ISA - Instruction Set Architecture (III)

CISC vs. RISC (cont.)

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.
- 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
- If the computer is a RISC machine, you will have to break the above instruction MUL 2 5 2 into four instructions to finish the product operation, such as:
  LOAD 2 A # load the number at the location 2 to register A
  LOAD 5 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 2 # store the value at C back at the location 2
- 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 MUL 2 5 2 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.
  • Things data transfer instruction must specify
    - the location of the source and destination operands
      • each location could be memory or a register (integer or floating-point registers)
    - the length of data to be transferred (e.g., a word or a halfword)
    - the mode of addressing for each operand (relative/absolute)
      • If the operand is an absolute value, it refers to the value at that particular location in
        main memory.
      • 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.
  • Examples of IBM EAS/390 Data Transfer Operations
• 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: \(11101011\) (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)
  - 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

- All the operations are specified by the main opcode. So, if a machine supports 32 different operations, the main opcode should be 5 bits ($2^5 = 32$).