Jetic Gū

1. Handwritten submissions and proprietary formats (e.g. Pages or MS Word) **will not be graded**.
2. Mathematical expressions must be written entirely using LaTeX, otherwise **50%-100%** of marks will be deducted.
3. Circuits must be **tested**. Untested circuits will receive 0.

Submission File structure:

```
submission.zip
    — mux8_16.sim.vwf
    — reg_arr.sim.vwf
    — au.sim.vwf
    — lu.sim.vwf
```

Lab 3

1. Register Array Design

    1. Implement a `dec_3`, a 3-to-8 decoder using Verilog. Ensure that the whole design is combinational. It should have `a[2:0]` as input, a chip enabler input `en` (when `en=0`, output 0 only), and `d[7:0]` as output.

    2. Implement a 16bit 8-to-1 Multiplexer called `mux8_16`. Here's its interface:

```
1.      module mux8_16 (
2.          input  wire [15:0] d0, d1, d2, d3, d4, d5, d6, d7,
3.          input  wire [2:0]  s,
4.          output wire [15:0] y
5.      );
```

    - This component must be tested, and you will need to submit `mux8_16.sim.vwf` as proof (1pt)

    3. Use `dec_3` and `mux8_16`, and all other necessary components to build a register array with 8 x 16bit registers (3pt).

```
1.      module reg_arr (
2.          input  wire rw, reset, clk,
3.          input  wire [15:0] rd_data,
4.          input  wire [2:0]  rn, rm, rd,
5.          output wire [15:0] rn_data, rm_data
6.      );
```

    - The `rw` signal here is used to control the `en` pin of your `dec_3`. so we can prevent the register array from accepting new values when that's not desired.
    - `rd_data` will carry data that will be stored in the register array, specifically one specified by `rd` (destination).
    - `rn` and `rm` will select registers for output buses. In ARM architecture, `rn` is called the base or first operand register, `rm` is called the optional shift or second operand register.

- You should implement all of the above in `reg_arr.v`, and have `reg_arr.sim.vwf` submitted as proof you've tested your component (3pt).

2. ALU design.

1. You will need to design an arithmetic unit, and a logical unit separately.
2. For the Arithmetic Unit, it needs to accept as Input, 3bit of `opB` , 8bits of Immediate (`imm`), 2bits `mode`, 16bit input buses `rn_data` and `rm_data`. Finally it produces 16bit output `rd_data`. You can assume all additions and subtractions are unsigned.

opB	mode	Instruction	Assembly	See
000	**-**	Logical Shift Left	-	Don't care
001	**-**	Logical Shift Right	-	Don't care
010	**-**	Arithmetic Shift Right	-	Don't care
011	**00**	Addition	`ADDS <Rd>, <Rn>, <Rm>`	`Rd_data = Rn_data + Rm_data`
011	**01**	Subtraction	`SUBS <Rd>, <Rn>, <Rm>`	`Rd_data = Rn_data - Rm_data`
011	**10**	Addition (Immediate)	`ADDS <Rd>, <Rn>, #<imm3>`	`Rd_data = Rn_data + Imm[2:0]`
011	**11**	Subtraction (Immediate)	`SUBS <Rd>, <Rn>, #<imm3>`	`Rd_data = Rn_data - Imm[2:0]`
100	**-**	Move	`MOVS <Rd>, #<imm8>`	`Rd_data = Imm`
101	**-**	Compare	-	Don't care
110	**-**	Add 8bits of immediate	`ADDS <Rdn>, #<imm8>`	`Rd_data = Rn_data + Imm`
111	**-**	Unsigned Subtract 8bits of immediate	`SUBS <Rdn>, #<imm8>`	`Rd_data = Rn_data - Imm`

- The Arithmetic Unit should be shown tested in `au.sim.vwf`. (3pt)

3. For the Logical Unit, it needs to accept 4bits of `opB` , 16bit input buses `rn_data` and `rm_data`, 16bit output bus `rd_data`.

OpcodeB	Instruction	Assembly	See
0000	Bitwise AND	`ANDS <Rdn>, <Rm>`	`Rd_data = Rn_data & Rm_data`
0001	Bitwise Exclusive OR	`EORS <Rdn>, <Rm>`	`Rd_data = Rn_data ^ Rm_data`
0010	Logical Shift Left	-	Don't care

OpcodeB	Instruction	Assembly	See
0011	Logical Shift Right	–	Don’t care
0100	Arithmetic Shift Right	–	Don’t care
0101	Add with Carry	–	Don’t care
0110	Subtract with Carry	–	Don’t care
0111	Rotate Right	–	Don’t care
1000	Test	–	Don’t care
1001	Reverse Subtract from 0	–	Don’t care
1010	Compare High Registers	–	Don’t care
1011	Compare Negative	–	Don’t care
1100	Bitwise OR	`ORRS <Rdn>, <Rm>`	`Rd_data = Rn_data | Rm_data`
1101	Multiply Two Registers	–	Don’t care
1110	Bitwise Bit Clear	–	Don’t care
1111	Bitwise NOT	`MVNS <Rd>, <Rm>`	`Rd_data = ~Rn_data`

- The Logical Unit should be shown tested in `lu.sim.vwf`. (3pt)