Lecture 26, Nov 15, 2022

Subroutines (Functions)

- Allows code modularization and reuse
- Subroutines have input arguments and return values
- To invoke a subroutine we need to branch into it, but we need to branch back when the subroutine is done, so branching is different here
- The jump and link instruction `jal` is used (the "link" part remembers how to get back)
  - `jal LABEL` jumps to the label, and saves the program counter of the instruction after it into the return address register, `ra`
- To return from a subroutine use the jump register instruction `jr`
  - `jr ra` jumps back into the address in `ra`
  - There is only one `ra` register, so if we want to call subroutines inside subroutines we need to use the stack

Passing Arguments and Returning Values

- Calling subroutines involve a *calling convention*, an agreement between caller and callee
  - The caller and callee need to agree on where the arguments and return values are stored
  - The callee must also not interfere with the behaviour of the caller
    - The subroutine must not change any registers that the caller are using
- In RISC-V the 8 registers `a0` to `a7` are used for function arguments, from left to right
  - If we have more than 8 arguments, we need to use the stack
- The return value is stored in `a0`
- Example:

```
int main() {
    add6(11, 22, 33, 44, 55, 66);
}

int add6(int a, int b, int c, int d, int e, int f) {
    return a + b + c + d + e + f;
}
```

```
_start:
    # Load all the arguments into registers
    addi a0, zero, 11
    addi a1, zero, 22
    addi a2, zero, 33
    addi a3, zero, 44
    addi a4, zero, 55
    addi a5, zero, 66
    # Call subroutine
    jal ADD6
END:
    ebreak

ADD6:
    # Add all the values together
    add s1, a0, a1
    add s2, a2, a3
    add s3, a3, a5
    add s1, s1, s2
    # Set a0 to return value
    add a0, s1, s3
```

```
# Jump back
jr ra
```

- Note problem with this: generally we don't want to use the `s` registers in the subroutine because the caller may be using these!
  - Solution is to use the stack