DDCA-u09-lab

Manual

Option 1 (challenging): Use the MARS simulator to write a faster version of the code that you have written in Lab 7. Make sure that the code is functional by testing it for smaller values[4]. If you have problems with the MARS simulator or assembly, refer to Lab 7.

Option 2 (easy): Download and extract the Lab9_helpers.zip file from the course website. This file contains a faster version of the assembly code already implemented for you in “helper_mul.asm”. You can also try to run it in the MARS simulator to convince yourself that the code is correct.

Download the Lab9_student.zip file from the course website. It contains a Vivado project with the processor designed in Lab 8 (MIPS.v) and a testbench (MIPS_test.v) to test the processor. If you look at the MIPS.v file, you may notice that we have changed the output of the processor to make debugging easier. The project also contains the ALU.v from Lab 8, which you have to modify.

Your task is to modify the ALU component so that:

1. It accepts a 6-bit aluop signal.
2. It accepts a 5-bit ShAmt (shift amount) from the MIPS.

3. It takes input B and shifts the value by ShAmt bits to the right when aluop is 6’b000010 (**srl**).
4. It multiplies A and B and writes the result to an internal 32-bit register Lo when aluop is 6’b011001 (**multu**). Note that for the register you will need to add clock and reset signals to the interface as well.
5. It takes the present value of the register Lo and copies it to the output when aluop is 6’b010010 (**mflo**).

Make sure that other instructions are not affected.

Make sure that the new ALU is integrated correctly at the top level. This requires small changes to the module instantiation within the MIPS.v file. You will need to extract the ShAmt signal (Shift Amount) from the instruction (Instr signal) to pass it to the ALU. You may need to declare additional wires for this. You can find bit positions for ShAmt from MIPS reference data:

https://safari.ethz.ch/ddca/spring2024/lib/exe/fetch.php?media=mips_reference_data.pdf

Show and describe your design modifications to a TA.

Option 1 (challenging): Your assembly program needs to be copied into the text file named “insmem_h.txt”. To do this, you can use the Memory Dump option within the MARS simulator. By selecting the “memory segment” as “.text” and “Dump Format” as “Hexadecimal Text” you will be able to generate the required file. All you have to do is to use an editor and make sure that the file has exactly 64 lines; all lines after your real code will be filled with zeroes. If something is not clear, refer to Lab 7.

Option 2 (easy): The Lab9_helpers.zip includes a “helper_insmem_h.txt” file that contains the binary program corresponding to the “helper_mul.asm” assembly program. Rename “helper_insmem_h.txt” to “insmem_h.txt” and place it in the same folder as the project files.

Use the file MIPS_test.v file to test your processor as explained previously. It is a simplified version of the one used in Lab 6 in which we no longer have to read in the expected responses. We simply need a clock generator, an initial reset signal and give the processor sufficient time to finish the calculation. Run the new test bench in the Vivado simulator and monitor the value of ‘result’ and the PC in the wave window.

Show your correctly running MIPS code to a TA.

Report

(1)

For the following values of A and B, how many clock cycles are needed to execute your first program from Lab 7 on your baseline MIPS processor, before adding optimizations of Lab 9? Assuming that we run the MIPS processor at 20 MHz, how much time (in seconds) would that take? (You can assume that the loading of the numbers requires only one instruction)

Value of A	**Value of B**	**Number of cycles**	**Time in seconds**
0	8		
6	8		
0	90’000		
89’996	90’002		

```
main:
    addi    $s0,    $0,     0           # A = 0
    addi    $s1,    $0,     8           # B = 8
    addi    $t2,    $0,     0           # S = 0
    slt     $t1,    $s0,    $s1         # $t1 = A < B ? 1 : 0
    beq     $t1,    $0,     end         # If A > B, jump to end
    j       loop                        # Jump to loop

loop:
    add     $t2,    $t2,    $s0         # S = S + A
    beq     $s0,    $s1,    end         # If A == B, jump to end
    addi    $s0,    $s0,    1           # A = A + 1
    j       loop                        # jump to loop

end:
    j       end                         # Infinite loop at the end
```

The formula to calculate the number of cycles is given by

$$N(A,B) = \begin{cases} 6 + 4 \cdot (B - A) + 2, & A \leq B \\ 5, & A > B \end{cases}$$

A	**B**	**Number of cycles**	**Time in seconds**
0	8	40	0.000002s
6	8	16	0.0000008s
6	90'000	359'984	0.0179992s
89'996	90'002	32	0.0000016s

(2)

Fill in the new values for the Table in Exercise 1 when using the modified MIPS architecture running the optimized code, as discussed in the manual for Lab 9. (You can assume that the loading of the numbers requires only one instruction)

Value of A	Value of B	Number of cycles	Time in seconds
0	8		
6	8		
0	90’000		
89’996	90’002		

```
.text
main:
        addi    $t0,    $0,     0               # $t0 = A
        addi    $t1,    $0,     8               # $t1 = B
        addi    $t2,    $t0,    -1              # A-1
        multu   $t0,    $t2                             # A (A-1)
        mflo    $t0                                             # mult result in t0
        srl     $t0,    $t0,    1               # divide by two

        addi    $t2,    $t1,    1               # B+1
        multu   $t1,    $t2                             # B (B+1)
        mflo    $t1                                             # mult result in t1
        srl     $t1,    $t1,    1               # divide by two
        sub     $t2,    $t1,    $t0             # end result is the difference

end:
        j               end                                             # loop t2 is
the result
```

The formula to calculate the number of cycles is given by

$$N(A,B) = 11$$

A	B	Number of cycles	Time in seconds
0	8	11	0.00000055s
6	8	11	0.00000055s
6	90'000	11	0.00000055s
89'996	90'002	11	0.00000055s

(3)

Compare the size/device utilization of the two implementations (before and after the modifications in Lab manual 9). What differences do you see? Briefly comment on them.
Hint: Look into the synthesis report.

Without additional instructions implemented:

```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  461 |     0 |     20800 |  2.22 |
|   LUT as Logic             |  205 |     0 |     20800 |  0.99 |
|   LUT as Memory            |  256 |     0 |      9600 |  2.67 |
|     LUT as Distributed RAM |  256 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers            |   31 |     0 |     41600 |  0.07 |
|   Register as Flip Flop    |   31 |     0 |     41600 |  0.07 |
|   Register as Latch        |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                   |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                   |    0 |     0 |      8150 |  0.00 |
+----------------------------+------+-------+-----------+-------+
```

with additional instructions implemented:

```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  629 |     0 |     20800 |  3.02 |
|   LUT as Logic             |  373 |     0 |     20800 |  1.79 |
|   LUT as Memory            |  256 |     0 |      9600 |  2.67 |
|     LUT as Distributed RAM |  256 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers            |   64 |     0 |     41600 |  0.15 |
|   Register as Flip Flop    |   64 |     0 |     41600 |  0.15 |
|   Register as Latch        |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                   |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                   |    0 |     0 |      8150 |  0.00 |
+----------------------------+------+-------+-----------+-------+
```

As evident, implementing the additional functionality increases the number of used LUTs (logic) by nearly 50 percent and doubles the number of registers. This surprised us a lot!