Lecture 22, Oct 31, 2022

Introduction to RISC-V

- One of the many instruction set architectures (ISAs)
- A newer, open source instruction set
  - Designed recently so it's less bloated and cleaner
- Different ISAs have different instructions, but some primitives are common across all of them
- The instruction set doesn't define the underlying hardware – it exists as an interface between hardware and software, but the hardware can be implemented in many different ways
- RISC-V comes in different flavours
  - We will be using RISC-V 32I (32-bit integer)
- Instructions define operation and operands
  - Operands can be registers, memory, constants, etc
  - RISC-V has 32 registers, each 32 bits

RISC-V Instructions

- Arithmetic instructions
  - e.g. an add operation:
    - In C: `a = b + c`
    - In assembly: `add s0, s1, s2`
      - `s0` holds `a`, `s1` holds `b`, `s2` holds c
      - `s1, s2` are source operands, `s0` is the destination operand
  - A subtraction would be `sub s0, s1, s2`
  - e.g. `a = b + c - d` is `add t0, s1, s2` and then `sub s0, t0, s3`
- Design principle: make the common case faster
  - Use multiple simple instructions rather than one complex instruction, since simpler instructions are faster in hardware
- Registers
  - Internal to a processor; much faster to access than main memory, but there is a limited number
  - In RISC-V the register set is `x0` to `x31`, but there are special names:
    - `zero` always holds the constant value 0
    - `s0` to `s11`, `t0` to `t6` are the "general purpose" registers, generally used to store variables
    - `ra`, `a0` to `a7` are used for function calls
    - `sp`, `gp`, `fp` are the stack pointer, global pointer, and frame pointer (more on this later)
- Constants ("immediate values")
  - These values are immediately available as part of the instruction (no fetching from memory necessary)
  - Use `addi` instruction: `addi s0, s0, 4` performs `a = a + 4`
    - Note there is no `subi` instruction, but we can use `addi` with a negative number
  - We can also initialize values using immediates, by using an `addi` with the zero register
    - e.g. `addi s4, zero, -78` initializes `s4` to -78
  - Use `0x` prefix for a hexadecimal number, `0b` for a binary number
  - Immediates can only be up to 12 bit two's complement numbers since we need to use the other 20 bits for the instruction
    - The numbers are sign-extended to 32 bits
  - If the numbers are bigger than 12 bits:
    - Use `lui`, load upper immediate, followed by an `addi`
      - `lui` allows specification of a 20-bit value, which is loaded into the most significant 20 bits of the instruction and sets the rest to 0
      - The `addi` can add in the other 12 bits
      - e.g. if we want `a = 0xABCDE123` we can do `lui s2, 0xABCDE` followed by `addi s2, s2, 0x123`
    - Alternatively we can use a pseudo-instruction `li`, load 32-bit immediate, and just do `li s2,`

```
0xABCDE123
```

- The assembler converts the `li` into `lui` and `addi`
- Pseudo-instructions make our lives easier; they are not real instructions but are converted into real instructions by the assembler