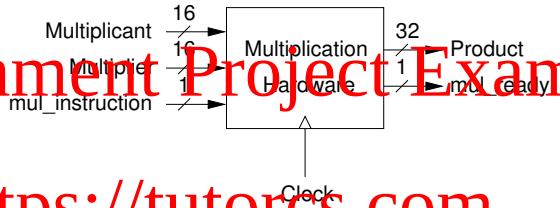
Project 4 - Unsigned Division and Square Root Hardware

CS 0447 — Computer Organization & Assembly Language

The purpose of this project is for you to build a division and a square root hardware for Q8.8 numbers. We will explain the specification of the circuit using an example of a multiplication hardware discussed in class. **Note that you do not have to implement a multiplication hardware for this project**.

Introduction to the Multiplication Hardware

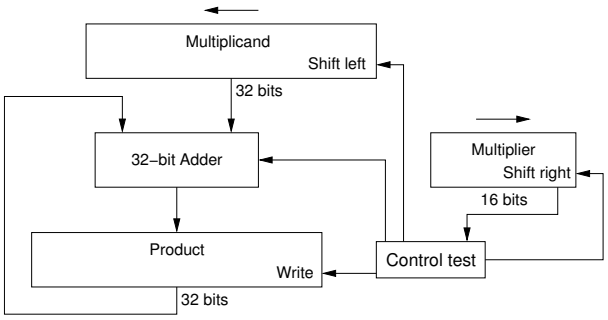
The 16-bit multiplication hardware that we discussed in class is shown below:



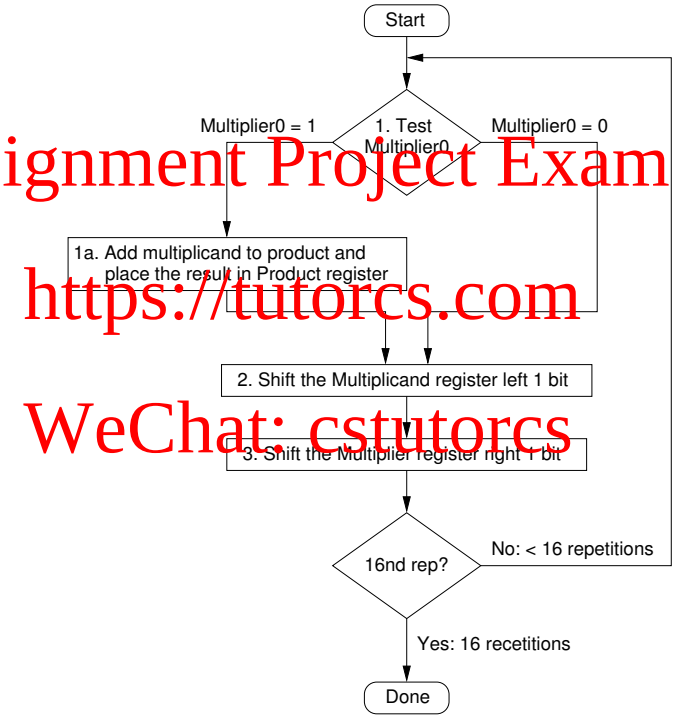
You can consider the above circuit as a sub-circuit named `multiplication` which contains the following input/output:

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

Note that we require to have the output `mul_ready` because the multiplication instruction will take multiple clock cycles to produce a product. Ideally, if a CPU see the instruction `mult`, it will set the appropriate `Multiplicant` and `Multiplier`. Then, it will set `mul_instruction` to 1 and wait until the signal `mul_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:

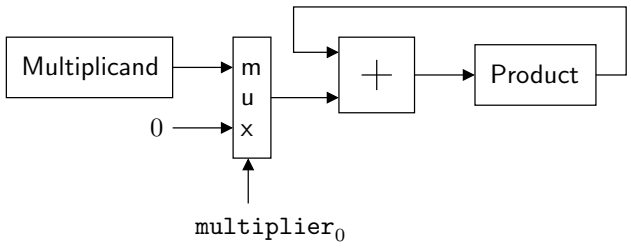


Inside the **16-bit** multiplication hardware, you need three registers, `Multiplicand` (32-bit), `Multiplier` (16-bit), and `Product` (32-bit). For these registers, you do not have build them from scratch. Simply use the register component under "Memory". Similarly, for the 32-bit adder, simply use 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:



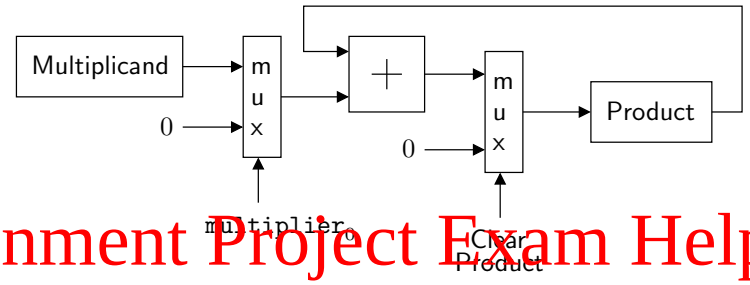
Recall that in the first step, this hardware have to load the top 16-bit of the `multiplicand` register with 0s and the bottom 16-bit with `Multiplicand`, load the `product` register with 0s, and load the `multiplier` register with the `Multiplier`. After all three registers are loaded with proper values,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

1. `product` = `product` + (`multiplicand` $*$ `multiplier`$_0$): In this step, if `multiplier`$_0$ is 0, we actually perform `product` = `product` + 0. But if `multiplier`$_0$ is 1, we perform `product` = `product` + `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 simply a 32-bit constant 0. Simply use the Least Significant Bit (LSB) of the `multiplier` register (`multiplier`$_0$) to choose which one to go to the output as shown below:



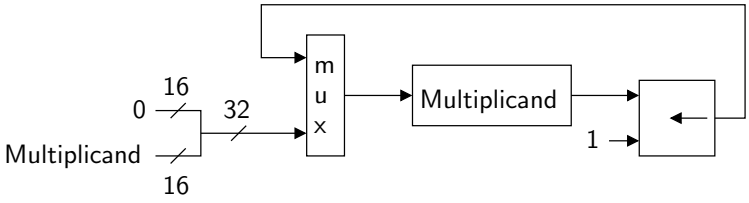
Note that before the algorithm starts, you must clear the `product` register which can be done in two ways:

(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:



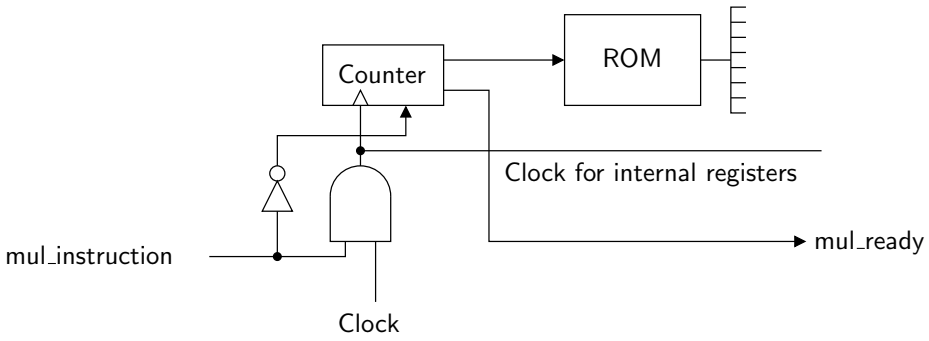
(b) use the `Clear` input pin of the register. Simply set it to 1 and the content will be cleared.

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` * $\texttt{multiplier}_0$). 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:

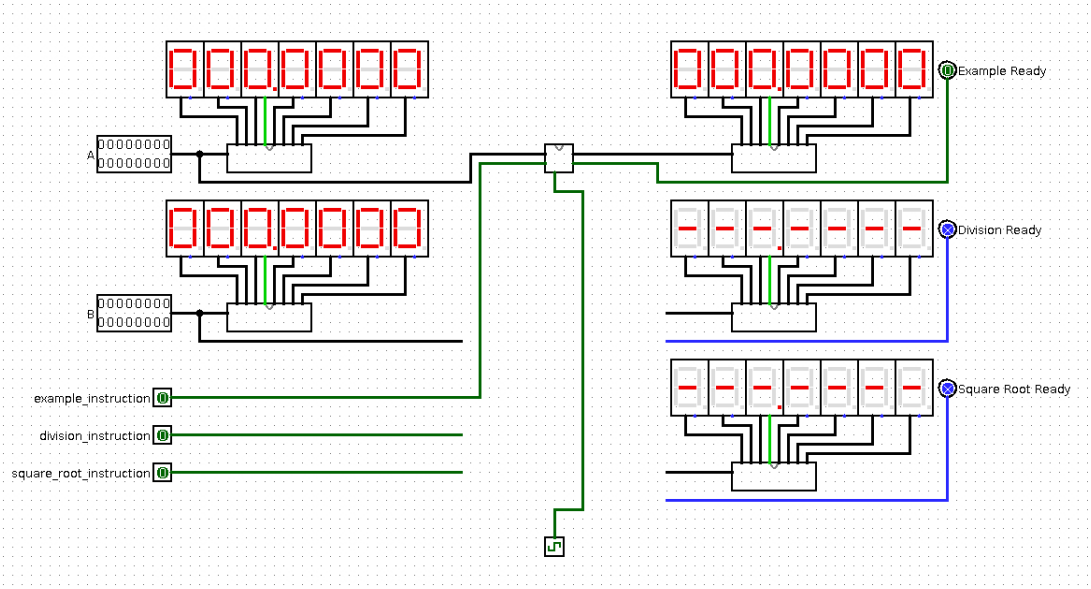


When `mul_instruction` is 1, it will clear the `Counter` to 0. At the same time, it will allow the clock signal to go to the `Counter`. So, the `Counter` will start counting up until its desired maximum value which can be set in `Counter`'s attribute. When it reaches its maximum value, its `Carry` signal will be 1 which can be used for the signal `mul_ready`. The output of the `Counter` will be use as the address of a ROM. The content of the ROM will be a control signal for each clock cycle. In other words, you can program what do you want to do at each clock cycle using the content of the ROM.

IMPORTANT The first instruction (control signal) of the ROM at the address 0x00 should be 0x00. **Make** sure that the control signal 0x00 should set the "Enable" input of each register to be 0 so that they will maintain their value. This is very important especially for the last control signal. The last control signal should be 0x00 as well. When we test your circuit, we will simply let the click ticks continuously. Once we see that the result is ready, we will look at the result without stopping the clock. Thus, your circuit must maintain the result. If you did not set the "Enable" input of your product register to 0, it will keep updating the content of the product register continuously. **Do not forget to set the counter's attribute "Maximum Value" to the address of your last instruction**.

Example Circuit

For this project, a starter file named `project4_div_sqrt.circ` is given. The main circuit of file is shown below:



Two display on the left side are for inputs A and B. The inputs A and B are in Q8.8 format. Both LED displays will show the approximate values of A and B in decimal with decimal point. On the right, there are three LED displays. The one on the top shows the result of the example circuit which will be explained later. The one in the center should be used to display the result of A/B. Similarly, the one at the bottom should be used to display the result of $\sqrt{A}$.

The given file contains four sub-circuits, Q8.8 to Decimal, Example, Division, and Square Root. **Do not modify both Q8.8 to Decimal and Example sub-circuits**. What do you need to do for this project is to implement **Division** and **Square Root** sub-circuits.

In the **Example** sub-circuit. You will see a circuit that perform a very simple tasks. The result of this circuit can be represented by the following equation:

$$result = (2 \times ((2 \times A + 1.0) - A) + 1.0) - A$$

which can be separated into three steps:

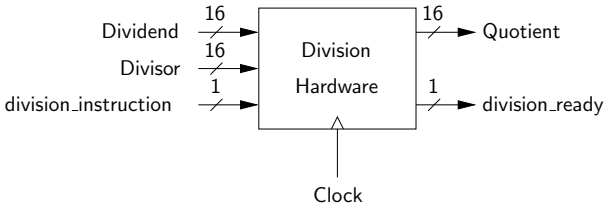
1. $result = A$
2. $result = (2 * result + 1.0) - result$, and
3. $result = (2 * result + 1.0) - result$.

Double click the "Example" circuit to see how it was implemented. Content of the ROM, and attribute of the "Counter".

Note that some components have been placed into both "Division" and "Square Root" sub-circuits. Simply add additional components as you needed. **You must set the "Trigger" attribute of your "registers" component to "Falling Edge" to match with the trigger of the given "Counter".**

Introduction to the Division Hardware

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

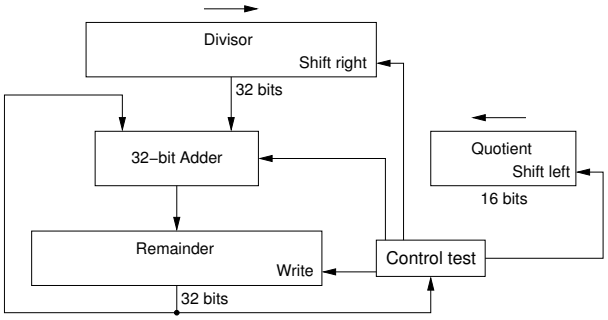


You can consider the above circuit as a sub-circuit named `division` which contains the following input/output:

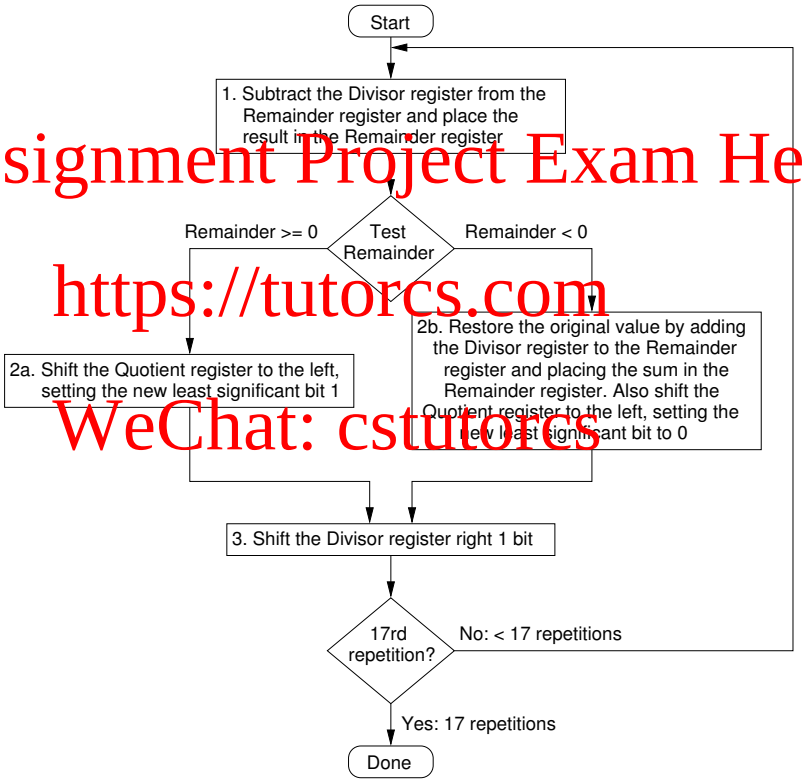
- `Dividend`: a 16-bit input
- `Divisor`: a 16-bit input
- `division_instruction` (1-bit input): This input will be one if the instruction is the division instruction
- `Clock` (1-bit input)
- `Quotient` (16-bit output)

- `division_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:



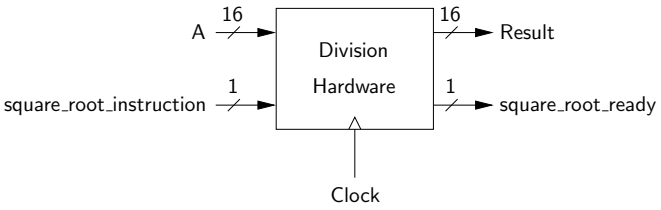
Again, the above hardware is for dividing two 16-bit numbers and produce a 16-bit quotient and 16-bit remainder. The flowchart of this hardware is shown below:



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. **Note** that this division circuit is for Q8.8 format. So, you need two 32-bit registers for Remainder and Divisor. Do not forget that Q8.8 divided by Q8.8 results in Q8.0. So, to get Q8.8 format, first you need to shift the dividend left by 8 and put it in the Remainder register. For the divisor, shift left 16 and put the result into the register Divisor as usual. The quotient should be a 16-bit register where its output should connect directly to the given "Quotient" output port.

Introduction to Square Root Hardware

For the square root hardware, it should look like the following:



This hardware should take an input from A and calculate the square root of A ($\sqrt{A}$). The instruction how to calculate a square root is provided in the project 1. For this hardware, use your imagination to implement this hardware. **Hint**: Try to break the square algorithm into smaller steps and translate them into a hardware.

What to Do?

As mentioned earlier, for this project, start with the given starter file named `project4_div_sqrt.circ`. This starter file contains two sub-circuits, `division` and `Square Root`. In both sub-circuit, the counter and ROM are provided. Simply build your division and square root 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. **Note that you only need to implement unsigned division. We will only test your circuit with unsigned numbers**.

Grading Rubric

The grading criteria for this project is shown below:

- 60 points for the unsigned division hardware, and
- 40 points for the square root hardware.

So, make sure at least your unsigned division hardware works.

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 `project4_div_sqrt.circ` via CourseWeb.