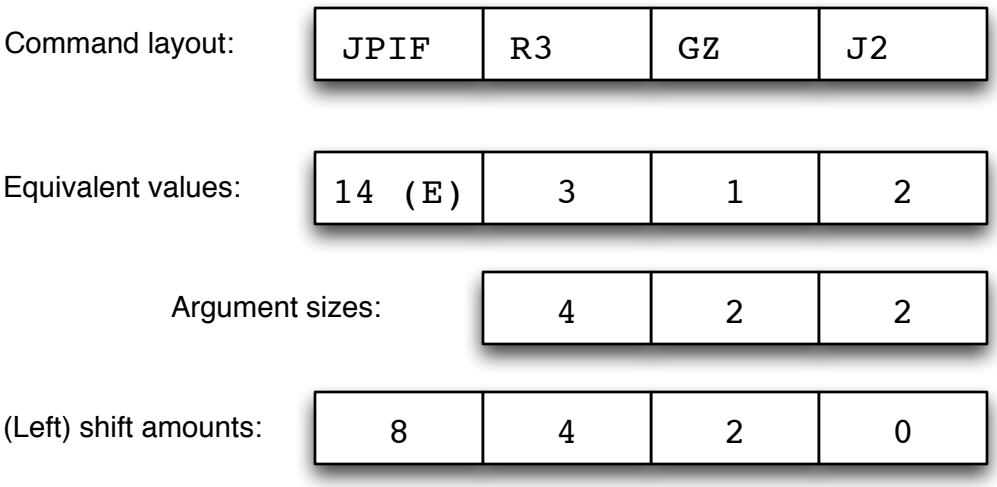
How to check and assemble opcode arguments?

Representation: string (e.g., hex string) or *number?*

Personally, I like to use numbers to represent the individual opcode and argument values; this way they can be easily masked and then combined using bit-shifts and additions. When you're all done with them (i.e., they are all assembled), you can just use "`toHexString`" to print them out.

If you use strings to represent your assembled values, there are awkward points where arguments don't happen to be 4 bits wide, such as the arguments to `DATA`, `JPIF`, `INC`, `SHIFT` and `CONST`. When you use numbers, you can assemble by shifting the pieces according to an accumulated sum of the widths of the arguments to the right:



Command value: `(14<<8) + (3<<4) + (1<<2) + 2`

`Integer.toHexString(` above `) = "E3A"`

Note that you might also want to use the argument size information to compute a mask for the arguments value. Also, if the masked value is not equal to the original, you could warn that the value had to be trimmed to fit the argument "slot".

How to store information about number and size (and other restrictions?) on arguments?

Given that you want to use the above technique to calculate instruction values from argument values and shift amounts, how will you store this information? A (small) array of ints? Perhaps you could squeeze it into an int! (Think about how you would implement an assembler in PC-231 code!)

Error-checking and warnings:

You may or may not want to check and issue warnings for certain kinds of instruction syntax. For example, should the following be allowed? This has an effect on the way you think about how many arguments an opcode takes, as well as the form they take.

```
COPY   xF3       ; same as COPY PC,R3    ??
SHIFT 3,xF       ; same as SHIFT R3,-8   ??
CONST xE3A       ; same as JPIF R3,GZ,J2 ??
```