

## ISA - Instruction Set Architecture (II)

### Basic ISA classes (cont.)

#### 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.
- This architecture allows fast access to temporary values, permits clever compiler optimization, but the instructions are longer than the accumulator designs.
- Operations **need to specify all the operands explicitly**.
- Depending on whether the operands are available in memory or registers, it can be further classified as *register-register*, *register-memory*, and *memory-memory*.

#### Register-Register

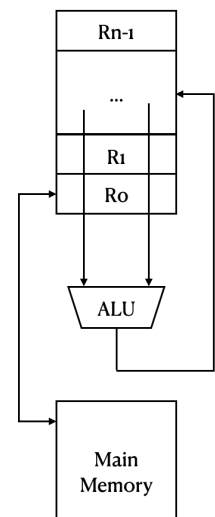
- The **registers are used for storing operands**. It's also called **load-store architecture**, as only load and store instructions can have memory operands.
- Most RISCs (reduced instruction set computers) adopt this architecture.
- Suppose we want to ask a Register-Register architecture machine to 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

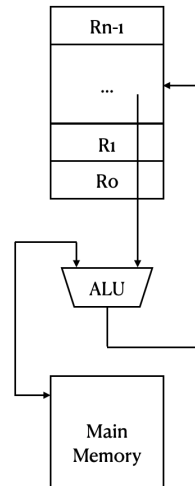
- **One operand is in a register, and one is in memory** (a memory address). The result of ALU stores back in the register.
- This architecture is different from the accumulator architecture in a way that it has multiple registers used to store operands and results.
- This architecture is used by IBM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



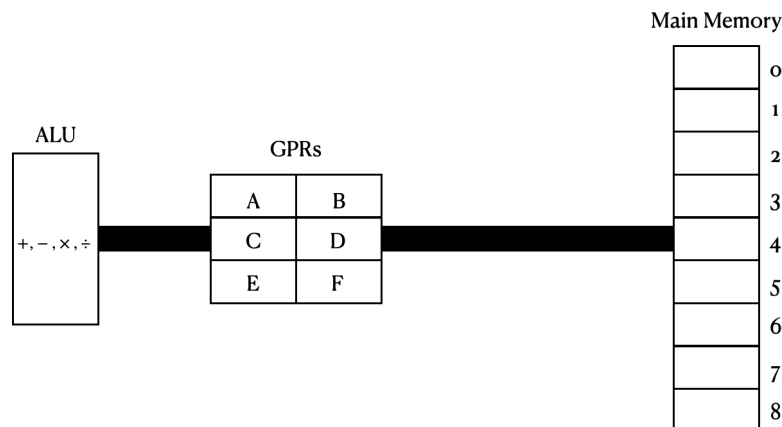
### Memory-Memory

- 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:

**ADD** A B C # 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.



- 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, multiplies 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 operate 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 CISC 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 **MUL Loc2 Loc5 Loc2** into four instructions to finish the product operation, such as :  
LOAD Loc2 A # load the number at the location 2 to register A  
LOAD Loc5 B # load the number at the location 5 to register B  
PROD A B C # conduct A x B and save the result back to C  
STORE C Loc2 # store the value at C back at the location 2
- The primary advantages of RISC machines are:
  - The hardware for RISC machines is simpler, so it **needs less hardware space**.
  - Faster execution. Assuming each simple instruction can be executed within one clock cycle, completing a complex instruction **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 **possible to leverage pipelining** to reduce the execution time.
- RISC is generally lower power. Most embedded systems use a RISC architecture.