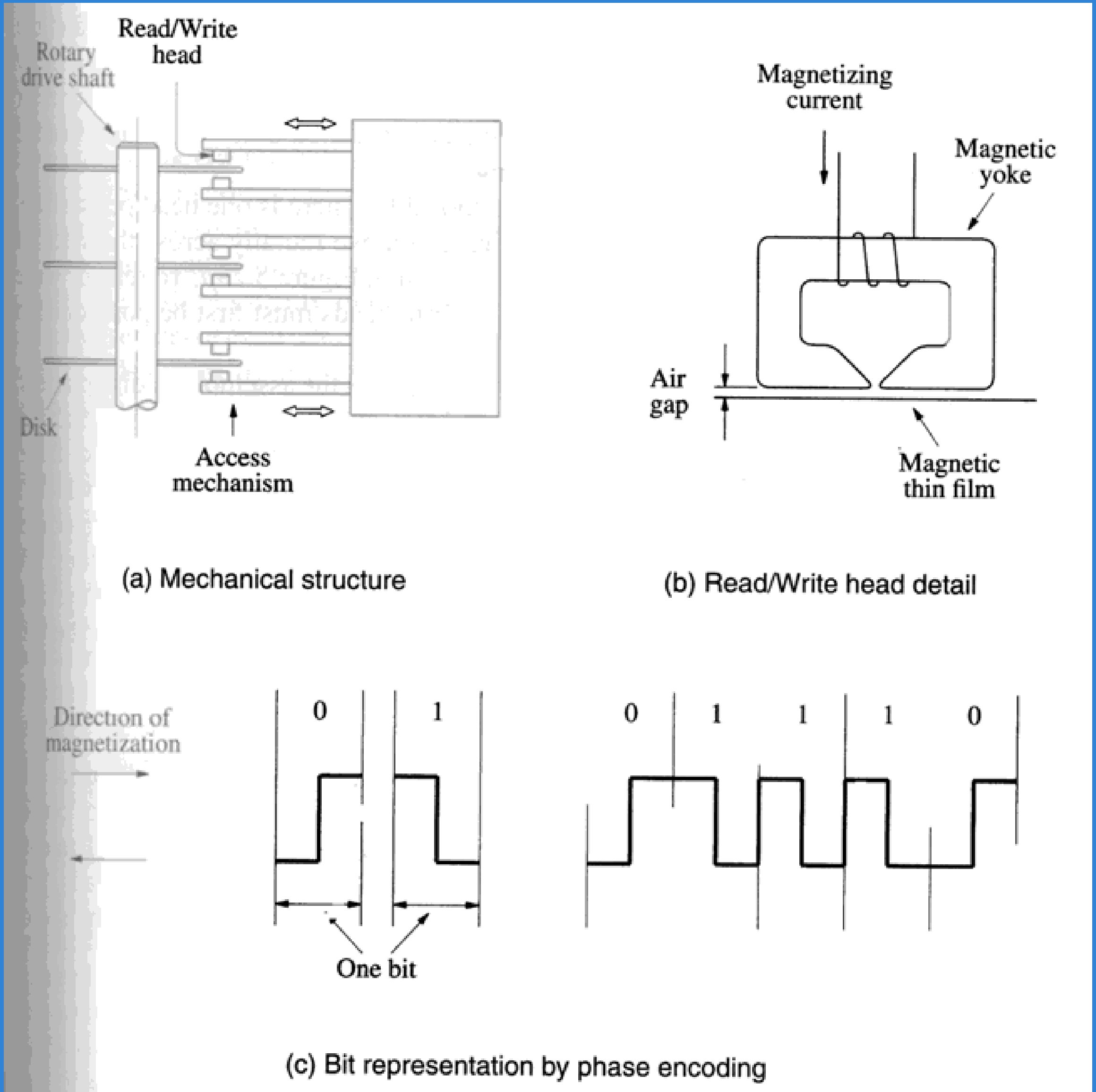
Memory management (contd..)

- When multiple, independent user programs coexist in the main memory, how to ensure that one program does not modify/destroy the contents of the other?
- Processor usually has two states of operation:
  - Supervisor state.
  - User state.
- Supervisor state:
  - Operating system routines are executed.
- User state:
  - User programs are executed.
  - Certain privileged instructions cannot be executed in user state.
  - These privileged instructions include the ones which change page table base register.
  - Prevents one user from accessing the space of other users.

The Memory System

Secondary Storage

Magnetic Hard Disks



(a) Mechanical structure

(b) Read/Write head detail

(c) Bit representation by phase encoding

Disk

Disk drive

Disk controller

Organization of Data on a Disk

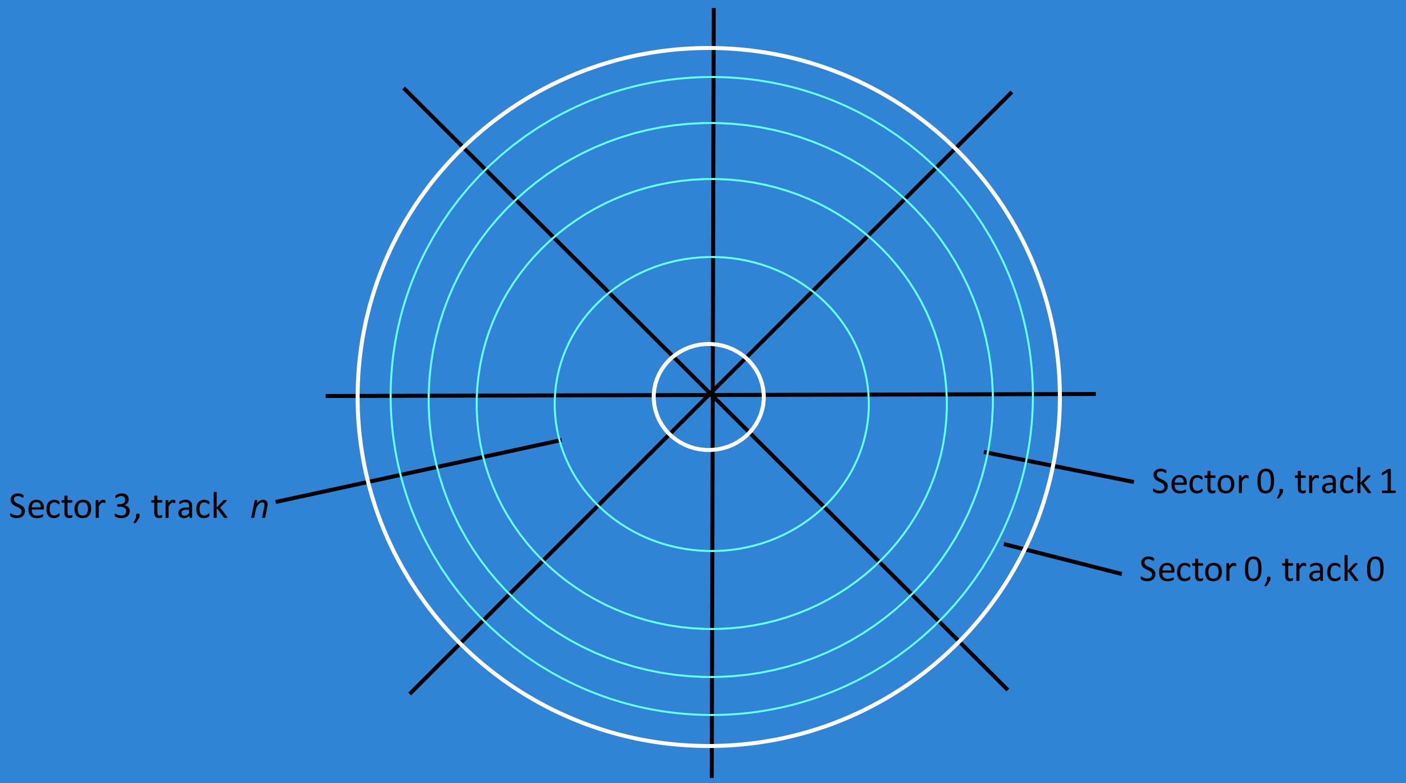


Figure 5.30. Organization of one surface of a disk.

Access Data on a Disk

- Sector header
- Following the data, there is an error-correction code (ECC).
- Formatting process
- Difference between inner tracks and outer tracks
- Access time – seek time / rotational delay (latency time)
- Data buffer/cache

Disk Controller

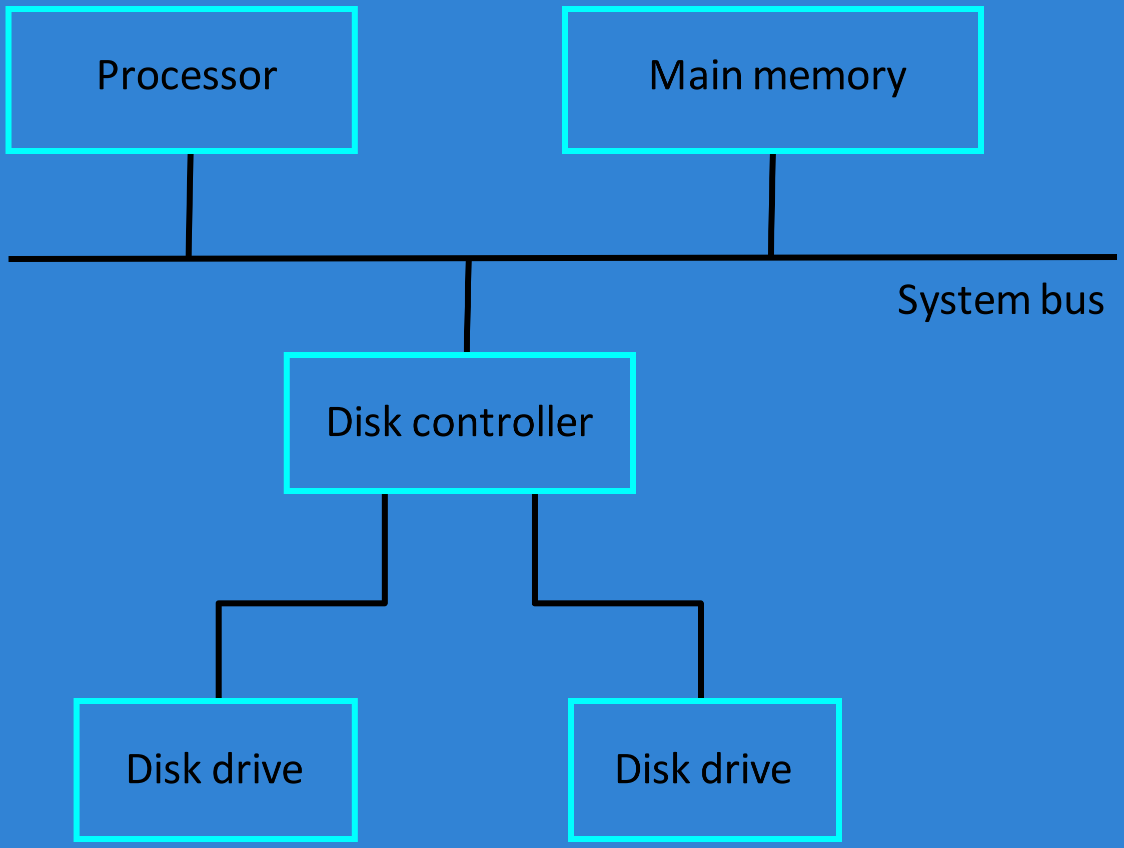


Figure 5.31. Disks connected to the system bus.

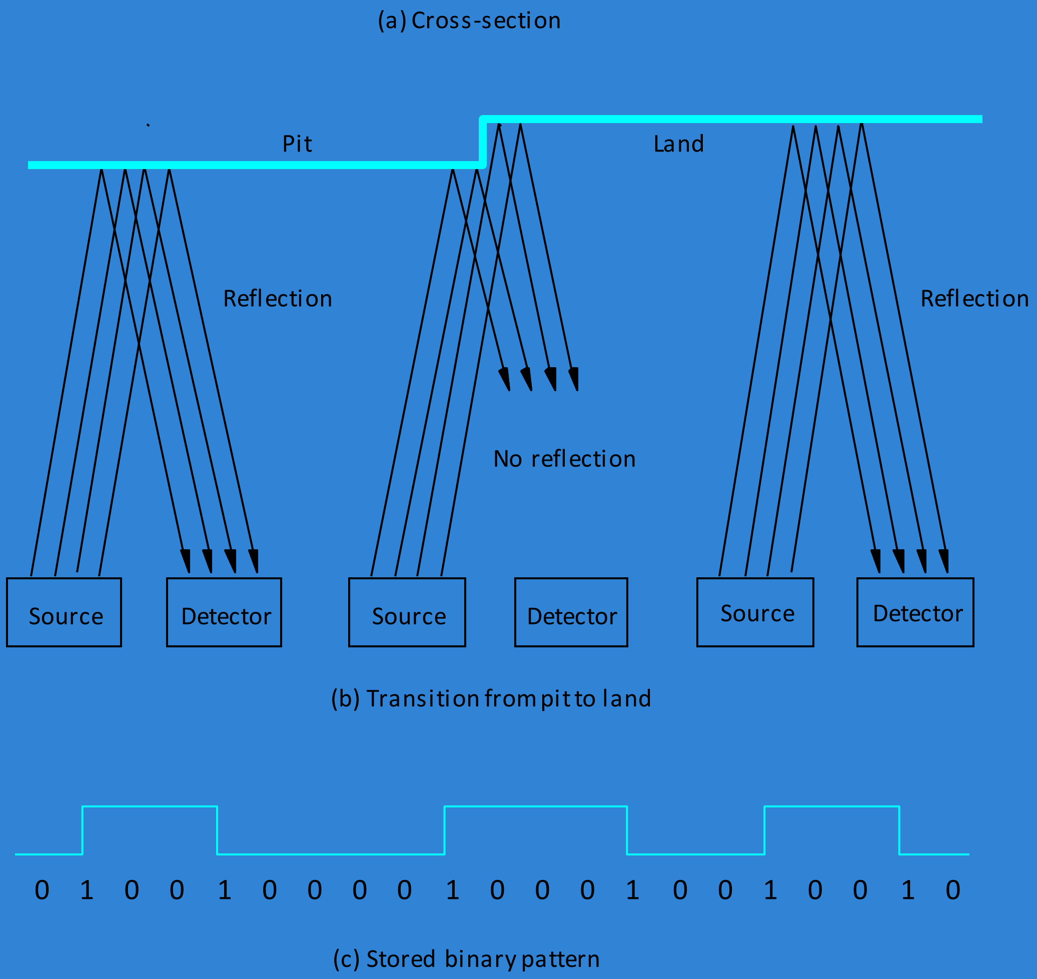
Disk Controller

- Seek
- Read
- Write
- Error checking

RAID Disk Arrays

- Redundant Array of Inexpensive Disks
- Using multiple disks makes it cheaper for huge storage, and also possible to improve the reliability of the overall system.
- RAID0 – data striping
- RAID1 – identical copies of data on two disks
- RAID2, 3, 4 – increased reliability
- RAID5 – parity-based error-recovery

Optical Disks



(a) Cross-section

(b) Transition from pit to land

(c) Stored binary pattern

Figure 5.32. Optical disk.

Optical Disks

- CD-ROM
- CD-Recordable (CD-R)
- CD-ReWritable (CD-RW)
- DVD
- DVD-RAM

Magnetic Tape Systems

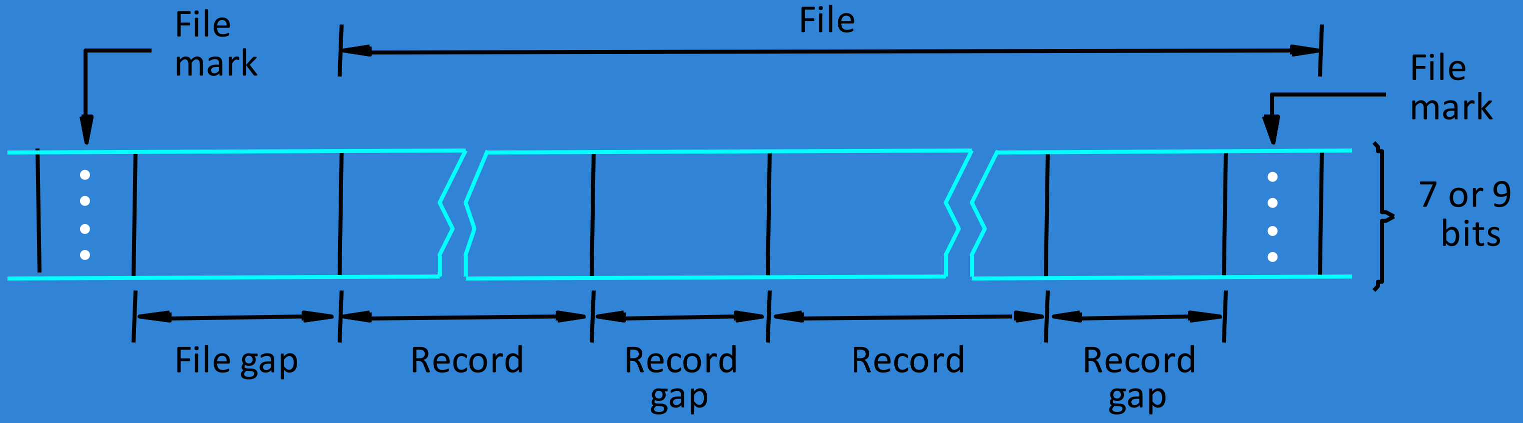


Figure 5.33. Organization of data on magnetic tape.