The author has provided an example of a guessing game and its implementation:

To design a **datapath** for a guessing game system with a digital scale input, W, which gives the object's weight, and a keypad input, K, representing the user's guess as an unsigned integer. The system has three outputs: LOW if the guess is less than 10 below the correct weight, HIGH if it's more than 10 above, and CORRECT if it's within 10. The value 10 is a tolerance used for illustration.

To create the LOW output, we need to evaluate if K < W - 10. In programming terms, this would be:

```
if (K < W - 10)
   LOW = 1;
else
   LOW = 0;
```

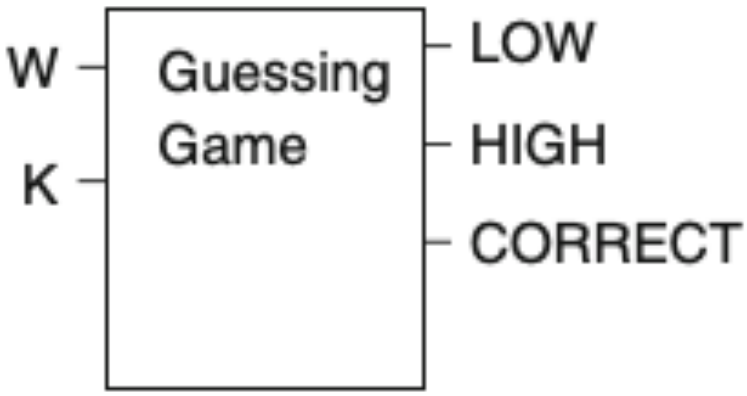
They have used a subtractor and comparator in their **hardware** to evaluate this expression and connect the result to the LOW output.

We can connect components similarly to compute the HIGH output. The subtractor and adder are intuitive, but the comparator evaluates **Boolean expressions**. While programming constructs are straightforward to translate into datapath design, they have use standard datapath components for computations to keep costs low.

The CORRECT output is active when neither LOW nor HIGH is active. They use Boolean connectives to build this circuit, ensuring only one of the three outputs (LOW, HIGH, CORRECT) is active at a time.

Combining these diagrams, we create a complete datapath implementing the algorithm. Although we can optimize the design by combining ALU components or reducing comparators, the initial version serves our purpose. We can use high-level behavioral programs in hardware description languages like Verilog or VHDL to automate the process, but understanding the **lower-level design** helps me gain better control first.

This approach emphasizes translating algorithms from abstract concepts to physical hardware.



final datapath