İstanbul
Bilgi Üniversitesi

CMPE321 - Computer Architecture

Lecture 8
Instruction Set Architecture

Hakan Ayral, PhD.

Instruction Set Architecture

- In this chapter, we will study material dealing with instruction set architecture for general-purpose computers.
- We will examine the operations that the instructions perform and focus particularly on how the operands are obtained and where the results are stored.
- We will contrast two distinct classes of architectures:
  - reduced instruction set computers (RISCs) and
  - complex instruction set computers (CISCs).
- We will classify elementary instructions into three categories:
  - data transfer,
  - data manipulation,
  - and program control.
- In each of these categories, we elaborate on typical elementary instructions.

Instruction Set Architecture

- Central to the material presented here are the general-purpose parts of the generic computer, including the central processing unit (CPU) and the accompanying floating-point unit (FPU).
- Since a small general-purpose microprocessor may be present for controlling keyboard and monitor functions, these components are also involved.
- Aside from addressing used to access memory and I/O components, the concepts studied apply less to other areas of the computer.
- Increasingly, however, small CPUs have appeared more frequently in the I/O components.

Computer Architecture Concepts

- The binary language in which instructions are defined and stored in memory is referred to as ***machine language***.
- A symbolic language that replaces binary opcodes and addresses with symbolic names and that provides other features helpful to the programmer is referred to as ***assembly language***.
- The logical structure of computers is normally described in assembly-language reference manuals.
- Such manuals explain various internal elements of the computer that are of interest to the programmer, such as processor registers.
- The manuals list all hardware-implemented instructions, specify the symbolic names and binary code format of the instructions, and provide a precise definition of each instruction.
- In the past, this information represented the ***architecture*** of the computer.
- A computer was composed of its architecture, plus a specific ***implementation*** of that architecture.
- The **implementation** was separated into two parts: the **organization** and the **hardware**.
- The ***organization*** consists of structures such as datapaths, control units, memories, and the buses that interconnect them.
- ***Hardware*** refers to the logic, the electronic technologies employed, and the various physical design aspects of the computer.

Computer Architecture Concepts

- As computer designers pushed for higher and higher performance, and as increasingly more of the computer resided within a single IC, the relationships among architecture, organization, and hardware became so intertwined that a more integrated viewpoint became necessary.
- According to this new viewpoint, architecture as previously defined is more restrictively called *instruction set architecture* (*ISA*), and the term ***architecture*** is used to encompass the whole of the computer, including instruction set architecture, organization, and hardware.
- This unified view enables intelligent design trade-offs to be made that are apparent only in a tightly coupled design process.
- These trade-offs have the potential for producing better computer designs.
- In this chapter, we focus on instruction set architecture.

Computer Architecture Concepts

- A computer usually has a variety of instructions and multiple instruction formats.
- It is the function of the control unit to decode each instruction and provide the control signals needed to process it.
- We now introduce typical instructions found in commercial general-purpose computers.
- We also investigate the various instruction formats that may be encountered in a typical computer, with an emphasis on the addressing of operands.
- The format of an instruction is depicted in a rectangular box symbolizing the bits of the binary instruction. The bits are divided into groups called ***fields***.
- The following are typical fields found in instruction formats:
    1. An ***opcode field***, which specifies the operation to be performed.
    2. An ***address field***, which provides either a memory address or an address that selects a processor register.
    3. A ***mode field***, which specifies the way the address field is to be interpreted.
- Other special fields are sometimes employed under certain circumstances—for example, a field that gives the number of positions to shift in a shift-type instruction or an operand field in an immediate operand instruction.

Basic Computer Operation Cycle

- In order to comprehend the various addressing concepts to be presented in the next two sections, we need to understand the basic operation cycle of the computer.
- The computer’s control unit is designed to execute each instruction of a program in the following sequence of steps:
    1. Fetch the instruction from memory into a control register.
    2. Decode the instruction.
    3. Locate the operands used by the instruction.
    4. Fetch operands from memory (if necessary).
    5. Execute the operation in processor registers.
    6. Store the results in the proper place.
    7. Go back to step 1 to fetch the next instruction.

Basic Computer Operation Cycle

- A register in the computer called the ***program counter*** (***PC***) keeps track of the instructions in the program stored in memory.
- The *PC* holds the address of the instruction to be executed next and is incremented each time a word is read from the program in memory.
- The decoding done in Step 2 determines the operation to be performed and the addressing mode or modes of the instruction.
- The operands in Step 3 are located from the addressing modes and the address fields of the instruction.
- The computer executes the instruction, storing the result, and returns to Step 1 to fetch the next instruction in sequence.

Register Set

- The ***register set*** consists of all registers in the CPU that are accessible to the programmer.
- These registers are typically those mentioned in assembly-language programming reference manuals.
- In the simple CPUs we have dealt with so far, the register set has consisted of the programmer-accessible portion of the register file and the *PC*.
- The CPUs can also contain other registers, such as the instruction register, registers in the register file that are accessible only to hardware controls and/ or microprograms, and pipeline registers.
- These registers, however, are not directly accessible to the programmer and, as a consequence, are not a part of the register set, which represents the stored information in the CPU that instructions are defined to access.
- Thus, the register set has a considerable influence on instruction set architecture.

Register Set

- The register set for a realistic CPU is quite complex.
- In this chapter, we add two registers to the set we have used thus far:
  - the ***processor status register*** (***PSR***) and
  - The ***stack pointer*** (***SP***).
- The processor status register contains flip-flops that are selectively set by status values *C*, *N*, *V*, and *Z* from the ALU and shifter.
- These stored status bits are used to make decisions that determine the program flow, based on ALU results, shifter results, or the contents of registers.
- The stored status bits in the processor status register are also referred to as the ***condition codes*** or the ***flags***.
- Additional bits in the *PSR* will be discussed when we cover associated concepts in this chapter.

Operand Addressing

- Consider an instruction such as ADD, which specifies the addition of two operands to produce a result.
- Suppose that the result of the addition is treated as just another operand.
- Then the ADD instruction has three operands: the addend, the augend, and the result.
- An operand residing in memory is specified by its address.
- An operand residing in a processor register is specified by a register address, a binary code of *n* bits that specifies one of at most 2^*n* registers in the register file.
- Thus, a computer with 16 processor registers, say, *R*0 through *R*15, has in its instructions one or more register address fields of four bits.
- The binary code 0101, for example, designates register *R*5.

Operand Addressing

- Some operands, however, are not explicitly addressed, because their location is specified either by the opcode of the instruction or by an address assigned to one of the other operands.
- In such a case, we say that the operand has an ***implied address***.
- If the address is implied, then there is no need for a memory or register address field for the operand in the instruction.
- On the other hand, if an operand has an address in the instruction, then we say that the operand is explicitly addressed or has an ***explicit address***.
- The number of operands explicitly addressed for a data-manipulation operation such as ADD is an important factor in defining the instruction set architecture for a computer.
- An additional factor is the number of such operands that can be explicitly addressed in memory by the instruction.
- These two factors are so important in defining the nature of instructions that they act a means of distinguishing different instruction set architectures.
- They also govern the length of computer instructions.

Operand Addressing

- We begin by illustrating simple programs with different numbers of explicitly addressed operands per instruction.
- Since the explicitly addressed operands have up to three memory or register addresses per instruction, we label the instructions as having three, two, one, or zero addresses.
- Note that, of the three operands needed for an instruction such as **ADD**, the addresses of all operands not having an address in the instruction are implied.
- To illustrate the influence of the number of operands on computer programs, we will evaluate the arithmetic statement $X = (A + B)(C + D)$ using three, two, one, and zero address instructions.
- We assume that the operands are in memory addresses symbolized by the letters *A*, *B*, *C*, and *D* and must not be changed by the program.
- The result is to be stored in memory at a location with address *X*.
- The initial arithmetic operations to be used in the instructions are addition, subtraction, and multiplication, with mnemonics **ADD**, **SUB**, and **MUL**, respectively.
- Further, three operations needed to transfer data during the evaluation are move, load, and store, denoted by **MOVE**, **LD**, and **ST**, respectively.
- **LD** moves an operand from memory to a register and **ST** from a register to memory.
- Depending on the addresses permitted, **MOVE** can transfer data between registers, between memory locations, or from memory to register or register to memory.

Three-Address Instructions

- A program that evaluates $X = (A + B)(C + D)$ using three-address instructions is as follows (a register transfer statement is shown for each instruction):

ADD T1, A, B	$M[T1] \leftarrow M[A] + M[B]$
ADD T2, C, D	$M[T2] \leftarrow M[C] + M[D]$
MUL X, T1, T2	$M[X] \leftarrow M[T1] \times M[T2]$

- The symbol *M*[*A*] denotes the operand stored in memory at the address symbolized by A.
- The symbol × designates multiplication.
- T1 and T2 are temporary storage locations in memory.