Basic ISA classes (cont.)

Register-set Architecture (cont.)

Register-Register
- The registers are used for storing operands. It's also called load-store architecture, as only load and store instructions can have memory operands.
- Most RISCs (reduced instruction set computers) adopt this architecture.
- Suppose we want to ask a Register-Register architecture machine to conduct the same operation $C = A + B$ as above.
- Then, to implement the operation in a Register-Register architecture, we need the following steps, assuming $R1, R2, R3$ are used in this process:

```
LOAD A R1 # load value at location A from main memory to R1
LOAD B R2 # load value at location B from main memory to R2
ADD R1 R2 R3 # add R1 and R2 values and save the result at R3
STORE R3 C # store the value at R3 to location C
```

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.
- Suppose we want to ask a Register-Memory architecture machine to conduct the same operation $C = A + B$ as above.
- Then, to implement the operation in a Register-Memory architecture, we need the following steps, assuming $R1$ and $R2$ are used in this process:

```
LOAD A R1 # load value at location A from main memory to R1
ADD R1 B R2 # add R1 and value at location B, save the result at R2
STORE R2 C # store the value at R2 to location C
```
Memory-Memory
- All operands are memory locations.
- Most CISCs (complex instruction set computers) use this architecture.
- Suppose we want to ask a Memory-Memory architecture machine to conduct the same operation $C = A + B$ as above.
- Then, to implement the operation in a Memory-Memory architecture, we need the following step:

  \[
  \text{ADD A B C} \quad \# \text{add value at location A and value at location B, save the result to location C}
  \]

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 Computer**, and **RISC** is short for **Reduced Instruction Set Computer**. You can tell by their names that they are two types of machines that use different register set architectures. CISC uses memory-memory architecture, and RISC uses register-register architecture.
- Let’s use an example to illustrate CISC and RISC.

![Diagram](image)

- 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**

- 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.

- 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 Loc2 Loc5 Loc2`.
- 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 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**

- 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.
- If the computer is a RISC machine, you will have to break the above instruction \texttt{MUL M(2) M(5) M(2)} into four instructions to finish the product operation, such as:
  \begin{align*}
  \text{LOAD Loc2 A} & \# \text{load the number at the location 2 to register A} \\
  \text{LOAD Loc5 B} & \# \text{load the number at the location 5 to register B} \\
  \text{PROD A B C} & \# \text{conduct A \times B and save the result back to C} \\
  \text{STORE C Loc2} & \# \text{store the value at C back at the location 2}
\end{align*}

- The primary advantages of RISC machines are:
  \begin{itemize}
    \item The hardware for RISC machines is simpler, so it \textit{needs less hardware space}.
    \item Faster execution. Assuming each simple instruction can be executed with in one clock cycle, completing a complex instruction \texttt{MUL Loc2 Loc5 Loc2} 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 \textit{possible to leverage pipelining} to reduce the execution time.
  \end{itemize}

- RISC is generally lower power. Most embedded system use a RISC architecture.

\section*{Types of Operations}
- Depending on the hardware, the operations a computer supports are decided.
- Based on the number of supported operations, the number of bits needed for opcode (operation code) for an ISA is also decided. Depending on the operation and the architecture, the number of operands, where the operands are, and the way to access these operands can be decided. These together decide the instruction format of an ISA.
- Some architectures have multiple instruction formats, while others have a uniform format.

- Operations can be roughly grouped into four categories:
  \begin{itemize}
    \item Data transfer: transfer data from one location to another
      \begin{itemize}
        \item most fundamental type of machine instruction
          \begin{itemize}
            \item e.g., \texttt{LOAD}, \texttt{STORE}, \texttt{PUSH}, \texttt{POP}, etc.
          \end{itemize}
        \item \textit{Things} data transfer instruction \textbf{must specify}
          \begin{itemize}
            \item the \textit{location of the source and destination} operands
              \begin{itemize}
                \item each location could be memory or a register (integer or floating-point registers)
              \end{itemize}
            \item the \textit{length of data} to be transferred (e.g., a word or a halfword)
            \item the \textit{mode of addressing} for each operand (relative/absolute)
              \begin{itemize}
                \item If the operand is an absolute value, it refers to the value at that particular location in main memory.
              \end{itemize}
          \end{itemize}
      \end{itemize}
  \end{itemize}
- If the operand adopts relative addressing, the operand could be a value in 2's complement, which refers to a memory location ahead of or behind the current PC value.
- If the indicated absolute/relative address could be found in cache, load the value from cache. Otherwise, look into the main memory.

• Arithmetic/logical operations: perform functions in ALU
  - Some machines have a set of instructions to support type conversion to facilitate arithmetic operations. So, the machines can support more types of data in addition to the ones that the arithmetic operations support without losing data precision.

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