CS2507 Computer Architecture

Lecture 2

The Instruction Set

Instruction Set Architecture (ISA) Level

The purpose of a processor is to operate on data: get data from memory, process it and do something with the results.

The fundamental model is the Stored-Program Computer. A list of instructions is stored in memory. The processor, or Central Processing Unit (CPU), fetches an instruction, executes it and continues to the "next" instruction. In most cases, this pattern is repeated to infinity. The repeated pattern is known as the Instruction Cycle.

Each instruction either processes data or manages the flow of instructions.

To process the data there is an Arithmetic and Logic Unit (ALU).

The ALU determines what data types are recognized directly: bit strings, characters, binary coded decimal (BCD), natural numbers, (signed) integers, floating point numbers.

The ALU determines what can be done directly with data: addition, subtraction, multiplication, division, shifting, rotating, complementing, logical AND, logical OR, and what data types can be operated on.

At ISA level the ALU is visible through the instructions that are provided.

Since many of the operations carried out by the ALU naturally have two operands, there are two inputs to the ALU. Data needs to be held temporarily at these inputs, and this is done by holding the data in special storage units called registers. Many traditional architectures recognize one of these input registers as the "principal" data register and name it the Accumulator (ACC or A-register), for historical reasons. Data produced at the output of the ALU is also held in a register. The number of bits in each of these registers is determined by the number of bits that can be processed in parallel by the ALU. This number of bits is known as the CPU word length and is commonly some power of 2, such as 16, 32 or 64.

Accessing instruction operands is a major concern. If the operands are already within the processor, they can be accessed much faster than if they are in memory. Therefore, most architectures provide additional registers within the CPU for the temporary storage of operands. There are normally relatively few of these registers, so the number of bits required to distinguish between them, the register address, is much less than the number of bits in a memory address. The registers are usually named individually for programming purposes.

As well as holding operands in registers, many architectures support a range of operand addressing modes that generally involve some degree of indirection. Indirection occurs when an operand is identified by placing its memory address in a register or another memory location and identifying the location of the operand address within the instruction. It gives the ability to modify the operand address from within a running program. Architectures that support this concept may contain registers that can be used to hold operand addresses.

Instructions are fetched from a memory unit known as main memory or primary storage. The memory address of the next instruction is held in a special CPU register, known generically as the Program Counter (PC). The flow of instructions is managed by manipulating the contents of the PC using operations such as Branch (Jump), Call and Return.

There are several ways to implement subroutine call and return, but reentrancy is generally achieved by using a stack, which is generally off-chip in main memory. The stack mechanism is supported by a Stack Pointer (SP) register within the CPU, as well as Push and Pop operations.

Some architectures recognize that a CPU will have to communicate with its environment. The designers consider how to accomplish this by thinking of how to identify input and output devices. There are two main ways of dealing with I/O: memory-mapped I/O or the use of a separate I/O address space. If I/O is only to be memory-mapped, this means that a range of main memory addresses is devoted to I/O devices, which are then accessed at the ISA level by ordinary data movement instructions. If a separate I/O address space is used, this is implemented at ISA level by having distinct Input and Output instructions, together with the concept of an input/output address that is separate to (and usually shorter than) the main memory address. If a processor architecture does include a separate I/O address space, this does not prevent a particular computer design that employs that processor from using memory-mapped I/O.

Finally, there is often a need to control the operation of the CPU. This may occur if one wishes to halt the instruction cycle, for instance, or to set up conditions that govern the operation of particular instructions. Thus, most ISA-level instruction sets include a Halt instruction, a No Operation instruction and instructions to set or clear particular condition-creating bits known as flags.

To summarise, our Instruction Set Architecture relates to a Central Processing Unit that comprises an Arithmetic and Logic Unit, an Accumulator register, additional general-purpose registers, operand addressing registers, a Program Counter register, and a Stack Pointer register. The CPU executes instructions that it fetches from a main memory unit. This memory unit contains a large number of locations, each identified by a unique memory address. The number of bits in a memory address is explicitly recognized by the CPU in the width of the PC, SP and other internal registers and in the operand address generation mechanism.

Instructions available at ISA level can be classed into the following main groups: Data Movement, Arithmetic and Logic, Flow of Control (Branch, Call and Return), Input/Output and Machine Control.

There are other instruction types that have not yet been mentioned here, but at the moment we are just establishing a base from which to proceed with our consideration of the ISA level.

Example Instruction Set

Look at the Intel 8085 datasheet. View the different instruction groupings.