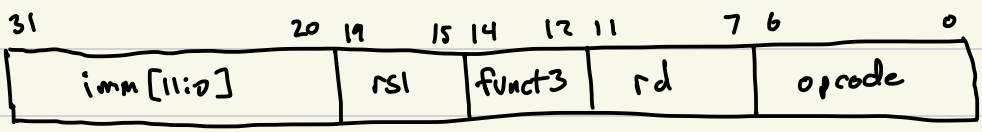
RISC-V Immediates

In some of the RISC-V instruction formats immediate values are embedded into the instruction word. When executing or emulating a RISC-V instruction we need to extract these immediate bits and reconstitute them to become a signed 64-bit value (i.e. int64_t). The signed 64-bit value then can be used a one of the operands for the instruction.

For example, here is the I-type format, for instructions like addi (addi t0, t1, 99):



For addi we need to add rs1 to the immediate. However we first need to extract the imm bits and then then sign extend to 64 bits to make it a signed 64-bit value in two's complement.

Here is how we do this for i-type instructions:

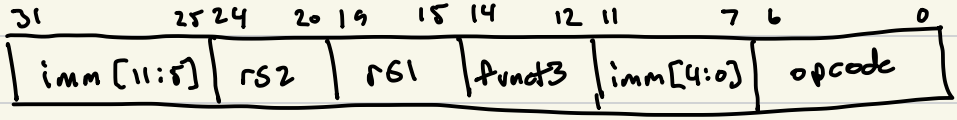
```
uint32_t iw; // assume the 32 bit RISC-V instruction
             // is in iw
uint64_t uimm; // unsigned extracted imm bits
int64_t imm; // final signed 64-bit immediate

uimm = get_bits(iw, 20, 12); // get the upper 12 bits
imm = sign_extend(uimm, 11); // signext to 64 bits
```

Now you can use imm in execution, e.g.,

```
rsp->regs[id] = rsp->regs[rs1] + imm
```

Complicated Immediates

You will notice the the other instruction formats have more complicated immediate embeddings. For example here is the s-type:



In this case we need to extract each immediat part, combine them, then sign extend the combined immediate, like this:

```
uint32_t iw;
uint64_t imm11_5;
uint64_t imm4_0;
uint64_t uimm;
int64_t imm;

imm11_5 = get_bits(iw, 25, 7);
imm4_0 = get_bits(iw, 7, 5);
uimm = (imm11_5 << 5) | imm4_0;
imm = sign_extend(uimm, 11);
```

Notice that we shift imm11.5 to the left by 5 bits to put it in its place, then "or" it with imm4_0. You can extend this technique to extract immediates from the other formats.

Note: if the embedded immediate does not include 0 (zero) then a 0 is implied in the first bit of the final immediate.