INTRODUCTION

Computer Architecture

- Refers to those attributes of a system visible to a programmer.
- In other words, refers to those attributes that have direct impact on logical execution of program.
- Example: Instruction set, data types, I/O mechanism, addressing modes, cache optimization.
- Says what to do?(instruction set)

Computer Organization

- Also referred microarchitecture(low level).
- Refers to the operational unit and their interconnection that realizes the architecture specification.
- Includes hardware details such as control signal, interface between computer & peripherals, memory technology used, address, etc.
- Says how to do? (implementation of architecture)

Example

- If a computer is running multiple instruction (what to do?), it is an architectural design issue.
- It is an organizational issue whether that instruction will be implemented by a special multiple unit or a unit that makes repeated use of add instruction. (how to do?)

Levels of Programming Languages:

- High Level Programming Language
- Assembly Level Language
- Machine Level Language

High Level Language

- Similar to natural languages.
- Easier for programmer to read and write.
- Platform independent.
- Should be converted to machine level language before execution; execution time is a bit longer.
- Example: C, C++, java.

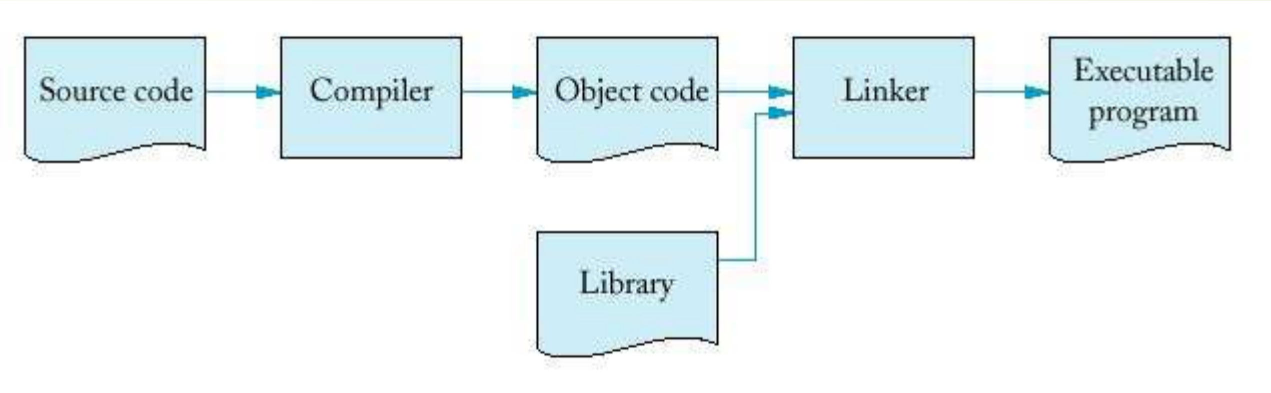
Assembly Level Language

- Makes use of mnemonics instead of numeric generation codes.
- Platform dependent.
- Assembler converts into machine language.
- Slower than machine language.

Machine Level Language

- Have the lowest level of abstraction.
- Also called binary language because they contain only 0s and 1s.
- Difficult to read write and debug
- Faster execution time.

Compiling and assembling program



(contd.)

- A program written in high level language is input to compiler. The compiler makes sure that every statement in program is valid.
- When a program has no errors, the compiling of the program will be finished i.e source code generates object code.
- At this point, the program has been compiled successfully, but it is not yet ready to be executed.
- Some programs use the object code of other program in addition to their own.
- Linker combines your object code with any other required object code.
- This combined code is stored as an executable file. It is actually the code that the computer run.
- The loader copies the executable file into memory, the microprocessor then runs the machine code contained in that file.

Contd….

- Each assembly language instruction corresponds to one unique machine code instruction.
- The assembler, like the compiler converts its source code to object code.
- From there, it follows the linking and loading procedure that was used for compiled code.

Instruction types:
(Based on types of operation)

- Data Transfer instruction
  =>do not modify the data; only copy values to destination.
  Example:
  1. load data from memory to microprocessor.
  2. store data from microprocessor into memory.
  3. move data within the microprocessor.
  4. input data to the microprocessor.
  5. output data from the microprocessor.

Contd..

- Data Operation Instruction

  =>modifies the value.
  =>performs some operations using one or two operands.
  Example:
  =>Arithmetic operations like add, subtract, multiply, increment, decrement
  =>Logical operations like AND, OR, XOR
  =>Shift instructions.

Contd…

- Program Control Instruction:
  =>instruction that determines the flow of execution of programs.
  Example:
  =>looping instruction like for, while, do…while
  =>CALL and RETURN statements(used by Assembly language) to run some micro-routine.
  =>Interrupt
  =>HLT, causes the microprocessor to stop executing instruction such as the end of program.

Contd..
(Based on address field)

- 3-address instruction
  =>each instruction specifies two operand location and one result location.
  Syntax:
  opcode x,y,z
  Example:
  x=y+z

Contd..

- 2-address instruction
  =>each instruction specifies either one operand and result or two operand location, one of which is also used as result location.
  Syntax:
  opcode X,Y
  Example:
  MOV A,B
  ADD R,B

Contd..

- 1-address instruction
  => uses implicit accumulator register for all data manipulation.
  Syntax:
  opcode X
  Example:
  ADD B implies A<-A+B

Contd…

- 0-address instruction
  =>does not use address field for the instructions like ADD,SUB,MUL, etc.
  =>the PUSH and POP, however need an address field to specify the operand that communicate the stack.
  =>the name 0-address is given because of the absence of address field in the computational instruction.
  Example:
  PUSH A //Top of Stack <- A
  PUSH B //Top of Stack <- B
  ADD //Top of Stack <- (A + B)

Instruction Set Architecture Design

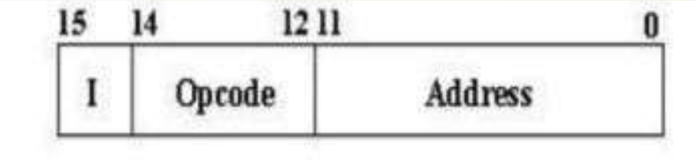
- Very important in the design of the microprocessor.
- Poorly designed ISA, if implemented well leads to bad microprocessor design.
- Well designed ISA leads to powerful microprocessor.
- No magical formula for designing ISA; same requirement can have different ISA design, each of which can be valid.
- Must evaluate trade-off's between performance, size and cost.
- Before beginning the design of ISA, it is to be decided what should ISA and its processor be able to do.

Contd..

- If processor is to be used for general purpose computing, requires relatively large set of instruction to perform variety of tasks; for specialized processor, the task of microprocessor is very little and well known in advance.
- Orthogonality: Instructions are orthogonal if they do not overlap or perform the same function. Good ISA design should minimize the overlapping.
- Some other issues that must be dealt while designing ISA are:
  => optimization of register set.
  =>type and size of data that microprocessor uses.
  =>Are conditional instruction needed?
  =>Are interrupts needed?

Instruction Format

- The bits of instructions are divided into groups called fields. The most common field in instruction format are:
  => an operation code field that specifies the operations to be performed.
  => the address field that designates an address or a processor register.
  =>the mode field that specifies the way operands or effective address is determined.



- In an instruction format:
  => First 12 bits(0-11), specify an address.
  =>Next 3 bit specify operation code (opcode).
  =>Leftmost bit specify the addressing mode I
  I = 0 for direct address.
  I = 1 for indirect address.

Addressing modes:

The way in which operands are specified in an instruction.

- Direct mode:
  => contains memory addresses that CPU accesses.
  Eg: LDAC 5, read data from memory location 5 and stores in accumulator.
- Indirect mode:
  => The address in the instruction is not the address of operand, rather it contains the address of memory location where operand resides.
  Eg: LDAC @5 or LDAC (5), first retrieves the content of memory location 5 , say 10. The CPU then retrieves the content of memory location 10, that actually contains the operand.
- Register direct and register indirect modes:
  => works same as direct and indirect modes, except they do not specify memory address, instead they specify a register.
  Eg; LDAC (R) or LDAC @R
  LDAC R

Contd..

- Immediate mode:
  =>operand specified is not an address, it is actual data to be used.
  Eg: LDAC #5 moves the data value 5 into accumulator.
- Implicit mode:
  =>doesn't explicitly specify an operand, instruction always refers to accumulator.
  Eg: CLAC, which clears the accumulator.
- Relative mode:
  =>operand specified is an offset, not the actual address which is added to the content of program counter register to generate required address.
  Eg: LDAC $5
  =>if next instruction is at location 12 i.e. PC has value 12, the instruction reads data from 12 +5 = 17 location and stores in accumulator.

Contd..

- Index mode and base address mode:
  =>Index mode works like relative mode, except the address supplied by instruction is added to the content of index register.
  Eg: LDAC 5(X), reads from 10 + 5 = 15(index register contains value 10)

  =>Base address mode is same as index mode except index register is replaced by base register.