Lecture 25, Nov 14, 2022

Program Flow (Continued)

- Jump instructions (unconditional branches): `j LABEL`
  - Jump and link `jal`, jump register `jr` relate to subroutines (function calls)
- In a loop, we jump back to the beginning of the loop if we want to keep looping
- In an if/else statement, we jump over the "if" code if the condition is not true

Examples

- Continuously decrement `s8` until it is zero:

```
LOOP1:
    addi s8, s8, -1 # Decrement s8
    bnez s8, LOOP1 # Jump back to the label if s8 is not zero
```

- Converting from C code:

```
if (s8 > s9) {
    // THEN code
}
else {
    // ELSE code
}
// AFTER code
```

```
    ble s8, s9, ELSE1
THEN1:
    # THEN code
    j AFTER1
ELSE1:
    # ELSE code
AFTER1:
    # AFTER code
```

- Note the conditional jump instructions can only compare against registers, not immediates, so we have to load an immediate into a register first if we want to compare against a constant value
- For loop example:

```
for (s8 = 1; s8 < 5; s8++) {
    s9 = s9 + s10;
}
```

```
    addi s8, zero, 1
    addi t0, zero, 5
LOOP3:
    bge s8, t0, DONE
    add s9, s9, s10
    addi s8, s8, 1
    j LOOP3
DONE:
    # Code after
```

Machine Code

- Assembly language is human readable, but ultimately compiled to machine code

- All instructions are encoded into 32 bits (even if they may not need as many), because regularity supports simplicity, which improves performance
- RISC-V has 4 min instruction formats:
    - R-type (register type): Instructions that use two register source operands, e.g. `add`
        - Bits 31-25 (7) are the function code `func7`
            - These are used if the instruction needs more bits than just the opcode to specify their behaviour
        - Bits 24-20 (5) represent source register #2 `rs2`
        - Bits 19-15 (5) represent source register #1 `rs1`
        - Bits 14-12 (3) are 3 more function bits `func3`
        - Bits 11-7 (5) represent the destination register `rd`
        - Bits 6-0 (7) represent the opcode `op`
            - These identify the operation
    - I-type (immediate type): Instructions that use a register and an immediate, e.g. `addi`
        - Bits 31-20 (12) are the immediate value `imm12`
        - Bits 19-15 (5) represent source register #1 `rs1`
        - Bits 14-12 (3) are 3 function bits `func3`
        - Bits 11-7 (5) represent the destination register `rd`
        - Bits 6-0 (7) represent the opcode `op`
        - Notice the regularity of how the 3 function bits, source register 1, destination register, and opcode are in the same bits as in R-type
    - S/B-type (store/branch type): Storing into memory or branching
    - U/J-type (upper immediate/jump type): Load upper immediate or jump
    - The type of instruction is part of the opcode
- Examples:
    - `add s2, s3, s4` (R-type)
        - Opcode for `add` is 51, `0b0110011`
        - Both `func7` and `func3` are 0
        - `s2` is `x18`, `0b10010`
        - `s3` is `x19`, `0b10011`
        - `s4` is `x20`, `0b10100`
        - The final encoded instruction is `0000000'10100'10011'000'10010'0110011` or `0x01498933`
    - `addi s0, s1, 15`
        - Opcode for `addi` is 19, function bits all 0
        - `s0` is `x8`, `s1` is `x9`
        - The final encoded instruction is `000000001111'01001'000'01000'0010011` or `0x00F48413`