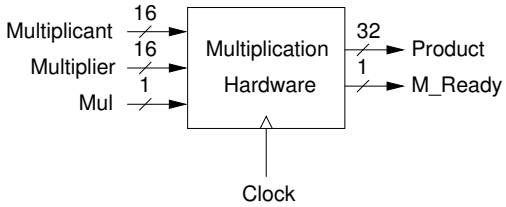
Project 4 - Multiplication and Division Hardware

CS 0447 — Computer Organization & Assembly Language

The purpose of this project is for you to build 16-bit multiplication and 16-bit division hardwares as we discussed in class in `logisim`.

Introduction to the Multiplication Hardware

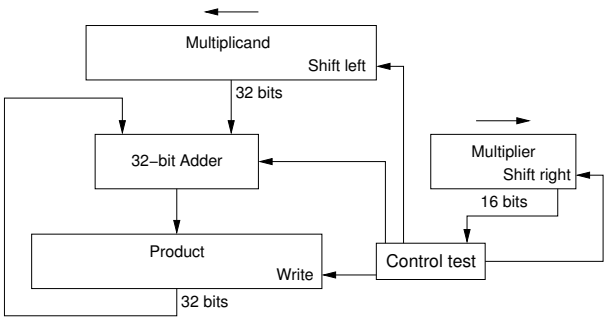
The 16-bit multiplication hardware that we discussed in class is shown below:



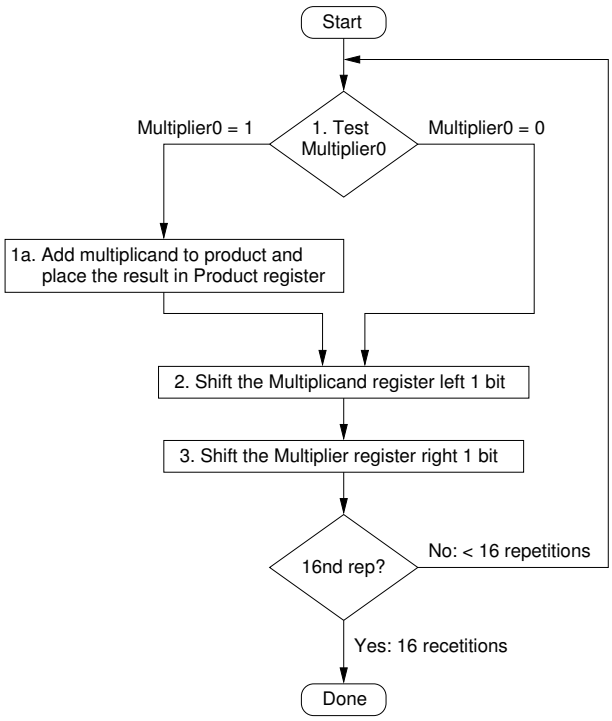
You can consider the above circuit as a sub-circuit named `multiplication` which contains the following input/output:

- `Multiplicand`: a 16-bit input
- `Multiplier`: a 16-bit input
- `Mul` (1-bit input): This input will be one if the instruction is the multiplication instruction
- `Clock` (1-bit input)
- `Product` (32-bit output)
- `M_Ready` (1-bit output): This output will be 1 if the product is ready

Note that we require to have the output `M_Ready` because the multiplication instruction will take multiple clock cycles to produce a result. Ideally, if a CPU see the instruction `mul`, it will set the appropriate `Multiplicant` and `Multiplier`. Then, it will set `Mul` to 1 and wait until the signal `M_Ready` to turn to 1 before it continues to the next instruction. The circuit inside will be the same as the multiplication hardware discussed in class as shown below:

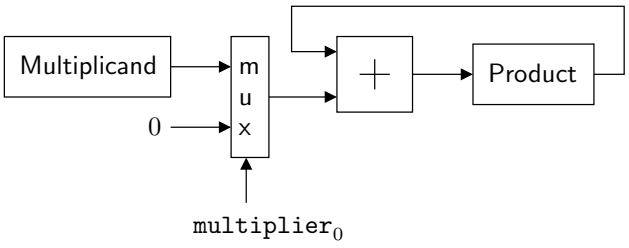


Inside the multiplication hardware, you need three registers, `Multiplicand` (32-bit), `Multiplier` (16-bit), and `Product` (32-bit). For these registers, you do not have build them from scratch. Simply use the register component under "Memory". Similarly, for the 32-bit adder, simply use the one supplied by the `logisim`. Note that the above hardware is for multiplying two 16-bit numbers and produce a 32-bit result. The flowchart of this hardware is shown below:



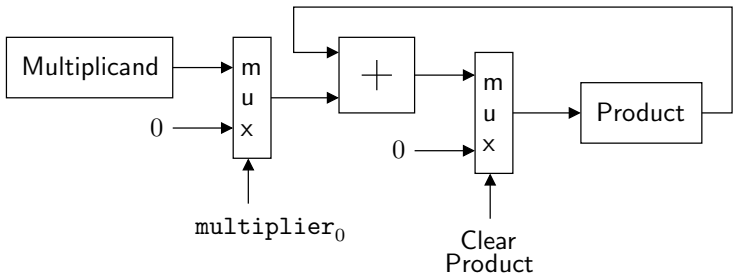
Recall that in the first step, this hardware have to load the top 16-bit of the `multiplicand` register with 0s and the bottom 16-bit with `Multiplicand`, load the `product` register with 0s, and load the `multiplier` register with the `Multiplier`. After all three registers are loaded with proper values, then the algorithm can start as follows.

1. $\texttt{product} = \texttt{product} + (\texttt{multiplicand} * \texttt{multiplier}_0)$: In this step, if $\texttt{multiplier}_0$ is 0, we actually perform $\texttt{product} = \texttt{product} + 0$. But if $\texttt{multiplier}_0$ is 1, we perform $\texttt{product} = \texttt{product} + \texttt{multiplicand}$. This can be done by adding a 32-bit (2-input) multiplexer. This multiplexer has two inputs, one from the `multiplicand` and another one is imply a 32-bit constant 0. Simply use the Least Significant Bit (LSB) of the `multiplier` register ($\texttt{multiplier}_0$) to choose which one to go to the output as shown below:



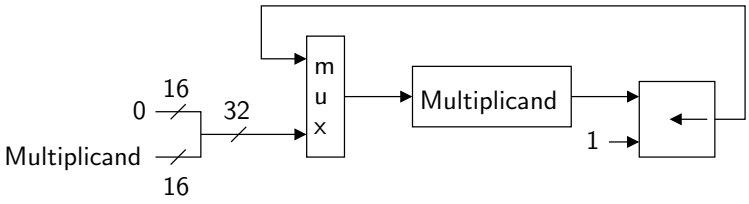
Note that before the algorithm starts, you must clear the `product` register which can be done in two ways:

(a) by writing 0. So, you also need another multiplexer to choose whether you want to write 0 or output from 32-bit adder to the `product` register as shown below:



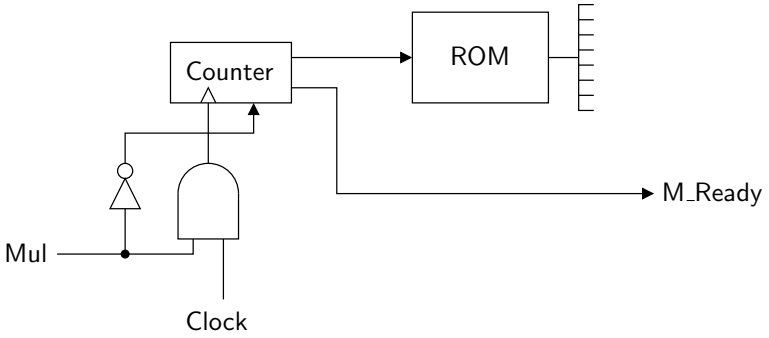
(b) use the `Clear` input pin of the register. Simply set it to 1 and the content will be cleared.

2. Shift `multiplicand` register left one bit: This step is simply update the `multiplicand` register by its data that has been shifted left by 1. Simply use a Shifter provided by `logisim` under Arithmetic. Note at the first step before the algorithm starts, you need to update `multiplicand` register by the input `Multiplicand`. So, you need a multiplexer to select which data should go to the `multiplicand` register (`Multiplicand` input or `multiplicand` $<<$ 1. The block diagram of the circuit is shown below:



3. Shift `multiplier` register right one bit: This step is pretty much the same as in previous step. You need to be able to load the content of the multiplier or update it with `multiplier` $>>$ 1

Note that we need an ability to control what to do at each clock cycle. For example, in the first clock cycle, we need to load contents of all registers. The next clock cycle, we need to perform `product` = `product` + (`multiplicand` $*$ `multiplier`$_0$). The third clock cycle, we need to perform `multiplicand` = `multiplicand` $<<$ 1. The fourth clock cycle, we need to perform `multiplier` = `multiplier` $>>$ 2, and so on. To be able to control each clock cycle, we will use a combination of counter and Read Only Memory (ROM) as shown below:



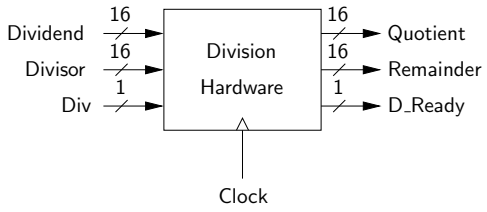
When `Mul` is 1, it will clear the `Counter` to 0. At the same time, it will allow the clock signal to go to the `Counter`. So, the `Counter` will start counting up until its desired maximum value which can be set. When it reaches its maximum value, its `Carry` signal will be 1 which can be used for

the signal `M_Ready`. The output of the `Counter` will be use as the address of a ROM. The content of the ROM will be control signal for each clock cycle. In other words, you can program what do you want to do at each clock cycle by content of the ROM.

Note that you **MUST** set the maximum value of the counter to stay at a specific value based on the number of clock cycles that your hardware uses. **MAKE SURE** that the last output value of the ROM should maintain the output of your product register. When we grade your circuit, we will simply put value of multiplicand and multiplier, and let the clock tick until `M_Ready` turn green **without stopping the clock** and check the result.

Introduction to the Division Hardware

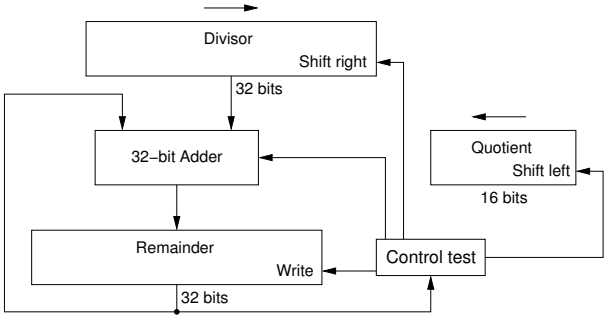
The 16-bit division hardware that we discussed in class is shown below:



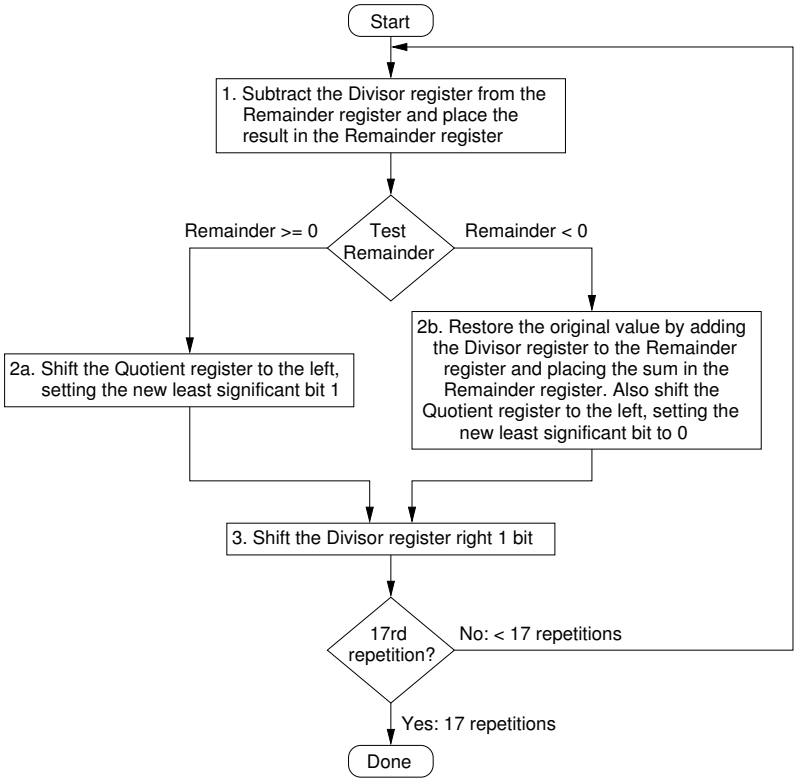
You can consider the above circuit as a sub-circuit named `division` which contains the following input/output:

- `Dividend`: a 16-bit input
- `Divisor`: a 16-bit input
- `Div` (1-bit input): This input will be one if the instruction is the division instruction
- `Clock` (1-bit input)
- `Quotient` (16-bit output)
- `Remainder` (16-bit output)
- `D_Ready` (1-bit output): This output will be 1 if the product is ready

The division hardware that we discussed in class is shown below:



Again, the above hardware is for dividing two 16-bit numbers and produce a 16-bit quotient and 16-bit remainder. The flowchart of this hardware is shown below:



The design concept of this division circuit will be pretty much the same as in multiplication circuit but it requires more steps. For example, when the subtraction result is less than 0, you have to restore to its original value by adding it back. Another different is the quotient, sometime we shift it left and insert a 0 but sometime we insert a 1.

What to Do?

For this project, start with the given starter file named `muldiv.circ`. This starter file contains two sub-circuits, `16-bit multiplication` and `16-bit division`. In both sub-circuit, the counter and ROM are provided. Simply build your multiplication and division circuits there. Once you are finish, put your circuits in the `main` and connect them with appropriate input/output. We will test your circuit from the `main` circuit.

Again, you **MUST** set the maximum value of the counter to stay at a specific value based on the number of clock cycles that each of your hardwares use. We will not stop the clock when we check your results.

Submission

The due date of this project is stated in the CourseWeb under this project. Late submissions will not be accepted. You should submit the file `muldiv.circ` via CourseWeb.