DAR ES SALAAM INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER STUDIES
COU07302 MICROPROCESSOR AND COMPUTER ARCHITECTURE
Lecture 4 - Instruction Set Architecture
by
E. Kondela

Various Addressing Modes

We have examined the types of operands and operations that may be specified by machine instructions . Now we have to 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.

Addressing Modes:
The most common addressing techniques are:

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

All computer architectures provide more than one of these addressing modes . The question arises as to how the control unit can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different opcodes will use different addressing modes. Also, one or more bits in the instruction format can be used as a mode field . The value of the mode field determines which addressing mode is to be used.
What is the interpretation of effective address . In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the 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

Immediate Addressing:

The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

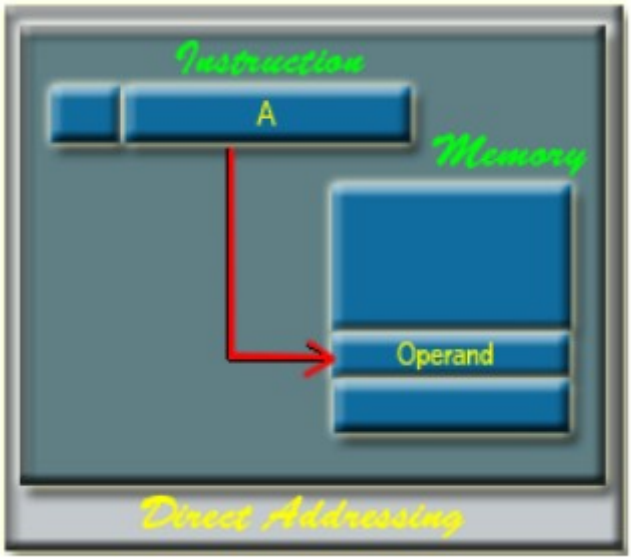
OPERAND = A

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

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

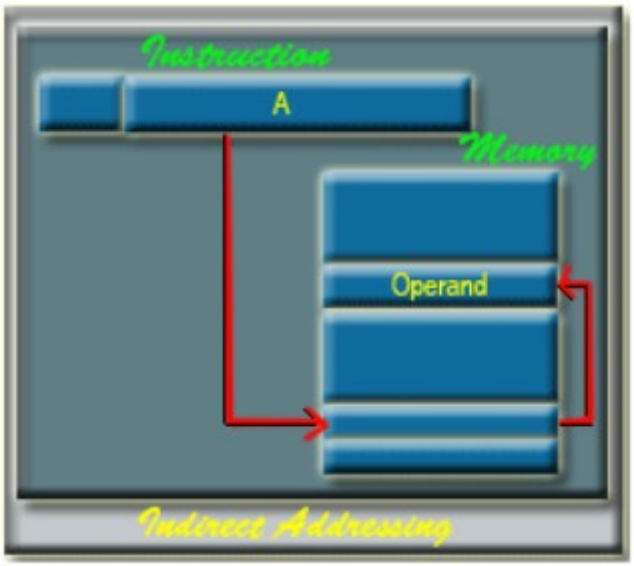
EA = A

It requires only one memory reference and no special calculation.



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 know as indirect addressing:

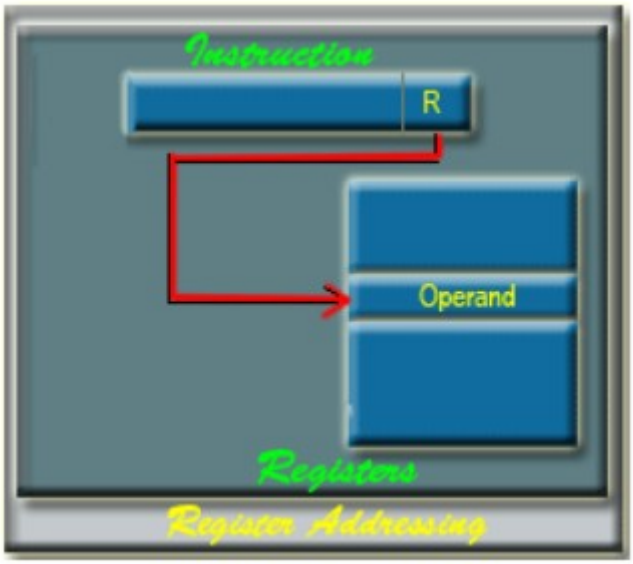
EA = (A)



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

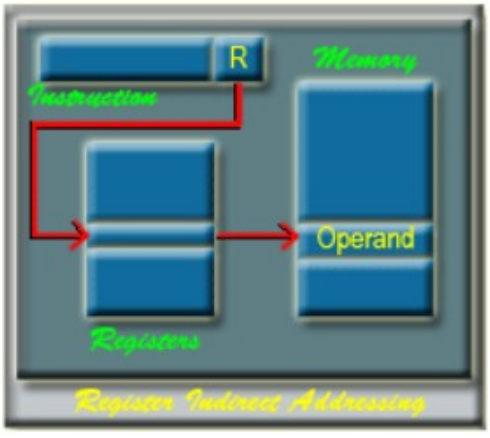
The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



Register Indirect Addressing:
Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location.
It requires only one memory reference and no special calculation.

EA = (R)

Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address. From that memory location, we use to get the data or information. In general, register access is much more faster than the memory access.

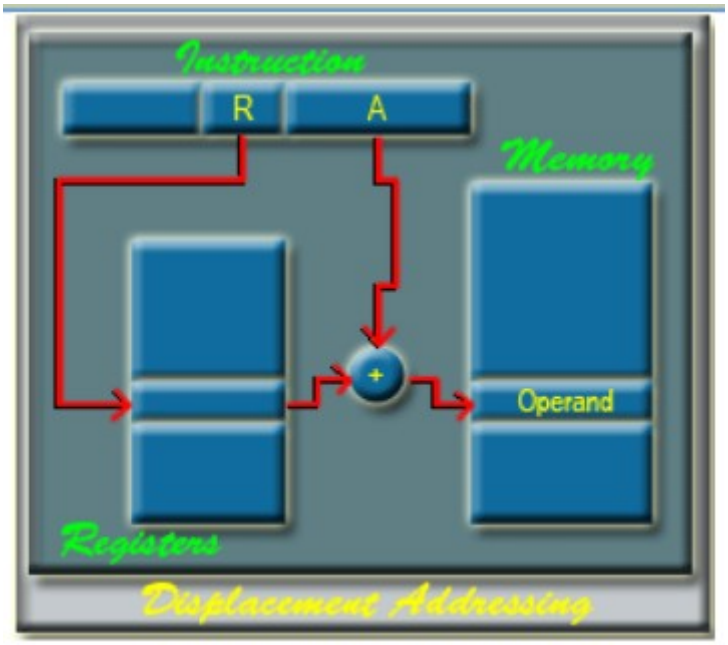


Displacement Addressing:
A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:

EA = A + (R)

Displacement addressing requires that 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 register whose contents are added to A to produce the effective address.
Three of the most common use of displacement addressing are:

- Relative addressing
- Base-register addressing
- Indexing



Relative Addressing:
For relative addressing, the implicitly referenced register is the program counter (PC). That is, the current instruction address is added to the address field to produce the EA. Thus, the effective address is a displacement relative to the address of the instruction.

Base-Register Addressing:
The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be explicit or implicit .
In some implimentation, a single segment/base register is employed and is used implicitly. In others, the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly.

Indexing:
The address field references a main memory address, and the reference register contains a positive displacement from that address. In this case also the register reference is sometimes explicit and sometimes implicit.
Generally index register are used for iterative tasks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some system will automatically do this as part of the same instruction cycle.

This is known as auto-indexing. We may get two types of auto-indexing:
- - one is auto-incrementing and the other one is
- - auto-decrementing.
If certain registers are devoted exclusively to indexing, then auto-indexing can be invoked implicitly and automatically. If general purpose register are used, the autoindex operation may need to be signaled by a bit in the instruction.

Auto-indexing using increment can be depicted as follows:
EA = A + (R)
R = (R) + 1

Auto-indexing using decrement can be depicted as follows:

EA = A + (R)
R = (R) - 1

In some machines, both indirect addressing and indexing are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection.

If indexing is performed after the indirection, it is termed postindexing

EA = (A) + (R)

First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value.
With preindexing, the indexing is performed before the indirection:

EA = (A + (R))

An address is calculated, the calculated address contains not the operand, but the address of the operand.

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. Associated with the stack is a pointer whose value is the address of the top of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses. 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.

Machine Instruction

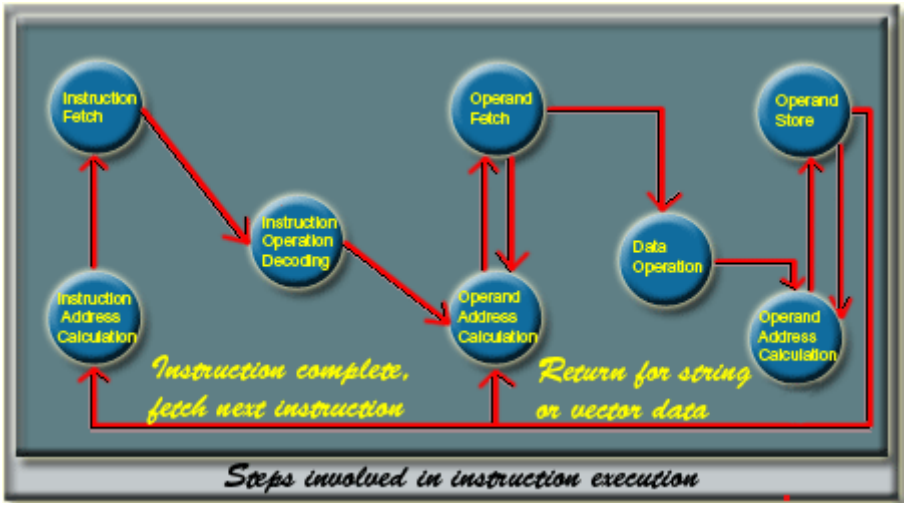
The operation of a CPU is determine by the instruction it executes, referred to as machine instructions or computer instructions. The collection of different instructions is referred as the instruction set of the CPU.
Each instruction must contain the information required by the CPU for execution. The elements of an instruction are as follows:

- Operation Code: Specifies the operation to be performed (e.g., add, move etc.). The operation is specified by a binary code, know as the operation code or opcode.
- Source operand reference: The operation may involve one or more source operands; that is, operands that are inputs for the operation.
- Result operand reference: The operation may produce a result.
- Next instruction reference: This tells the CPU where to fetch the next instruction after the execution of this instruction is complete.

The next instruction to be fetched is located in main memory. But in case of virtual memory system, it may be either in main memory or secondary memory (disk). In most cases, the next instruction to be fetched immediately follow the current instruction. In those cases, there is no explicit reference to the next instruction. When an explicit reference is needed, then the main memory or virtual memory address must be given.
Source and result operands can be in one of the three areas:

- main or virtual memory,
- CPU register or
- I/O device.

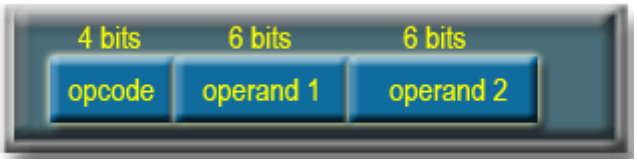
The steps involved in instruction execution is shown in the figure-



Instruction Representation
Within the computer, each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction. The instruction format is highly machine specific and it mainly depends on the machine architecture. A simple example of an instruction format is shown in the figure. It is assume that it is a 16-bit CPU. 4 bits

are used to provide the operation code. So, we may have to 16 (2 4 = 16) different set of instructions. With each instruction, there are two operands. To specify each operands, 6 bits are used. It is possible to provide 64 (2 6 = 64) different operands for each operand reference.
It is difficult to deal with binary representation of machine instructions. Thus, it has become common practice to use a symbolic representation of machine instructions. Opcodes are represented by abbreviations, called mnemonics , that indicate the operations. Common examples include:

A simple instruction format.



ADD Add
SUB Subtract
MULT Multiply
DIV Division
LOAD Load data from memory to CPU
STORE Store data to memory from CPU.

Operands are also represented symbolically. For example, the instruction
MULT R, X : R ← R x X
may mean multiply the value contained in the data location X by the contents of register R and put the result in register R
In this example, X refers to the address of a location in memory and R refers to a particular register.
Thus, it is possible to write a machine language program in symbolic form. Each symbolic opcode has a fixed binary representation, and the programmer specifies the location of each symbolic operand.

Instruction Types
The instruction set of a CPU can be categorized as follows:

- Data Processing: Arithmatic and Logic instructions Arithmatic instructions provide computational capabilities for processing numeric data. Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers. Logic instructions thus provide capabilities for processing any other type of data. There operations are performed primarily on data in CPU registers.
- Data Storage: Memory instructions Memory instructions are used for moving data between memory and CPU registers.
- Data Movement: I/O instructions I/O instructions are needed to transfer program and data into memory from storage device or input device and the results of computation back to the user.
- Control: Test and branch instructions

Test instructions are used to test the value of a data word or the status of a computation. Branch instructions are then used to branch to a different set of instructions depending on the decision made.

Number of Addresses

What is the maximum number of addresses one might need in an instruction? Most of the arithmantic and logic operations are either unary (one operand) or binary (two operands). Thus we need a maximum of two addresses to reference operands. The result of an operation must be stored, suggesting a third address. Finally after completion of an instruction, the next instruction must be fetched, and its address is needed.
This reasoning suggests that an instruction may require to contain four address references: two operands, one result, and the address of the next instruction. In practice, four address instructions are rare. Most instructions have one, two or three operands addresses, with the address of the next instruction being implicit (obtained from the program counter).

Instruction Set Design
One of the most interesting, and most analyzed, aspects of computer design is instruction set design. The instruction set defines the functions performed by the CPU. The instruction set is the programmer's means of controlling the CPU. Thus programmer requirements must be considered in designing the instruction set.
Most important and fundamental design issues:
Operation repertoire : How many and which operations to provide, and how complex operations should be.
Data Types : The various type of data upon which operations are performed.
Instruction format : Instruction length (in bits), number of addresses, size of various fields and so on.
Registers : Number of CPU registers that can be referenced by instructions and their use.
Addressing : The mode or modes by which the address of an operand is specified.

Types of Operands
Machine instructions operate on data. Data can be categorised as follows :
Addresses: It basically indicates the address of a memory location. Addresses are nothing but the unsigned integer, but treated in a special way to indicate the address of a memory location. Address arithmatic is somewhat different from normal arithmatic and it is related to machine architecture.

Numbers: All machine languages include numeric data types. Numeric data are classified into two broad categories: integer or fixed point and floating point.

Characters: A common form of data is text or character strings. Since computer works with bits, so characters are represented by a sequence of bits. The most commonly used coding scheme is ASCII (American Standard Code for Information Interchange) code.

Logical Data: Normally each word or other addressable unit (byte, halfword, and so on) is treated as a single unit of data. It is sometime useful to consider an n-bit unit as consisting of n 1-bit items of data, each item having the value 0 or 1. When data are viewed this way, they are considered to be logical data. Generally 1 is treated as true and 0 is treated as false.

Types of Opearations
The number of different opcodes and their types varies widely from machine to machine. However, some general type of operations are found in most of the machine architecture. Those operations can be categorized as follows:
- Data Transfer
- Arithmatic
- Logical
- Conversion
- Input Output (I/O)
- System Control

■ Transfer Control

Data Transfer:
The most fundamental type of machine instruction is the data transfer instruction. The data transfer instruction must specify several things. First, the location of the source and destination operands must be specified. Each location could be memory, a register, or the top of the stack. Second, the length of data to be transferred must be indicated. Third, as with all instructions with operands, the mode of addressing for each operand must be specified.
The CPU has to perform several task to accomplish a data transfer operation. If both source and destination are registers, then the CPU simply causes data to be transferred from one register to another; this is an operation internal to the CPU.
If one or both operands are in memory, then the CPU must perform some or all of the following actions:
a) Calculate the memory address, based on the addressing mode.
b) If the address refers to virtual memory, translate from virtual to actual memory address.
c) Determine whether the addressed item is in cache.
d) If not, issue a command to the memory module.

Commonly used data transfer operation:

Operation Name	Description
Move (Transfer)	Transfer word or block from source to destination
Store	Transfer word or block from source to destination
Load (fetch)	Transfer word from memory to processor
Exchange	Transfer word from memory to processor
Clear (reset)	Transfer word of 0s to destination
Set	Transfer word of 0s to destination
Push	Transfer word from source to top of stack
Pop	Transfer word from top of stack to destination

Arithmatic:
Most machines provide the basic arithmatic operations like add, subtract, multiply, divide etc. These are invariably provided for signed integer (fixed-point) numbers. They are also available for floating point number. The execution of an arithmatic operation may involve data transfer operation to provide the operands to the ALU input and to deliver the result of the ALU operation.
Commonly used operation:

Operation Name	Description
Add	Compute sum of two operands
Subtract	Compute sum of two operands
Multiply	Compute product of two operands
Divide	Compute quotient of two operands
Absolute	Compute quotient of two operands
Negate	Change sign of operand
Increment	Change sign of operand
Decrement	Subtract 1 from operand

Logical:
Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units.

Most commonly available logical operations are:

Operation Name	Description
AND	Performs the logical operation AND bitwise
OR	Performs the logical operation AND bitwise
NOT	Performs the logical operation NOT bitwise
Exclusive OR	Performs the logical operation NOT bitwise
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmatic comparison Set flag(s) based on outcome
Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control etc.
Shift	Left (right) shift operand, introducing constant at end
Rotate	Left (right) shift operation, with wraparound end

Conversion:
Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.

Input/Output:
Input/Output instructions are used to transfer data between input/output devices and memory/CPU register.
Commonly available I/O operations are:

Operation Name	Description
Input (Read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (Write)	Transfer data from specified source to I/O port or device.
Start I/O	Transfer instructions to I/O processor to initiate I/O operation.
Test I/O	Transfer status information from I/O system to specified destination

System Control:
System control instructions are those which are used for system setting and it can be used only in privileged state. Typically, these instructions are reserved for the use of operating systems. For example, a system control instruction may read or alter the content of a control register. Another instruction may be to read or modify a storage protection key.

Transfer of Control:
In most of the cases, the next instruction to be performed is the one that immediately follows the current instruction in memory. Therefore, program counter helps us to get the next instruction. But sometimes it is required to change the sequence of instruction execution and for that instruction set should provide instructions to accomplish these tasks. For these instructions, the operation performed by the CPU is to upload the program counter to contain the address of some instruction in memory. The most common transfer-of-control operations found in instruction set are: branch, skip and procedure call.

Branch Instruction
A branch instruction, also called a jump instruction, has one of its operands as the address of the next instruction to be executed. Basically there are two types of branch instructions: Conditional Branch instruction and unconditionla branch instruction. In case of unconditional branch instruction, the branch is made by updating the program counter to address specified in operand. In

case of conditional branch instruction, the branch is made only if a certain condition is met. Otherwise, the next instruction in sequence is executed.
There are two common ways of generating the condition to be tested in a conditional branch instruction First most machines provide a 1-bit or multiple-bit condition code that is set as the result of some operations. As an example, an arithmetic operation could set a 2-bit condition code with one of the following four values: zero, positive, negative and overflow. On such a machine, there could be four different conditional branch instructions:
BRP X Branch to location X if result is positive
BRN X Branch to location X if result is negative
BRZ X Branch to location X is result is zero
BRO X Branch to location X if overflow occurs
In all of these cases, the result referred to is the result of the most recent operation that set the condition code.

Another approach that can be used with three address instruction format is to perform a comparison and specify a
branch in the same instruction.
For example,
BRE R1, R2, X Branch to X if contents of R1 = Contents of R2.

Skip Instruction
Another common form of transfer-of-control instruction is the skip instruction. Generally, the skip imples that one instruction to be skipped; thus the implied address equals the address of the next instruction plus one instruction length. A typical example is the increment-and-skip-if-zero (ISZ) instruction. For example,
ISZ R1
This instruction will increment the value of the register R1. If the result of the increment is zero, then it will skip the next instruction.

Procedure Call Instruction
A procedure is a self contained computer program that is incorporated into a large program. At any point in the program the procedure may be invoked, or called. The processor is instructed to go and execute the entire procedure and then return to the point from which the call took place.
The procedure mechanism involves two basic instructions: a call instruction that branches from the present location to the procedure, and a return instruction that returns from the procedure to the place from which it was called. Both of these are forms of branching instructions.
Some important points regarding procedure call:

- A procedure can be called from more than one location.
- A procedure call can appear in a procedure. This allows the nesting of procedures to an arbitrary depth.
- Each procedure call is matched by a return in the called program.

Since we can call a procedure from a variety of points, the CPU must somehow save the return address so that the
return can take place appropriately. There are three common places for storing the return address:

- Register
- Start of procedure
- Top of stack

Consider a machine language instruction CALL X, which stands for call procedure at location X. If the register apprach is used, CALL X causes the following actions:

RN ← PC + IL
PC ← X
where RN is a register that is always used for this purpose, PC is the program counter and IL is the instruction length.
The called procedure can now save the contents of RN to be used for the later return. A second possibilities is to store the return address at the start of the procedure. In this case, CALL X causes

X ← PC + IL
PC ← X + 1

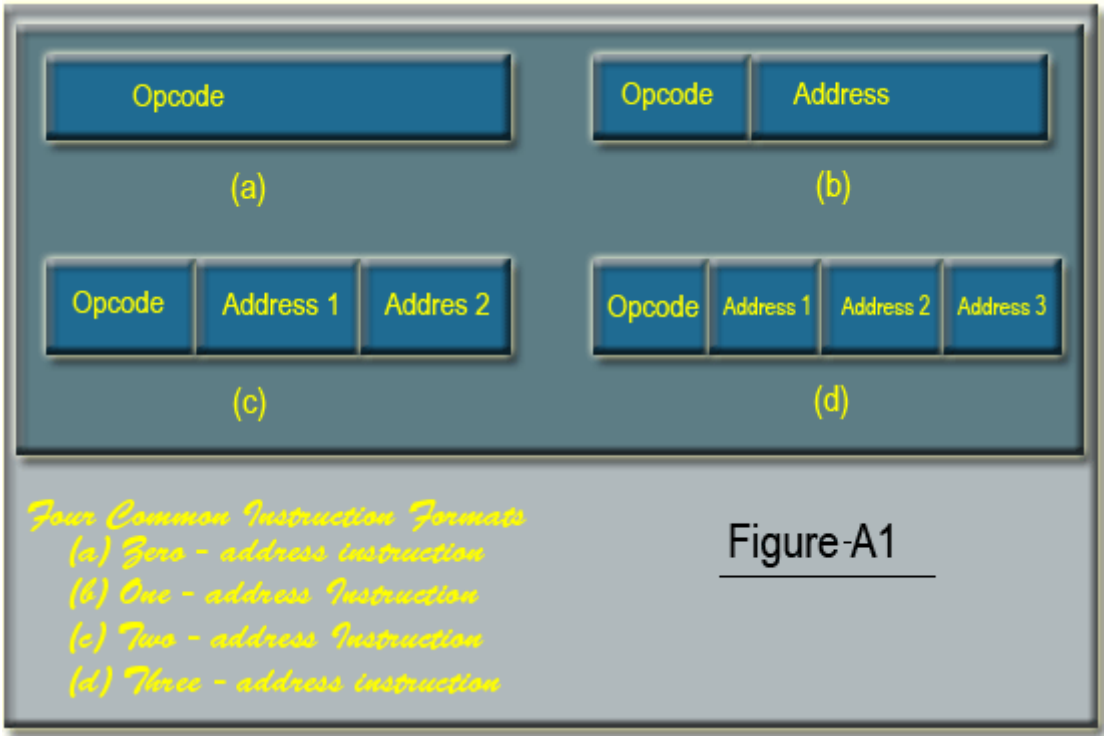
Both of these approaches have been used. The only limitation of these approaches is that they prevent the use of reentrant procedures. A reentrant procedure is one in which it is possible to have several calls open to it at the same time.

A more general approach is to use stack. When the CPU executes a call, it places the return address on the stack. When it executes a return, it uses the address on the stack.
It may happen that, the called procedure might have to use the processor registers. This will overwrite the contents of the registers and the calling environment will lose the information. So, it is necessary to preserve the contents of processor register too along with the return address. The stack is used to store the contents of processor register. On return from the procedure call, the contents of the stack will be popped out to appropriate registers.
In addition to provide a return address, it is also often necessary to pass parameters with a procedure call. The most general approach to parameter passing is the stack. When the processor executes a call, it not only stacks the return address, it stacks parameters to be passed to the called procedures. The called procedure can access the parameters from the stack.Upon return, return parameters can also be placed on the stack. The entire set of parameters, including return address, that is stored for a procedure invocation is referred to as stack frame.

Most commonly used transfer of control operation:

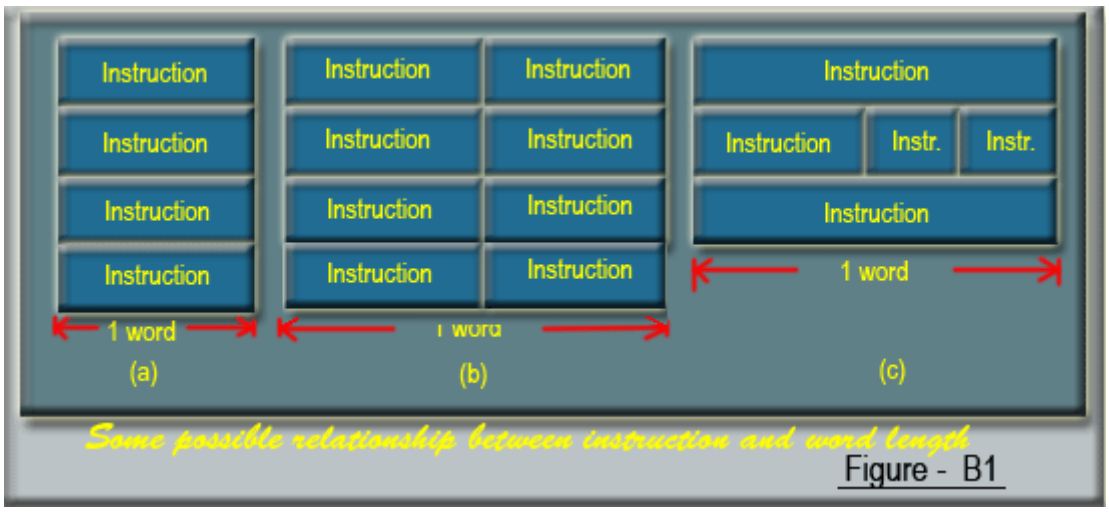
Operation Name	Description
Jump (branch)	Unconditional transfer, load PC with specific address
Jump conditional based on condition	Test specific condition; either load PC with specific address or do nothing,
Jump to subroutine	Place current program control information in known location; jump to specific address
Return	Replace contents of PC and other register from known location
Skip	Increment PC to skip next instruction
Skip Conditional	Test specified condition; either skip or do nothing based on condition
Halt	Stop program execution

Instruction Format:
An instruction format defines the layout of the bits of an instruction, in terms of its constituents parts. An instruction format must include an opcode and, implicitly or explicitly, zero or more operands. Each explit operand is referenced using one of the addressing mode that is available for that machine. The format must, implicitly or explictly, indicate the addressing mode of each operand. For most instruction sets, more than one instruction format is used. Four common instruction format are shown in the figure on the next slide .



Instruction Length:

On some machines, all instructions have the same length; on others there may be many different lengths. Instructions may be shorter than, the same length as, or more than the word length. Having all the instructions be the same length is simpler and make decoding easier but often wastes space, since all instructions then have to be as long as the longest one. Possible relationship between instruction length and word length is shown in the figure.



Generally there is a correlation between memory transfer length and the instruction length. Either the instruction length should be equal to the memory transfer length or one should be a multiple of the other. Also in most of the case there is a correlation between memory transfer length and word length of the machine.

Allocation of Bits:
For a given instruction length, there is a clearly a trade-off between the number of opcodes and the power of the addressing capabilities. More opcodes obviously mean more bits in the opcode field. For an instruction format of a given length, this reduces the number of bits available for addressing. The following interrelated factors go into determining the use of the addressing bits:

Number of Addressing modes:
Sometimes as addressing mode can be indicated implicitly. In other cases, the addressing mode must be explicit, and one or more bits will be needed.

Number of Operands:
Typical instructions on today's machines provide for two operands. Each operand address in the instruction might require its own mode indicator, or the use of a mode indicator could be limited to just one of the address field.

Register versus memory:
A machine must have registers so that data can be brought into the CPU for processing. With a single user-visible register (usually called the accumulator), one operand address is implicit and consumes no instruction bits. Even with multiple registers, only a few bits are needed to specify the register. The more that registers can be used for operand references, the fewer bits are needed.

Number of register sets:
A number of machines have one set of general purpose registers, with typically 8 or 16 registers in the set. These registers can be used to store data and can be used to store addresses for displacement addressing. The trend recently has been away from one bank of general purpose registers and toward a collection of two or more specialized sets (such as data and displacement).

Address range:
For addresses that reference memory, the range of addresses that can be referenced is related to the number of address bits. With displacement addressing, the range is opened up to the length of the address register.

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

Variable-Length Instructions:
Instead of looking for fixed length instruction format, designer may choose to provide a variety of instructions formats of different lengths. This tectic makes it easy to provide a large repertoire of opcodes, with different opcode lengths. Addressing can be more flexible, with various combinations of register and memory references plus addressing modes. With variable length instructions, many variations can be provided efficiently and compactly. The principal price to pay for variable length instructions is an increase in the complexity of the CPU.

Number of addresses :
The processor architecture is described in terms of the number of addresses contained in each instruction. Most of the arithmatic and logic instructions will require more operands. All arithmatic and logic operations are either unary (one source operand, e.g. NOT) or binary (two source operands, e.g. ADD).

Thus, we need a maximum of two addresses to reference source operands. The result of an operation must be stored, suggesting a third reference.
Three address instruction formats are not common because they require a relatively long instruction format to hold the three address reference.
With two address instructions, and for binary operations, one address must do double duty as both an operand and a result.
In one address instruction format, a second address must be implicit for a binary operation. For implicit reference, a processor register is used and it is termed as accumulator(AC). the accumulator contains one of the operands and is used to store the result.

Consider a simple arithmatic expression to evalute:
Y= (A + B) / (C * D)