Register-Memory
- One operand is in a register, and one is in memory (a memory address). The result of ALU stores back in the register.
- This architecture is different from the accumulator architecture in a way that it has multiple registers used to store operands and results.
- This architecture is used by IBM 360/370.

Memory-Memory
- All operands are from memory.
- Most CISCs (complex instruction set computers) use this architecture.

CISC vs. RISC
- We’ve mentioned CISC and RISC above when talking about the classification of ISA. So, what are CISC and RISC?
- **CISC** is short for **Complex Instruction Set Architecture**, and **RISC** is short for **Reduced Set Instruction Set Architecture**. You can tell by their names that they are two types of machines that use different instruction set architectures. CISC uses memory-memory architecture, and RISC uses register-register architecture.
- Let’s use an example to illustrate CISC and RISC.

The above diagram represents the storage scheme for a generic computer. It represents a computer with a two-level memory system that doesn’t have caches. GPRs (general-purpose
registers) are the memory that operations can directly access in most processors, unlike cache and main memory that are pools of data.

- Main memory has 9 words addressed by decimal numbers from 0 to 8.
- ALU is responsible for all computations but can only operate data that have been loaded into the six GPRs: A, B, C, D, E, or F.
- Suppose we want to let the computer conduct an operation that calculate the product of two numbers, one stored at the location 2 and the other stored at the location 5 in the main memory, and store the product back at the location 2.

**CISC**

- If the computer is a CISC machine, the CISC processor will prepare a specific instruction for multiplication, assuming it’s named **MUL**. When executed, this instruction loads the two numbers from the main memory into two separate registers, multiples the operands in the ALU, stores the product in an appropriate register, and then stores the product back to the main memory. The entire process of multiplying two numbers can be completed with only one instruction: `MUL 2 2 5`.

- Instructions like **MUL** are known as “complex instructions.” They are operates directly on the main memory and do not require the programmers to explicitly call any “load” or “store” functions to load the numbers from the main memory to GPRs or store the computing results back to the main memory.

- The primary goal of CISC architecture is to complete a task in as few lines of instructions as possible and build complex instructions directly into the hardware.
- So, the processor hardware of CISC machines is capable of understanding a complex instruction and executing a series of operations behind it.

- The primary advantages of CSIS machines are:
  - The compilers have to do very little work to translate a high-level language program into assembly program, as the complex instructions are already close to the high-level languages.
  - Because of the length of an assembly program is relatively short, very little RAM is required to store instructions.
RISC

- If the computer is a RISC machine, you will have to break the above instruction \texttt{MUL 2 2 5} into four instructions to finish the product operation, such as:
  \begin{verbatim}
  LOAD A 2 # load the number at the location 2 to register A
  LOAD B 5 # load the number at the location 5 to register B
  PROD C A B # conduct A \times B and save the result back to C
  STORE 2 C # store the value at C back at the location 2
  \end{verbatim}

- The idea of RISC architecture is to make hardware simpler by using “simple instructions” that do one thing per time.

- So, the compilers for RISC machines have to do more work. One high-level language statement may need to be translated into several simple instructions to let the operands be loaded from the main memory to GPRs before executing an operation, and then store the operation result back to the main memory. Consequently, more RAM is needed to store instructions.

- The primary advantages of RISC machines are:
  - Assuming each simple instruction can be executed with in one clock cycle, completing a complex instruction \texttt{MUL 2 2 5} may take the same amount of time as to execute the four simple instructions. However, since all the simple instructions execute in a uniform amount of time, it is possible to leverage pipelining to reduce the execution time.
  - The hardware for RISC machines is simpler, so it needs less hardware space.
  - RISC is generally lower power. Most embedded system use a RISC architecture.
  - Current CPUs are CISC on RISC with more elaborate microcode under the hood to abstract away the problems of an actual physical CISC implementation.

Types of Operations

- In addition to the internal storage space in a processor, another factor impacting an ISA is the types of operations.

- You need to decide on the types of operations that you want to support in the ISA. Then, you can decide how you want to plan the instruction formats the ISA has and the how bits in each format refers to the operations. Even if the ISA just has one instruction format, the supported operations decide the number of bits the operation field needs to refer to those operations.

- Operations can be roughly grouped into four categories:
  - Data transfer: transfer data from one location to another
- most fundamental type of machine instruction
- e.g., LOAD, STORE, PUSH, POP, etc.

- Arithmetic/logical operations: perform function in ALU
- e.g., ADD, SUBTRACT, MULTIPLE, DIVIDE, AND, OR, NOT, XOR, Shift, etc.
- Logical shift: on one end, the bit shifted out is lost, on the other end, a 0 is shift in
  - Example: 10100110
    - Logical right shift 3 bits: 00010100 (pat the left with 0s)
    - Logical left shift 3 bits: 00110000 (pat the right with 0s)
- Arithmetic shift: treat the data as a signed integer and does not shift the sign bit
  - Example: 10100110
    - Arithmetic right shift 3 bits: 1110100 (pat the left with the sign bits—leftmost bit of the original number)
    - Arithmetic left shift 3 bits: 10110000 (keep the leftmost sign bit and pat the right with 0s)

- Transfer of control: update program counter to execute sequentially or branch
- I/O: transfer information between registers, memory, and input/output devices