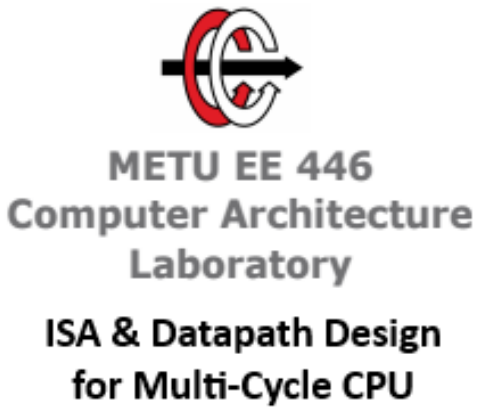

Laboratory Work 4 - ISA & Datapath Design for Multi-Cycle CPU

Objectives

The purpose of this laboratory work is to understand theoretically and design the operation of the datapath of multi-cycle CPU, construct and embed it on a development board, DE0-Nano, and finally, validate its operation.

The design procedure will be based on provided high level specifications and constraints, then implement certain modifications on the CPU core template covered in lectures in order to design a capable general purpose computer CPU datapath. The functional validation of the proposed datapath will be done by providing the control signals using a testbench.

1 Preliminary Work

To fulfill the requirements of this laboratory work, the following tasks should be performed.

1.1 Reading Assignment

The laboratory manual where the regulations and some other useful information exist is available at the ODTUClass course page. Read that manual thoroughly. If you feel yourself unfamiliar with **multi-cycle CPU architecture**, please refer to the corresponding lecture notes of EE446 course.

1.2 ISA Configuration (10% Credits)

For this part, you will design the instruction set architecture (ISA) that will operate on the designed CPU. The ISA needs to support general purpose computing, and therefore will contain different instructions for arithmetic-logic, memory load-store, and program control operations. There should be sufficient variety of addressing modes to support both simple constants and variables, but also more structures such as indexed arrays.

Similar to the ISA in Laboratory Work # 3, the ISA configuration is expected to include the following instructions and optional extensions of your preference. Please note that this instruction set is not specified on ARM.

1. Arithmetic Operations
    - addition
    - addition indirect (bonus)
    - subtraction
    - subtraction indirect (bonus)
2. Logic Operations
    - and
    - or
    - xor
    - clear
3. Shift Operations
    - rotate left
    - rotate right
    - shift left
    - arithmetic shift right
    - logical shift right
4. Branch Operations
    - branch unconditional
    - branch with link
    - branch indirect with link
    - branch if zero
    - branch if not zero

- branch if carry set
- branch if carry clear

5. Memory Operations
    - load to register from memory
    - load immediate to register
    - store from register to memory

You are not expected to convert the instructions into its binary equivalents within the scope of this work; however you are to group the instructions that will use the same path for execution and state which modules (registers, computational units, etc.) which will help you forge an architecture for the entire ISA. Generate your own mnemonics for the instructions and define the ordering of the operands. Please note that this structure is highly essential for the flow of data within the datapath, as the inconsistencies within the ordering would result in a more complex architecture.

1.2.1 Memory/Register Transfer Instructions

The data and instructions to be executed on this multi-cycle architecture is stored in the instruction/data memory. MRT instructions are responsible for the bi-directional transfer of data between the memory and the registers. For loading of data to the registers, two addressing modes are used, namely direct addressing and immediate addressing, whereas only direct addressing is used for data saving on the memory. To illustrate, reference pseudo instructions may be examined:

```
R0 <-- MEM1; load with direct addressing
R3 <-- DATA; load with immediate addressing
MEM2 <-- R2; data store into memory
```

The length of the registers is expected to be 8-bits; however there is no strict criterion on the length for the data/instruction memory. Instead, you are expected to determine the optimum value for the design depending on the architecture.

1.2.2 Arithmetic Instructions

The arithmetic operations, namely addition and subtraction, are realized on 8-bit signed/unsigned operands. Direct operations use two registers from the register file as operands and the results is saved again to a register, whereas in indirect ones (bonus), the second operand is specified as a memory address. Example pseudo instruction may be stated as follows:

```
R0 <-- R1 + R2; direct addition
R3 <-- R1 - MEM; indirect subtraction  (bonus)
```

You may use the ALP that was designed in the very first laboratory work, with modifications if necessary. **For the implementation of the bonus indirect operations:** Please note that indirect operations require an extra data access step from the corresponding memory location since the architecture expected in this laboratory work restricts the ALP operations to registers of the register file, therefore direct transfer between the instruction/data memory.

1.2.3 Logic Instructions

The arithmetic operations, namely addition and subtraction, are realized on 8-bit signed/unsigned operands, similar to the arithmetic instructions and the ALP of designed earlier can be used once again. The pseudo instruction may be given as

```
R1 <-- R1 & R3; logical and
R2 <-- R1 X R0; logical exor
R0 <-- clr(R0); clear
```

You may use the ALP that was designed in the very first laboratory work, with modifications if necessary. Please note that indirect operations require an extra data access step from the corresponding memory location as the architecture expected in this laboratory work restricts the ALP operations to the registers of the register file, therefore direct transfer between the instruction/data memory and the ALP is not permitted.

1.2.4 Shift Instructions

The shift operations that are included in the ISA involves rotation and logical & arithmetic shifting of the register data, accepting merely a single operand (register). For the rotation operation consider the 8-bit data only, which means a 9th bit, the carry bit, shall not be taken into account. You may use the shift register implementation from the previous laboratory work, with the required modifications. This part of the ISA may be illustrated as

```
R2 <-- {R2[6:0],0}; shift left
R0 <-- {R0[0],R0[7:1]}; rotate right
R1 <-- {0,R1[7:1]}; logical shift right
```

1.2.5 Branch Instructions

Within this very last section of the ISA, program control instructions are covered. Branch instructions are essential to maintain the the modularity and conditionality of the routines that are executed on the processor. You are expected to handle unconditional as well as conditional branching where the change in the condition code register (operation flags) are to be considered. Also, it is to be noticed that certain branch instructions alter the value of additional registers as well as the program counter (PC).

For all the instructions stated above:

1. Group the instructions that will operate in a similar manner and state which functional component/units they require in common
2. Come up with a mnemonic that would represent the operation in the instruction and state why you prefer the specific ordering of the operands

1.2.6 Multi-cycle CPU Datapath Design (60% Credits)

A processor, in brief, is composed of two parts, one of which is the datapath where all the where the desired data manipulation is conducted. The latter, the control unit, which is beyond the objectives of this laboratory work, is the entity that directs the operation of the processor. In this multi-cycle processor implementation, the von Neumann architecture with fetch-decode-execute phases is to be considered.

In this part of the laboratory work, given your ISA, you are expected to design a full datapath that would support all the instructions specified. The instructions are stored in a unified instruction/data memory, IDM, read from and executed. As a design limitation, the IDM is to have at least 2^6 memory slots of W-bits. You may either declare a novel functional unit to be used as the IDM. SDRAM usage can be an option, with the required control signals, provided that you embed the code to operate on the FPGA.

As the operations are realized on the registers as stated in the ISA, a register file, of 8 registers, is required. Once again you may use the extended version of your previous code or rewrite it from scratch. Please note that, along with the multi-purpose register, the architecture is to include specific registers such as PC and LR. You may either implement these as two of the 8 registers in the register file or

prefer a separate implementation. However, this choice of implementation may alter the datapath and inevitably the whole architecture.

As stated in the previous subsection, the previous ALP can be used, with or without modification, which would inherently affect the control signals required for the proper operation. As another design limitation, you are allowed to use only a single arithmetic logic processor and no wired connection between the ALP and the IDM is allowed. Besides, you may as well use other functional components and registers for temporary data storage, provided that you support your reasoning. You are expected to provide the codes for the self-defined complex modules; however you may utilize the schematic design feature of Quartus for inter-module bus connections and wiring.

In brief, you are expected to

1. Modify or rewrite the required modules for the register file, ALP, shift register and other functional components in Verilog HDL.
2. Design and implement the instruction/data memory module with respect to the required design constraints.
3. Extract their schematics as explained in the *Laboratory_Manual.pdf* that is available on ODTU-Class.
4. Verify their operation through a test bench module if any modification is made.
5. Design and implement the entire datapath using the schematics extracted for the functional modules and provide the wiring and bus connections for proper operation

It is to be reminded that, in the succeeding laboratory work, this architecture will be completed by the addition of a control unit for automatic execution of a full multi-cycle CPU. Inherently, the ISA, the datapath and the control unit are interdependent and the complexity of the design in one may result in further limitations on the others. Therefore, it would be beneficial to figure out, even though not compulsory, how the control unit would operate and how the machine codes(binary version of the instructions) would seem, so as to have a more clarified idea of the datapath that you are to come up with. Moreover, this will also make the design for the 5TH laboratory work more straightforward.

1.2.7 Validation of Operation via TestBench (30% Credits)

As stated before, a multi-cycle architecture with fetch-decode-execute phases is to be considered for the operation. The transition between these cycles, and the data flow within the datapath are regulated by the signals generated by the control unit. Due to the absence of a designed control unit for the current laboratory work, these control signals are to be provided manually. Therefore, you are expected to write a testbench that will synchronize the proper operation where timing of the control signals are critical. Consider each three phase, namely fetch, decode and execute separately and adjust the control signals so as to mimic their sequential operation. Apart from the correct flow of data and its manipulation, the change in the functional registers is essential for the valid operation over a sequence of instructions.

In order to verify the operation the datapath,

1. Explain the required data flow within each cycle and state the control signals required to maintain the validity of operation.
2. Write a testbench module including the control signals that you have stated in the previous step.
3. Verify that your implementation is correct by providing the external signals with the proper timing that would obey the multi-cycle operation.

2 Experimental Work

As the architecture of concern is rather a complex one, the verification will be on software only, using the simulation tool ModelSim.

1. Test the correct functionality of your processor by proving the control signals and required parts of the instructions in the instruction/data memory. Ensure each instruction in the ISA is covered.
2. Verify the operation of your design by performing exemplar operations and demonstrate it to your lab instructor.

!!! Important Notice !!! When using DEES in your experiments, **DO NOT CONNECT VCC pins of DE0-Nano Board** to DEES's VCC terminal. Also, to make a common voltage reference, CONNECT GND of DE0-Nano Board to DEES's GND terminal. Please make sure that you turn all supplies OFF while making any connection throughout laboratory work.

3 Parts List

```
DE0-Nano Board
DEES
Oscilloscope
```