Stage 2: A simple processor design

This stage involves putting together the four modules designed at stage 1, namely ALU, Register File, Program Memory and Datat Memory, to form a simple processor which can execute the following subset of ARM instructions with limited variants/features.

{add, sub, cmp, mov, ldr, str, beq, bne, b}

Though the ALU has been designed for a larger set of operations, only some of these will get exercised at this stage.

A. Instruction variants to be implemented

DP instructions:

- Register operands as well as immediate operands, with no shift/rotate (LSL #0).
- S bit always 0 for {add, sub, mov}, always 1 for {cmp}.

DT instructions:

- Only immediate offsets (addition as well as subtraction option).
- Only word transfers, no write back, only pre-indexing.

Branch instructions:

- Only {b, beq, bne}.

B. Assumptions

For this design, assume that the Program Counter is a separate register and has nothing to do with R15 of register file.

Assume that Program Memory and Data Memory have independent address spaces. That is, both have byte addresses 0 to 255 (or word addresses 0 to 63).

C. New components do be designed

i. Program Counter with next address logic:

This normally updates the Program Counter on every clock by adding 4. However, if the current instruction is a branch instruction, and the specified condition is true, it adds appropriate offset to the Program Counter.

ii. Flags and associated circuit:

On every clock it conditionally updates the Flags. A detailed note on this has been provided separately.

iii. Condition Checker:

This is a combinational circuit that looks at the Flags and condition field of the instruction to decide whether the specified condition is true or not. Although only EQ and NE conditions are relevant for this stage, this is a simple circuit and may as well be designed for all the conditions.

iv. Instruction Decoder:

This is a combinational circuit that looks at an instruction and identifies its classes and various subclasses.

Design of this is being posted on moodle along with a package defining some useful types and subtypes and may be used freely.

D. Overall Design

The overall design may be based on slide 40 of Lec09. Once the components discussed above and those that were designed at Stage 1 are in place, the rest of the design is quite straight forward. It would consist of instantiation of these components, declaration of local signals that will connect these instances and some glue logic. This glue logic takes care of the following.

- Selects the second operand of the ALU
- Decides the operation to be done by the ALU
- Selects the data to be written into Register File
- Selects the second read address of the Register File
- Generates the enable signal for Register File write
- Generates the enable signal for Data Memory write

E. Testing the design

It is advisable to test the components individually before putting them together. For testing the entire design, some program needs to be placed in the Program Memory. For this some small test programs may be written in assembly and code may be generated using ARMSim#.

Examples of Program Memory initialization are shown below.

```
signal mem_array : memory_type :=
                   (0 => X"E3A0000A",
                    1 => X"E3A01005",
                    2 => X"E5801000",
                    3 => X"E2811002",
                    4 => X"E5801004",
                    5 => X"E5902000",
                    6 => X"E5903004",
                    7 => X"E0434002",
                    others => X"00000000"
                   );
```

```
.text
mov r0, #10
mov r1, #5
str r1, [r0]
add r1, r1, #2
str r1, [r0, #4]
ldr r2, [r0]
ldr r3, [r0, #4]
sub r4, r3, r2
.end
```

```
signal mem_array : memory_type :=
                   (0 => X"E3A00000",
                    1 => X"E3A01000",
                    2 => X"E0800001",
                    3 => X"E2811001",
                    4 => X"E3510005",
                    5 => X"1AFFFFFB",
                    others => X"00000000"
                   );
```

```
        .text
        mov r0, #0
        mov r1, #0
Loop: add r0, r0, r1
        add r1, r1, #1
        cmp r1, #5
        bne Loop
        .end
```

These are just samples. You would need more test programs to check your design.

F. Report

Include your main design decisions/features in your report along with testing/simulation results. Do synthesis to check synthesizability. Zip the report and all design files together and submit.