

ISA - Instruction Set Architecture (I)

Overview

- An **Instruction Set Architecture (ISA)** defines the **interface between software and hardware**. Software is converted to machine instructions using software (compiler/interpreter). Then the instructions are executed using hardware.
 - An **ISA contains**
 - the functional **definition of storage locations** (registers, memory) & **operations** (add, multiply, branch, load, store, etc)
 - precise **description of how to invoke & access** them
 - ISA does not contain: non-functional aspects
 - How operations are implemented
 - Which operations are fast and which are slow
 - Which operations take more power and which take less
 - **Instructions** are **bit-patterns**. Hardware interprets as **commands**.
-

Basic ISA classes

- We know that hardware decides the instruction format. Internal storage space in a processor is a factor impacting an ISA.
- Instructions are to execute operations that will take one or more operands.
- Based on where the operands are stored and whether they are named explicitly or implicitly, the ISA can be classified as Stack Architecture, Accumulator Architecture, and Register-set Architecture.

Stack Architecture

- The operands are put into the stack. Operations take place at the top two locations on the stack, destroying the operands, and leaving the result on the top of the stack.
- Stack is a block of memory in RAM, but CPU keeps a **stack pointer register** points to the top of the stack. Data in stack are *first-in last-out*.
- Two operations, **push and pop, have one operand**. Push a value at the top of the stack, or pop the top element of the stack to a destination. **Other operations have implicit operands, which are the top element(s) of the stack**. To perform an operation, the operands are pushed into the stack first. Once performed an operation, its operand(s) are deleted from the stack, and its result is stored at the top of the stack. Then, it can use pop command to store the result back to the memory.
- This architecture was used in computers in 1960s.

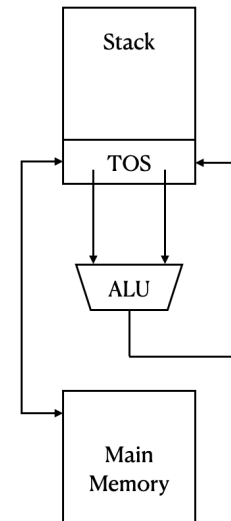
- Suppose we want to ask a stack architecture machine to conduct the operation $C = A + B$, where A, B, and C are memory addresses. The result of $A + B$ are stored at location C.
- Then, to implement the operation in a stack architecture, we need the following steps:

PUSH A # push value at location A to the top of the stack

PUSH B # push value at location B to the top of the stack

ADD # add the top two elements of the stack and save the result back to the top of the stack

POP C # store the value at the top of the stack (which is the result) to location C



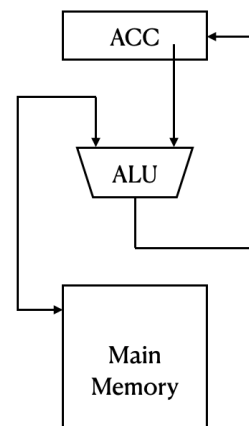
Accumulator Architecture

- It places **one operand in the accumulator and one in memory**. The **one in the accumulator is implicit, and the one in the memory is an explicit memory address**. The **result of ALU is written back to the accumulator**.
- The operand in the accumulator is loaded from memory using the command LOAD, and the result is stored in memory from accumulator using the command STORE.
- This architecture was used by early computers like IBM 7090. Today, it's used by some microprocessor chips, such as some digital signal processors.
- Suppose we want to ask an accumulator architecture machine to conduct the same operation $C = A + B$ as above.
- Then, to implement the operation in an accumulator architecture, we need the following steps:

LOAD A # load value at location A from main memory to ACC

ADD B # fetch the value at location B, add it with the value at ACC (which is the value A), and save the result back to ACC

STORE C # store the value at ACC (which is the result of $A + B$) to location C



Register-set Architecture

- It's the **dominant architecture** used by modern computers.
- Modern CPUs have a number of general-purpose registers (GPRs) for internal storage, at least 8 and as many as 32.