COMPUTER ORGANIZATION AND ARCHITECTURE (IT 2202)

Lecture 5-6

oftware and Architecture Type

- **A software or a program consist of a set of instructions required to solve a specific problem. Example**
- **A program to sort ten numbers.**
- **A program to add some numbers.**
- **A program to find out inverse of a matrix.**
- **The compiler translates a C program to machine language. It converts the high level language into machine coded language. All of these are a kinds of software.**
- **Software consists of a set of instructions. A set of instructions are provided to perform certain task. The operating system is also an software that helps us in using the computer system.**

Types of Programs

- **Broadly we can classify the programs/software into two types.**
- **An application software helps the user to solve a particular user level problem and it may require a system software for execution.**
- **A system software is basically collection of many programs that helps the user to create, analyze and run their programs.**
- **So, this is very important to know that we have two kinds of programs; one kind is application software, another kind is system software.**

Types of Programs

Application Software

- This helps the user to solve a particular problem.
- In most cases, application software resides on the computer's hard disk or removable storage media (DVD, USD drive, etc.)
- Typical examples:
  - Financial accounting package to do some kind of financial accounting stuffs.
  - Mathematical package like MATLAB or Mathematica, which is used to perform a particular kind of mathematical operations and various programs can be written using that.
  - an mobile app to call a cab that is also an application
  - An app to monitor the health of a person
  - Other apps to do various other functions.
  - All these comes under application software.

Types of Programs

System Software

- **A system software is a collection of programs which help user to run other programs.**

- **Typical operations carried out by system software:**
  - **handles user requests**
  - **Managing application programs and storing them as files**
  - **File management in secondary storage devices**
  - **Running standard applications such as word processor, internet browser, etc,**
  - **Managing input/output unit**
  - **Program translation from source form to object form.**
  - **linking and running user program and many others.**

Types of Programs

Some Commonly used System Software:

- Operating Systems (Windows, Linux, MAC/OS, ANDROID, etc.)
  - Instance of a program that never terminates.
  - The program continues running until either the machine is switched off or the user manually shuts down the machine.
  - If the machine is shut down, the system software will automatically stop, but as long as your system is running those software will be running.
- Compilers and assemblers
- Linkers and loaders
- Editors and debuggers.

Types of Programs

Operating System:

An operating system is a system software as it provides an interface between the computer hardware and the user. So, the interface between the hardware (processor/memory/input-output) and user is done through system software. The user actually interacts with the hardware through an operating system. So, operating system is sitting in between the user and the hardware. The hardware and the user talk to each other through this operating system.

Two layers:

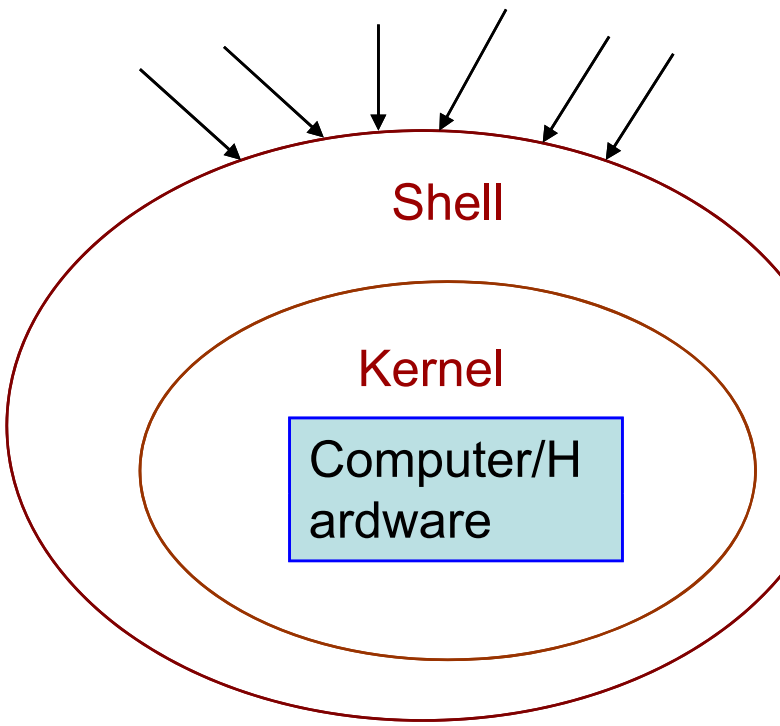
a) **Kernel: contains low-level routines for resource management.**

a) **Shell: provides an interface for the users to interact with the computer hardware/ through the kernel.**

Types of Programs

There are two layers:-kernel layer contains the low level routine.

The hardware sits in the bottom, and then we have a kernel which contains low level routines for resource management, and then we have a shell. Actually, all users and application software cannot access this computer hardware directly.



Through this shell, it provides an interface for the user to interact with the computer hardware through the kernel. Kernel is sitting between shell and computer hardware; and between kernel and user, shell is sitting. So, shell is an interface for the user or the application software to interact with the computer hardware through this kernel.

Types of Programs

Operating system is a collection of routines that are used to control sharing of variou computer resources as they execute application programs.

When we execute a program, some set of instructions get executed. For executing suc instruction, we have to load those instruction into the memory, and then from memory it brought into the processor and each time it is executed.

Operating system determines how the various resources can be managed and allocated t processor.

When the processor will be allocated to a particular process at what time, it is up to th operating system to decide upon.

The task includes

- **assigning memory and disk space to programs and data files.**
- **Moving data are moved between IO devices, memory and disk units.**
- **Handling input/output operations with parallel operations.**
- **Handling multiple user programs that are running at the same time. There are man more task that an OS performs.**

Types of Programs

Depending on the use of computer system, the goal of an operating system may differ:-

In classical multi programming system, there are several user programs loaded i memory and the OS can switch to another program when any other program is blocke for I/O. Suppose, a program is running and at that point of the time that program need some I/O, an I/O request has come for that particular program. So, it requires the time t handle that IO request and the processor can switch to another task and that another tas can be assigned to the processor and run. So, the switching from one task to another the main goal of classical multi programming system.

Main objective is to maximize the resource utilization. CPU must not sit idle. If it doing some particular task and at a time that particular task requires some othe resources for completion, then the processor has the flexibility to bring another task an execute this task, and later when that particular task has completed its I/O operation, tha task can get executed.

Types of Programs

Modern time sharing systems are widely used because every user can now afford to have a separate terminal.

The processor time is shared among number of interactive users.

Main objective is to reduce the user response time.

The user has requested for a task and by what time that particular request can be taken care of, so that is the user response time.

Types of Programs

In real time systems, there is a time constraint associated with it. Whenever there is a time constraint associated with it, there is a specific deadline associated with the task. These deadlines can be hard or soft. By hard deadline, we mean that that particular task has to finish within that particular time.

By soft deadline, we mean that if that deadline is not met, the system will not fail, but in a hard real time system if the deadline is not met, the system may fail.

Interrupt driven operations are also there where the processor is interrupted when a task arrives, this happens for some sporadic real time tasks, where there is no fixed time for task arrival. We do not know at what time a task will arrive.

Examples are missile control system, industrial manufacturing plant, patient health monitoring and control, and automotive control systems.

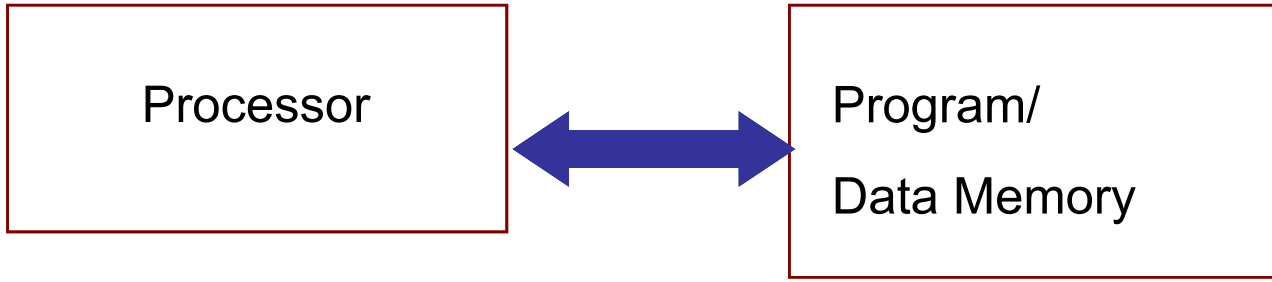
Types of Programs

- **For mobile system, the user responsiveness is most important.**
- **Sometime a program makes the system slow.**
- **We are running some programs in mobile.**
- **In mobile, we have limited memory, limited capability of certain resource. We are putting everything in a very small space. If the computer is not able to handle it and it is not able to stop those programs, it is forcibly stopped basically.**

ypes of Computer Architecture

Broadly the computer architecture can be classified into two types

- Von-Neumann architecture
- Harvard architecture



Von-Neumann architecture supports stored program concept, where we load both the progra and data into the same memory and the programs are executed one by one and whatever data required, data are also read.

In von-Neumann architecture, both the instructions that is the program and the data are stor in the same memory module. This fig shows the architecture consisting of memory module a processor where both the program and the data are stored in same memory and it is very flexib and easier to implement. This architecture is suitable for most of the general purpose processor.

Bottleneck and disadvantage:- This architecture has bottleneck. Here, we have loaded both t programs and data into the memory. There may be a situation that we need to access both t program that is the instruction and the data at the same time, we cannot do that because we ha a single memory where both my program and data are stored.

Types of Computer Architecture

In Harvard Architecture, we have separate memory for program and data.

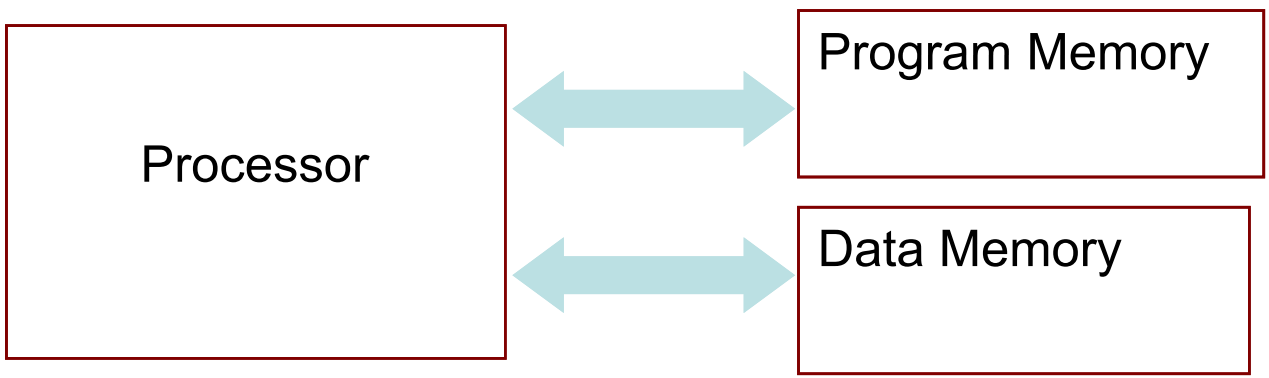
Instructions are stored in program memory and data are stored in data memory.

We have a program memory and a separate data memory. Instruction and data access ca be done in parallel.

The processor can access the program memory and the processor can also access the dat memory at the same time.

Some of the microcontrollers and pipelines with separate instruction and data cache follow this concept.

But here also this processor and memory bottleneck still remains. The processor wi access the memory, processor will access the data, but multiple data cannot be brought i at the same time.



Types of Computer Architecture

Emerging Architecture falls beyond von-Neumann architecture. Several architectura concepts have been proposed that deviate significantly from von-Neumann architecture.

In memory computing architecture, the storage and the computation can be done in th same functional unit. So, this is an emerging area, researchers are still working on i looking into various aspects of it.

It is projected that a circuit element called memristor; we have heard of resisto capacitor, inductor. So, memristor is another circuit element which is projected t make it possible in near future.

Memristors can also be used in high capacity non volatile resistive memory system and can also be controlled to carry out some computation. So, memristor can be use as memory, memristor can also be used for logic computation. Because of thes features we are saying that we can have some kind of in memory computing usin memristor in near future.

Types of Computer Architecture

We have separate program memory and data memory. The instructions are stored i instruction memory and data are stored in data memory.

Pipeline is a concept that is used to speed up the execution of instructions. An instructio execution is typically divided into five stages:-

- Fetch the instruction,
- Instruction decode. After decoding the instruction, we execute that instruction. Afte execution of that instruction, it may be required that we need to store the result int memory or we need to store the result into any of the processor register as well.
- In write back stage, the result is written to register file in the 5^{th} stage.
- An instruction execution cycle is basically divided into these five broad step instruction fetch, decode, execute, memory operation and write back.
- These five stages can be executed in an overlapped fashion. We are not doing paralle processing rather we are doing some kind of overlapped execution of instructions.
- This results in a significant speed up by overlapping instruction execution.

ypes of Computer Architecture

ovement of instructions from one stage to other under different the time steps or clock cycles. Assu

e instructions are coming one by one.

clock cycle 1, first instruction *i* enters the IF stage; instruction i, is fetched. After it goes to the
age in cycle 2. While it is being decoded, next instruction can be fetched. After this is done, in 3 cy
struction 'i' will go to EX stage, instruction i+1 moves to ID stage, and instruction (i+2) (3rd) can
to IF. In 4th cycle, instruction i moves to MEM stage, (i+1) moves to EX stage, (i+2) moves to ID st
d instruction (i+3) enters in IF stage. In 5th cycle, instruction i moves to WB stage, (i+1) move
EM stage, (i+2) moves to EX stage, (i+3) moves to ID stage and instruction (i+4) enters into IF stag
fter 5 cycle instruction 'i' will finish, after cycle 6, instruction (i+1) will finish time, after cycle 7, (
ll finish, after cycle 8, (i+3) will finish and after cycle 9, (1+4) will finish . Now, the initial delay,
ts filled up. In the ideal pipeline case, one instruction will be completed every clock cycle.

Instr. i+1 finishes

Instr. i+2 finish

Instruction number	Clock Cycles or time steps								
	1	2	3	4	5	6	7	8	9
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Instr. i+1 finishes

Instr. i+3 finishes

Instr. i+4 finishes

Types of Computer Architecture

- **In clock cycle 4, instruction-4 is trying to fetch an instruction from memory, while instruction 1, may be trying to access some data from memory.**

- **In von-Neumann architecture, one of these two operations will have to wait resulting in pipeline slowdown.**

- **In Harvard architecture, both the operations can be done parallelly because we have separate data memory and separate program memory.**

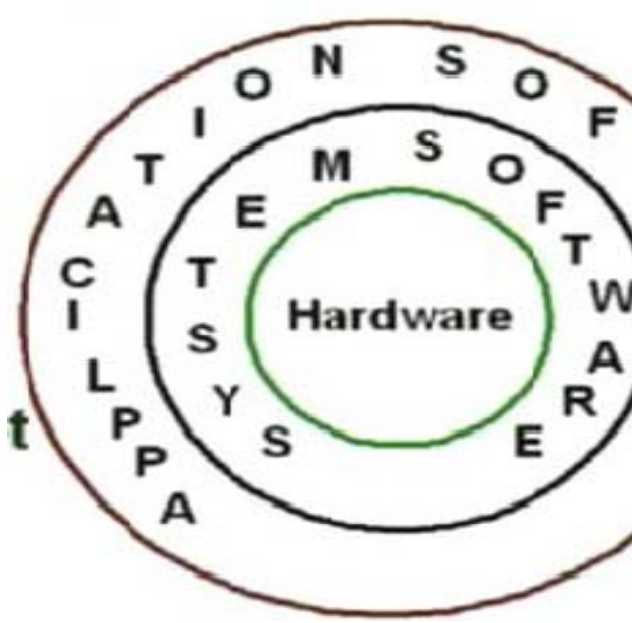
- **We have seen various software that are existing like application software, system software. The various kind of architecture is existing, i.e. von-Neumann architecture, and the Harvard architecture.**

- **Harvard architecture actually helps in executing an instruction in a better and faster fashion.**

Instruction Set Architecture

A modern computer is an integrated system consisting of

- Machine hardware [Processor etc]
- System software
- Application programs



- System architecture is represented by three circles
- Functionality of a processor is characterized by its instruction set
- Interface between the application software and hardware is the system software.
- A processor can execute a set of instructions, which can be used to write a program.
- A program can be considered as a sequence of instructions.
- Predefined instruction set is called instruction set architecture.

Instruction Set Processor Design

ISA (instruction set architecture) serves as an interface between the hardware and software

In terms of processor design methodology, an ISA can be considered as the specification of a design.

Specification is the behavioral description of what the processor can do?

Synthesis step attempts to find an implementation based on the specification

Processor is the implementation of the design

Instruction processor design concerns with how a processor is constructed. It is referred to as a micro architecture.

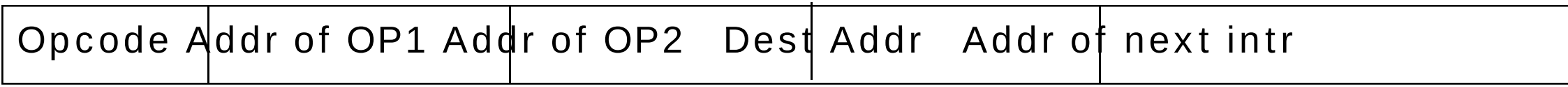
Realization of the physical implementation of the design is done using VLSI technology.

Instruction Set Architecture

- To command a computer, we must understand its language.
  - Instructions: words in a computer's language
  - Instruction set: the vocabulary of a computer's language
- Instructions indicate the operation to perform and the operands to use.

Instruction Set Architecture

- **What Information an Instruction should convey to the CPU?**



- **Instruction is too long if everything is specified explicitly**
  - **More space in memory**
  - **Longer execution**
- **Can we reduce the size of an instruction?**
  - **Specifying information Implicitly**
  - **How?**
- **Using Program Counter, Accumulator, General Purpose Registers, Stack Pointer**

Instruction Set Architecture

- Instruction Set Architecture is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct (time independent) program for that machine.

- ISA defines
  - Operations that the processor can execute
  - Data Transfer mechanisms plus how to access data
  - Control Mechanisms (branch, jump, etc)
  - Contract between programmer/compiler and HW

- ISA is important
  - Not only from the programmer's perspective
  - From processor design and implementation perspectives as well.

Instruction Set Architecture

- Programmer visible part of a processor:
  - Registers (Where are data located?
  - Addressing Modes (How is data accessed?)
  - Instruction Format (How are instructions specified?)
  - Exceptional Condition (What happens if something goes wrong?)
  - Instruction Set (What operations can be performed?)

ISA Design Choices

- **Types of operation supported**
  - **E.g. arithmetic/logical, data transfer, control transfer system, floating-point decimal (BCD), String.**
- **Types of operands supported**
  - **e.g. byte, character, digit, half word, word (32 bits), double word, floating points number.**
- **Types of operands storage allowed**
  - **e.g. stack, accumulator, registers, memory**
- **Implicit vs. explicit operands in the instructions and number of each**
- **Orthogonality of operands, operand location, and addressing modes**

Classification of ISAs

- **Determined by the means used for storing data in CPU**
- **Major choices are:**
  - **A stack, an accumulator, or a set of registers**
- **Stack architecture:**
  - **Operands are implicitly on the top of the stack**
- **Accumulator architecture**
  - **One operand is in the accumulator (register) and the others are elsewhere [IBM 701]**
  - **Essentially this is a 1 register machine**
  - **Found in older machines**
- **General purpose registers**
  - **Operands are in registers or specific memory locations**

  [IBM 360, intel 80586, MIPS]

Instruction Set Architecture

- encoded as binary numbers in a format called *machine language*.
- Machine language: computer-readable format (1's and 0's)
- Writing/reading code using machine language is tedious
- Assembly language: human-readable format of instructions

Instructions

- **Instructions for arithmetic**
- **Instructions for data movement**
- **Instructions for decision making**
- **Instructions for handling constant operands**

Instructions

- **Language of the Machine**
- **Primitive compared to HLLs**
- **Easily interpreted by hardware**

Instruction set design goals

- **maximize performance**
- **minimize cost,**
- **reduce design time**

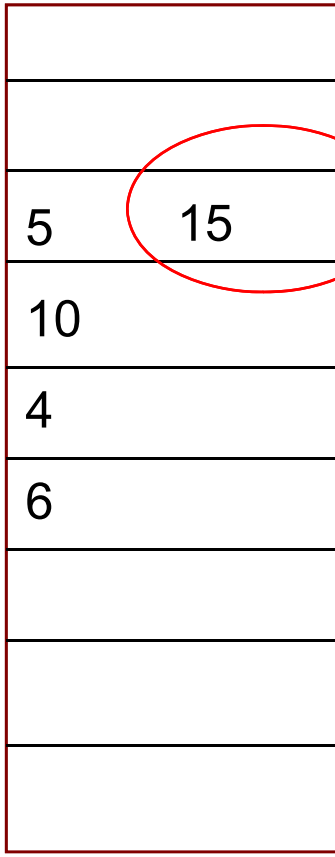
Instruction Set Architecture

- **Accumulator based systems**

By accumulator-based system, a processor register known as Accumulator is used to store one of the operand and store the result of the operation. Example: ADD R1; R1 is added with the content of the accumulator and the result is stored back in accumulator itself.

- **Stack based systems**

In stack-based architecture, a portion of memory called stack is made available. Data on the top of the stack can be accessed. If we just say ADD, and do not specify any operand, then by default the first two elements of the stack will be taken out. Here, 5 and 10, will be added and stored back. This becomes 15. We are storing the data in the stack, and then we are performing the operation where we need not have to specify any operand --- by default the operands are taken from the stack. So, it is a 0-address instruction.



Instruction Set Architecture

- **Memory-memory based Architecture**

This architecture support both two-address and three-address instructions.

Example: ADD A,B, where the data from memory location A is added with data from memory location B and the result is stored back in A.

- **Register-memory based systems**

One operand will be a register and one operand will be a memory location.

Example: LOAD R1,X, where load from memory location pointed by X into R1.

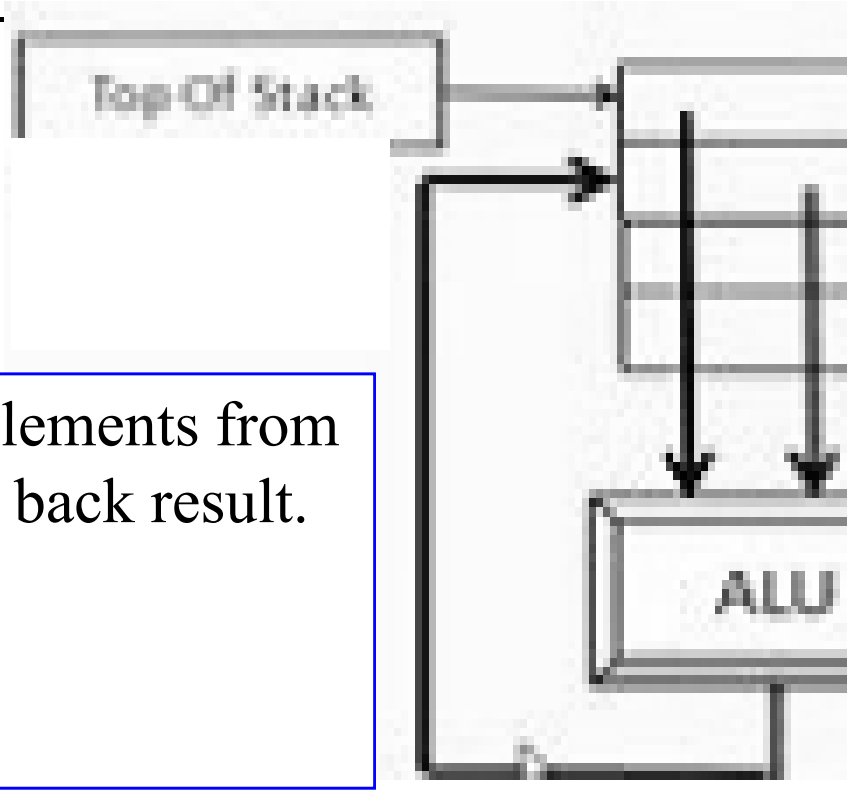
Similarly, STORE R3, store the value of R3 into some other location.

- Finally, we have three-address instructions where we are specifying three addresses
- ADD R1, R2, R3; the result of the addition (R2+R3) is stored in R1.

struction Set Architecture

Example code sequence Z = X + Y

We will use the various instruction set architectures let us consider this stack-based machine.



STACK MACHINE

PUSH X
PUSH Y
ADD
POP Z

ADD Instruction pops two elements from stack, adds them and pushes back result.

First we have to push X. Next we have to push Y, and then both of these are now in two position of the stack.

When we perform this ADD, these two values are taken out, added and stored back in top of the stack. Finally, when we do pop, then the value from top of the stack is taken and stored back in Z.

by PUSH X, X is added, by PUSH Y, Y is added. Once we perform ADD, X + Y is ac and stored back. When we perform POP Z, then Z is a memory location where we will the result.

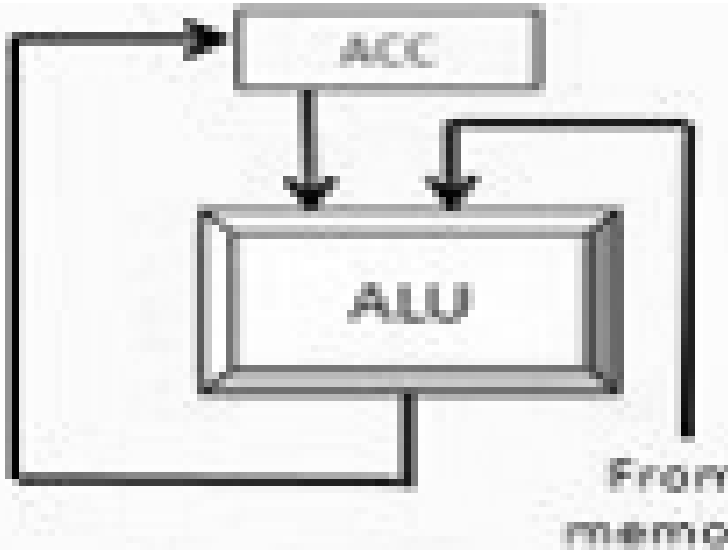
struction Set Architecture

Example code sequence Z = X + Y

let us consider the Accumulator-based machine.

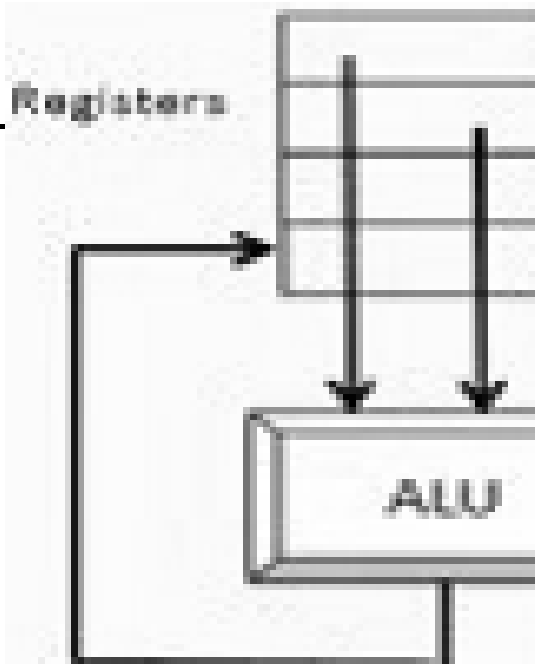
LOAD X
ADD Y
STORE Z

All instructions assume that one of the operands (and also the result) is a special register called accumulator



In an accumulator-based system, we perform operation Z = X + Y. Both X and Y are some values stored in memory. We load X in accumulator. Then ADD Y will actually add the content of location Y with accumulator and store back the result in accumulator. In the ALU, one value is coming from the accumulator and another is coming from memory. We are adding these two values and we are storing back the result in accumulator. Here, one of the operands and also the result is in a special register called accumulator.

struction Set Architecture



Example code sequence Z = X + Y

let us consider the Register-Register machine.

```
LOAD R1,X
LOAD R2,Y
ADD R3,R1,R2
STORE Z,R3
```

Also called load-store architecture, as only LOAD and STORE instructions can access memory.

In register-register machine, we have to load everything into some register first. Only operation is performed only on registers and not on any memory location. That is why i is called register-register machine.

First load all the values of the locations memory location into some register. We are using two back-to-back LOAD. In the first LOAD, value of X will be loaded in R1, and in next the value of Y will be loaded in R2. Now we can add these two registers and store the value in R3. Now, finally, we have to store the result in Z. This kind o architecture is also called load-store architecture.

By load-store architecture, only these two instructions LOAD and STORE will be used to access the memory, and no other instructions will be used to access the memory.

General Purpose Registers

Older architectures have large number of special-purpose register like program cou stack pointer, some index register, flag registers, accumulator, etc.

In newer architectures, we have more number of general-purpose registers. Instea using special purpose registers, most of the operations are performed using gen purpose registers.

The compiler can assign some variables to registers. So, there are so many variables can be used and they can be assigned it to registers, and registers are much faster t memory.

So, once the data is loaded into the registers and we are performing operation within register it will be much faster. But first we have to load the data from the memor register and then only we can perform the operation within register.

More compact instruction encoding is possible as fewer bits are required to spe register. So, the instruction encoding can be much less when we are bringing everyt into registers and the registers cannot be unlimited as compared to the memory addres If memory addresses a 32-bit, memory addresses will have 32-bit to encode, but if we h 40 register, we will require a maximum of 7-bits to encode. So, many processors hav or more general purpose registers.

COMPUTER ORGANIZATION AND ARCHITECTURE (IT 2202)

Lecture 6

Prof Hafizur Rahaman
Indian Institute of Engineering Science and Technology, Shibpur
(Formerly, Bengal Engineering and Science University, Shibpur)
Howrah 711 103, West Bengal, India

nstruction Format and Addressing Modes

Instruction comprises of two parts--- first part is Operation or Opcode, and the next part is the Operand.

The opcode specifies the operation to be performed.

There are various categories of instructions:-data transfer, arithmetic, and logical control, I/O and special machine control.

In the Example: (ADD R1, R2) is adding two register values, and result stored back in some register. So, R1 will store R1 + R2. ADD is the operation code that specifies the operation to be performed by the instruction. This is an arithmetic instruction.

In Example, MOVE R2, R1, this instruction will move the data from R2 to R1. So, R1 will have the value of R2. This instruction is called data transfer instructions.

In example JUMP 8000, the program control is transferred to address 8000. this is branching instruction. So, branch to this particular location will move to location 8000, and the location will be added into PC and from this location, the next instruction will be executed because branch to this location means in some particular location some instruction is present which will be execute.

nstruction Format and Addressing Modes

The next part is operand.

An operand specifies either a single source or two sources and a destination of the operation. Source operand can be specified by an immediate data or by naming a register.

Example ADDI 2000, it is example of immediate data and it is adding 2000 with the content of the Accumulator (A) and the result will be saved in A.

Example ADD R1, MEMORY, here we are giving the name of the register or specify a memory address. Here, we are specifying one operand in a register, another operand a memory location. So, a register, a memory location or an immediate value will be used.

A destination operand should always be either a register or a memory location like "MEMORY".

Instruction Format

- **What Information an Instruction should convey to the CPU?**

Opcode	Addr of OP1	Addr of OP2	Dest Addr	Addr of next intr.

- **Instruction is too long if everything is specified explicitly**
  - **More space in memory**
  - **Longer execution**
- **Can we reduce the size of an instruction?**
  - **Specifying information Implicitly**
  - **How?**
- **Using Program Counter, Accumulator, General Purpose Registers, Stack Pointer**

Addressing Modes

The number of operands varies from instruction to instruction.

We can have zero address instruction, one address instruction.

We can even have a two address or even have a three address instruction.

Number of operands present in an instruction may vary. While specifying an operand, we need to know the various addressing modes.

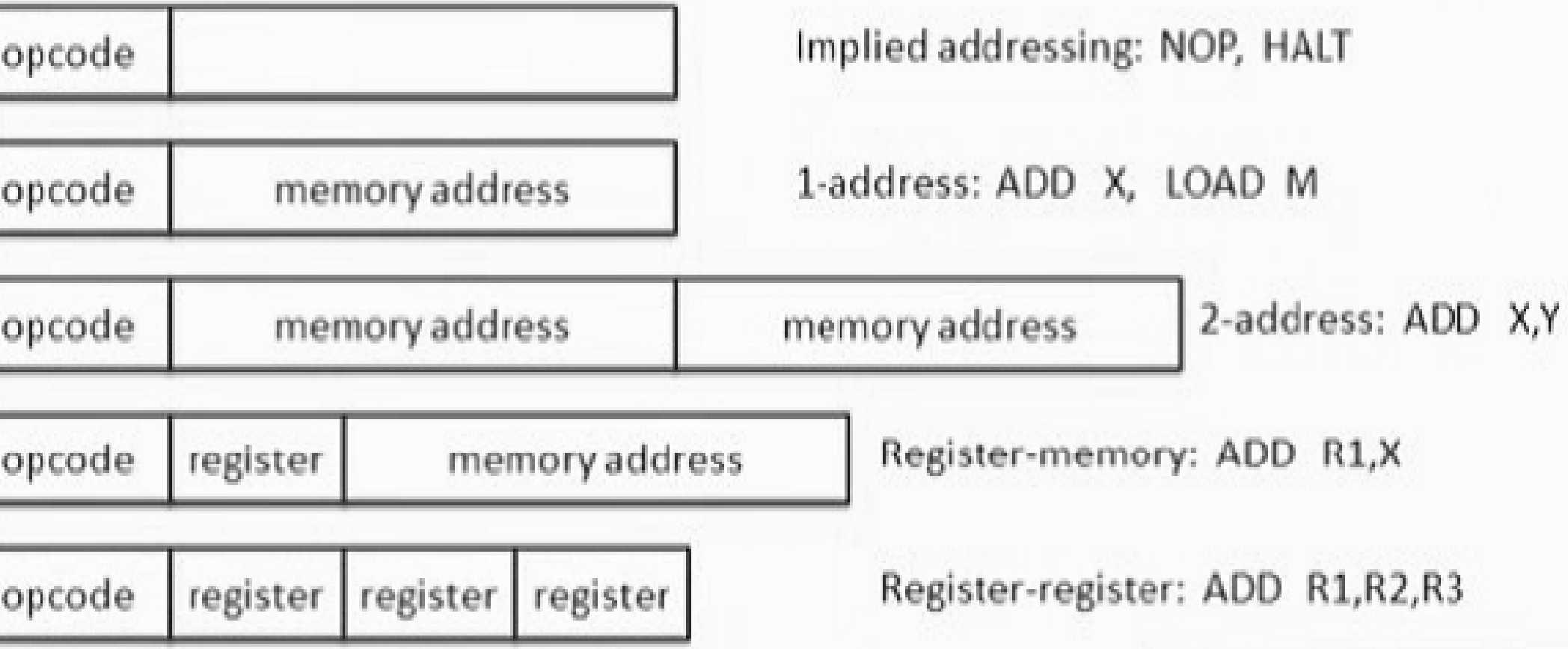
Addressing mode actually is a way by which the location of the operand is specified in the instruction.

We will see how is the address of an operand specified, and how are the *bits* of an instruction organized to define the *operand addresses* and operation of that instruction.

There can be many possible addressing modes: immediate, direct, indirect, relative and index, and many more.

ddressing Modes

nstruction Formats



ddressing Modes

Addressing Modes:

The most common addressing techniques are:

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

ddressing Modes

All computer architectures provide more than one of these *addressing modes*. Th *control unit* can determine which addressing mode is being used in a particul instruction. Several approaches are used. Often, different *opcodes* will use differe addressing modes. Also, one or more bits in the *instruction format* can be used as *mode field*. The value of the mode field determines which addressing mode is to b used.

In a system without virtual memory, the effective address will be either a main memo address or a register. In a virtual memory system, the effective address is a virtu address or a register. The actual mapping to a physical address is a function of th paging mechanism and is invisible to the programmer.

To explain the addressing modes, we use the following notation:

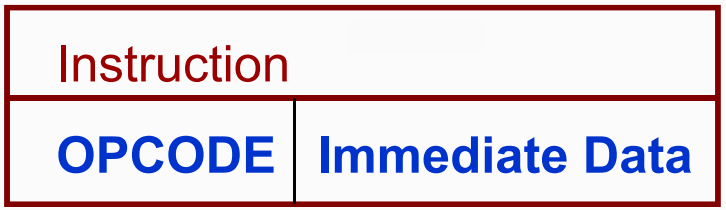
A = contents of an address field in the instruction that refers to a memory
R = contents of an address field in the instruction that refers to a register
EA = actual (effective) address of the location containing the referenced operand
(X) = contents of location X

ddressing Modes

mmediate Addressing:

The simplest form of addressing is immediate addressing, in which the operand is actually present the instruction. In immediate addressing mode, here the operand is part of the instruction itself.

OPERAND = Immediate data



This mode can be used to define and use constants or set initial values of variables. The advanta of immediate addressing is that no memory reference other than the instruction fetch is required obtain the operand. The disadvantage is that the size of the number is restricted to the size of t address field, which, in most instruction sets, is small compared with the world length.

Examples:

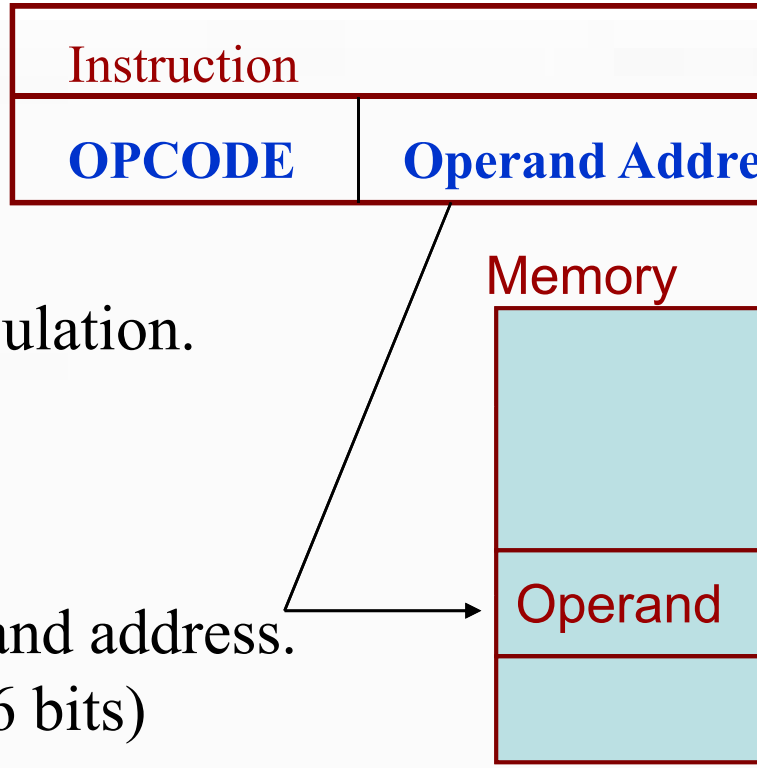
```
ADD #25              // ACC=ACC+25
ADDI R1,R2,42        // R1=R2+42
```

No memory access is required to get the operand and it is fast, but limited range because we c only specify a limited number using immediate mode like ADD #25. When we write #, it means is immediate data. So, when we write ADD #25 that means, 25 will be added with accumulator and t result will be stored back in the accumulator. Here, we have an immediate data 42, which is add to R2 and result stored in R1.

ddressing Modes

rect Addressing:

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand: EA = A



It requires only one memory reference and no special calculation.

Examples:

ADD R1, 80A6H // R1=R1+MEM[80A6]

Single memory access is required to access the operand.

No additional calculations required to determine the operand address.

Limited address space (as number of bits is limited, say,16 bits)

Instruction contains a field that holds the memory address of the operand. Here, th content of 80A6 will be the operand that we are looking for. ADD R1,80A6 means th content of 80A6 will be added with R1 and result will be stored in R1.

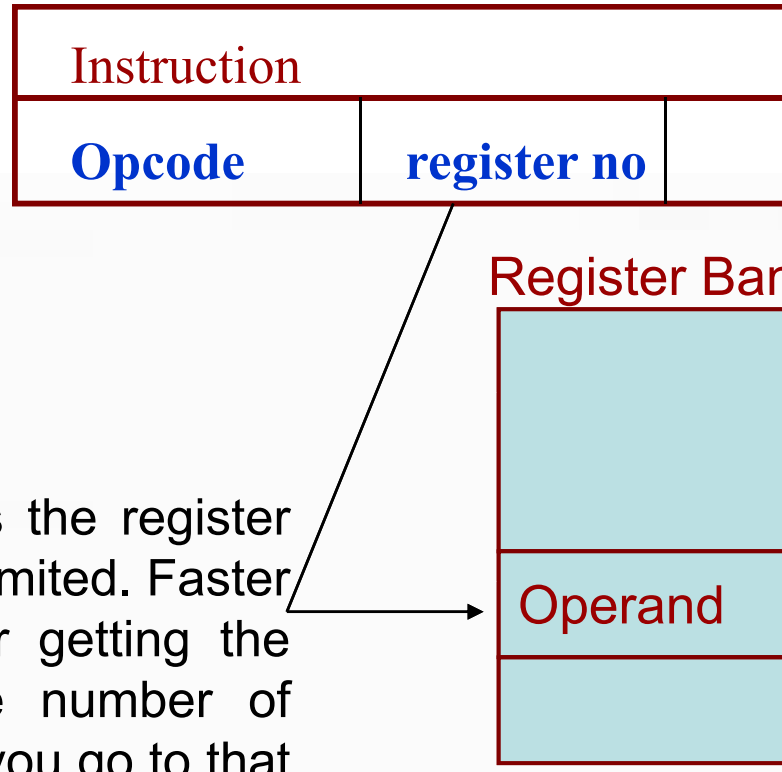
We have to go to the particular address and fetch the instruction. So, going to thi particular address, we need one memory access to access the operand, no additiona calculation is required to determine the operand address. If this address space is 16- bit it is limited. We can only have direct addressing within that 16-bit address span.

ddressing Modes

Register Addressing: Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:

EA = R

The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



The operand is held in a register, and the instruction specifies the register number. Very few bits are needed, as the number of registers is limited. Faster execution is possible, and no memory access is required for getting the operand. The modern load-store architecture supports large number of registers. The register number is specified in the instruction and you go to that particular register to get the operand.

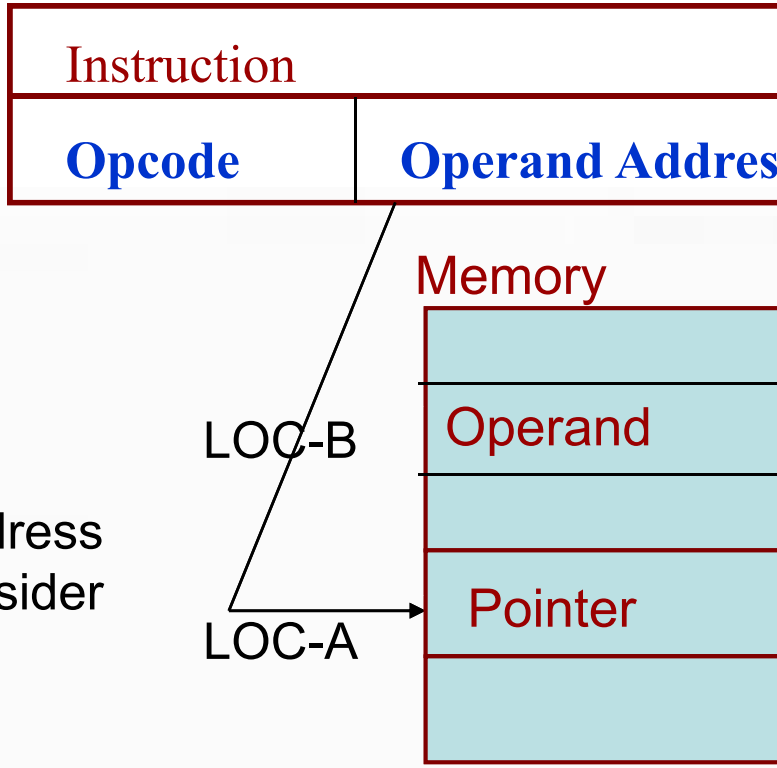
Examples:

```
ADD R1, R2, R3          // R1=R2+R3
MOV R2,R5               // R3=R5
```

ddressing Modes

Indirect Addressing: With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing:

EA = (A)



In indirect mode, it contains a field that holds the memory address which in turn holds the memory address of the operand. Consider following example.

Examples:

ADD R1,(40A6H) // R1=R1+(mem[40A6])

We have an instruction ADD R1,(LOC-A). Location (LOC-A) contains another address LOC-B. And We will not get the operand from LOCA, rather we will get operand from LOC-B. In this particular case, we have to go to the location LOC-A and LOC-A location will give another location LOC-B and this location will give the value. So, in this case, two memory accesses are required to get the operand value. This is slower, but can access larger address space. It is not limited to number of bits in the operand address like direct addressing.

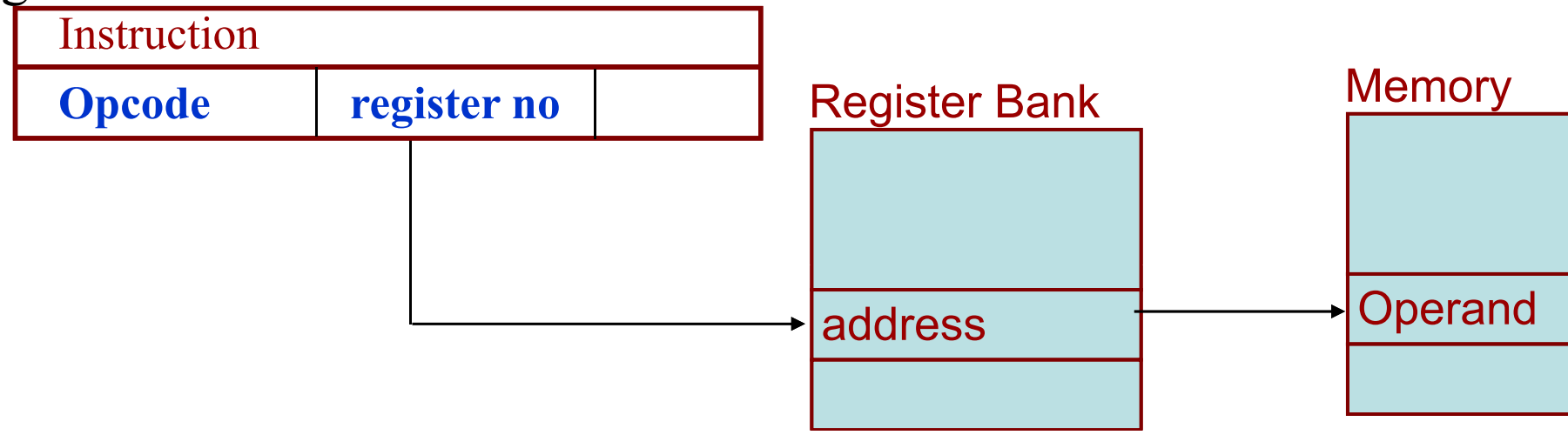
LOC-A points the address (LOC-B) the operand

ddressing Modes

Register Indirect Addressing: Register indirect addressing is similar to indirect addressin
xcept that the address field refers to a register instead of a memory location.
t requires only one memory reference and no special calculation.
EA = (R)
Register indirect addressing uses one less memory reference than indirect addressing. Because, t
irst information is available in a register which is nothing but a memory address. From th
nemory location, we use to get the data or information. In general, register access is much mo
aster than the memory access.

- Examples: ADD R1,(R5) // PC=R1+(mem[R5])

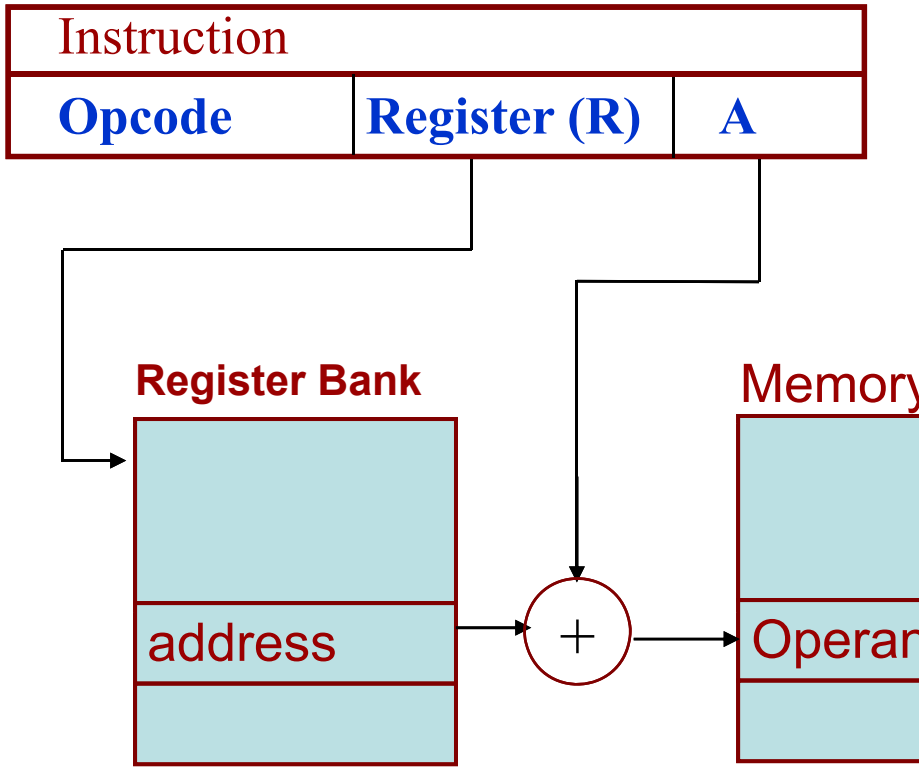
Here we are putting the memory address in a register. This register holds memory addr
that is used to access the operand. One fewer memory access is required as compared
indirect addressing mode.



Displacement Addressing:

A powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:

- EA = A + (R)



In displacement addressing, the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly.

The other address field, or an implicit reference based on opcode, refers to a registe whose contents are added to A to produce the effective address. Three of the mos common use of displacement addressing are:

- Relative addressing
- Base-register addressing
- Indexing

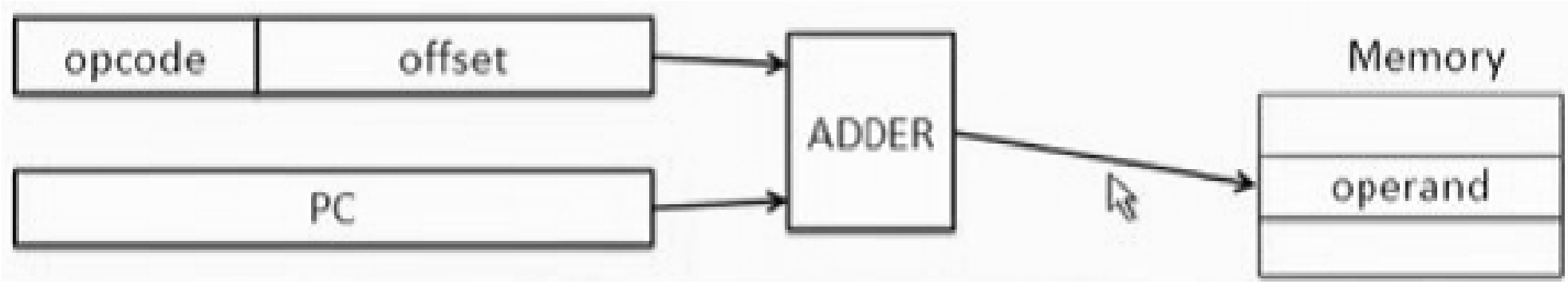
ddressing Modes

elative Addressing (PC Relative)

t is always with respect to PC. In this addressing modes the instruction specifies n offset or displacement, which is added to the program counter to get the ffective address of the operand.

ince the number of bits to specify the offset is limited, the range of relative ddressing is also limited. So, if a 12-bit offset is specified, it can have values anging from -2048 to +2047.

n branch instruction, we specify a branch address. So, this particular branch ddress will be added with the content of the PC. The offset is added with the ontent of PC and then we fetch the operand from EA.

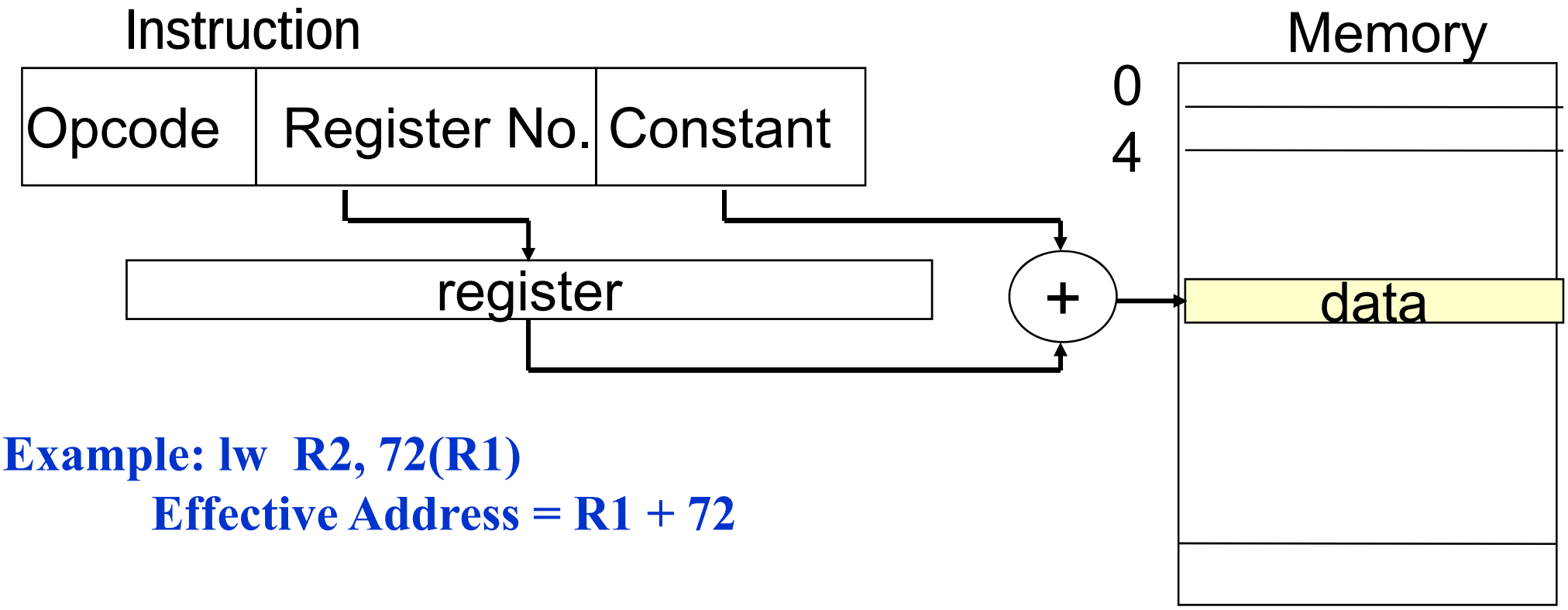


ddressing Modes

Base-Register Addressing:

The reference register contains a memory address, and the address field contains displacement from that address. The register reference may be *explicit* or *implicit*.

n some implementation, a single segment/base register is employed and is used implicitl n others, the programmer may choose a register to hold the base address of a segmer nd the instruction must reference it explicitly.



Example: lw R2, 72(R1)

Effective Address = R1 + 72

ddressing Modes

ndexed Addressing mode:

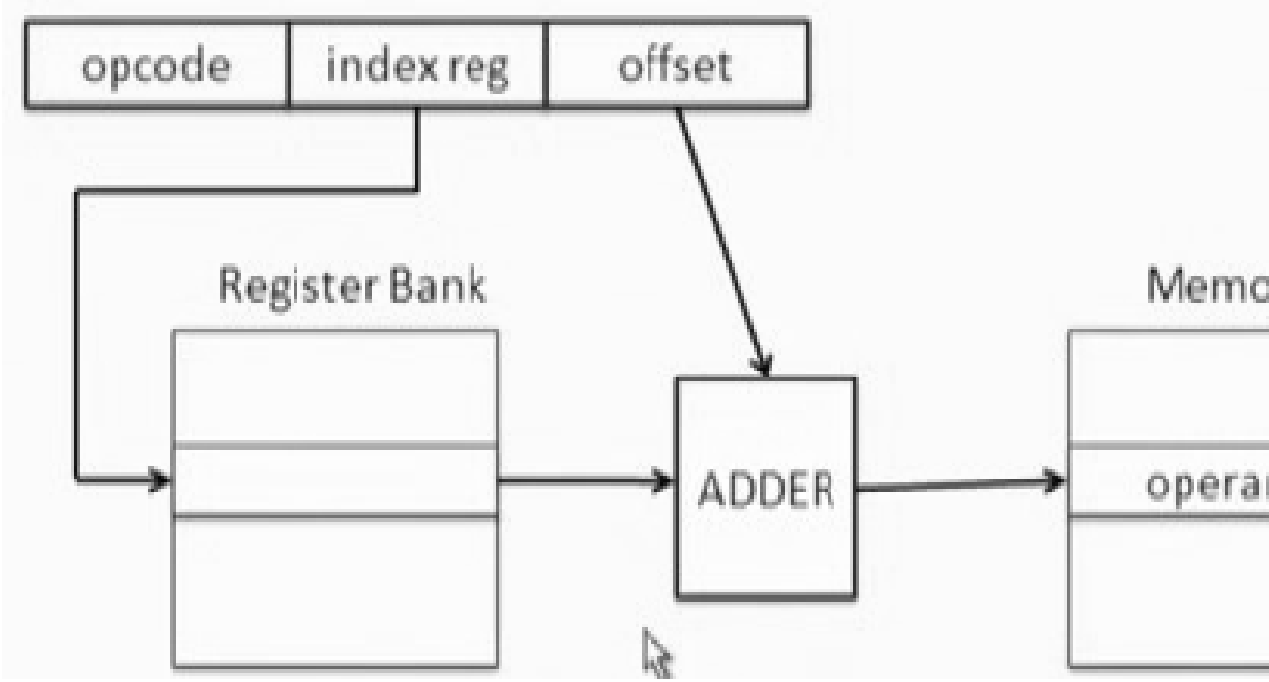
n indexed addressing mode, here either a special-purpose register or a general-purpose register sed as index register. In this addressing mode, we can access an array. Array is a set of consecuti nemory location.

n index addressing mode, the general-purpose register can be the used as index register and th nstruction specifies an offset or displacement that is added to the index register to get the effecti ddress of the operand.

Example:
OAD R1, 2050(R2) // R1=Mem (2050+R2)

ocation 2050(R2) will give the operand. 050 is added with R2. We get a value that alue from which address in memory we get he operand, and it can be used to equentially access the elements of an rray. So, we load the first address and then e move to the next address by adding an ffset to it.

ffset gives the starting address of the array nd the index register value specifies the rray element to be used; The first can be th element, then the next, then next and so



ddressing Modes

Stack Addressing:

A stack is a linear array or list of locations. It is sometimes referred to as a *pushdown list* or *last-in-first out queue*. A stack is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is partially filled.

The value of a pointer register known as stack pointer is the address of the top of the stack. The stack pointer is associated with the stack. The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

Example:

ADD, PUSH X, POP X

stack addressing, the operand is implicitly on the top of the stack and it is used in ero-address machines. The first two elements on top of the stack will be taken out, will e added, and stored back there. PUSH X will push the X value into top of the stack. OP X will take out the top value to this location and store in X. Many processors have a pecial register called a stack pointer that keeps track of the top of the stack.