Mehmet Ataş & 2304020 05.06.2022

EE446 PRELIMINARY WORK #4

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

Introduction:

The goal of this laboratory study is to conceptually comprehend and design the operation of a multi-cycle CPU's datapath, build and embed it on a development board, DE0-Nano, and lastly validate its operation.

The design procedure will be based on high level specifications and constraints provided, where one must configure the instruction set and format supported by your CPU, as well as implement certain modifications on the CPU core template covered in lectures, in order to design a capable general purpose computer CPU datapath. The proposed datapath will be operationally validated by supplying control signals via a testbench.

Preliminary Work:

Datapath

– architectural state elements, memories combinational logic

– nonarchitectural state elements to hold intermediate results between the steps

Controller

– produces different signals on different steps during execution of a single instruction

– finite state machine rather than combinational logic

Replace Instruction and Data memories with a single unified memory – more realistic

Step 1: Instruction Fetch:

Read instruction in one cycle, read/write data in another cycle

New non-architectural instruction register (IR): read the instruction in IR, store it for future cycles.

IRWrite: asserted when the IR should be loaded with a new instruction.

Step 2: LDR Register Read:

Read one source (Rn) from register file into non-architectural register A

Extend the second source from the immediate field

Step 3: LDR Address:

Add base address to offset

ALUOut: Non-architectural Register

Step 4: LDR Read data from memory:

Memory address: Adr

AdrSrc selects Adr from either the PC or ALUOut

Data: Non-architectural register

Step 5: LDR Write-back to Register File:

Other instructions will need to write a result from the ALU

Multicycle Data path for LDR

Non architectural registers Instr (IR), A, ALUOut, Data to store

intermediate results

Reuse of ALU

Go over the instruction execution together with PC updates

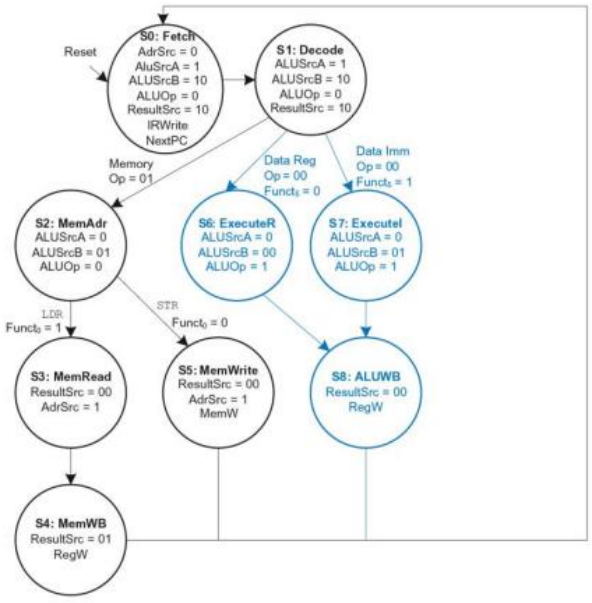
Data Processing Immediate

Read the first source from Rn and extend the second source from an 8-bit immediate.

The datapath already contains all the connections necessary

The ALU uses the ALUControl signal to determine the type of data processing instruction to execute.

The ALUFlags are sent back to the controller to update the Status register.



State	Datapath µOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset

1.2 ISA Configuration

I built the instruction set architecture (ISA) that will run on the specified CPU for this section. Because the ISA must allow general-purpose computing, it includes instructions for arithmetic-logic, memory load-store, and program control activities. There should be enough addressing modes to accommodate both basic constants and variables, as well as more complicated data structures like indexed arrays. The ISA setup, like the ISA in Laboratory Work #3, is anticipated to include the following instructions as well as possible additions of my choice.

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 • branch if zero • branch if not zero

(I couldn't do the branch operations)

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

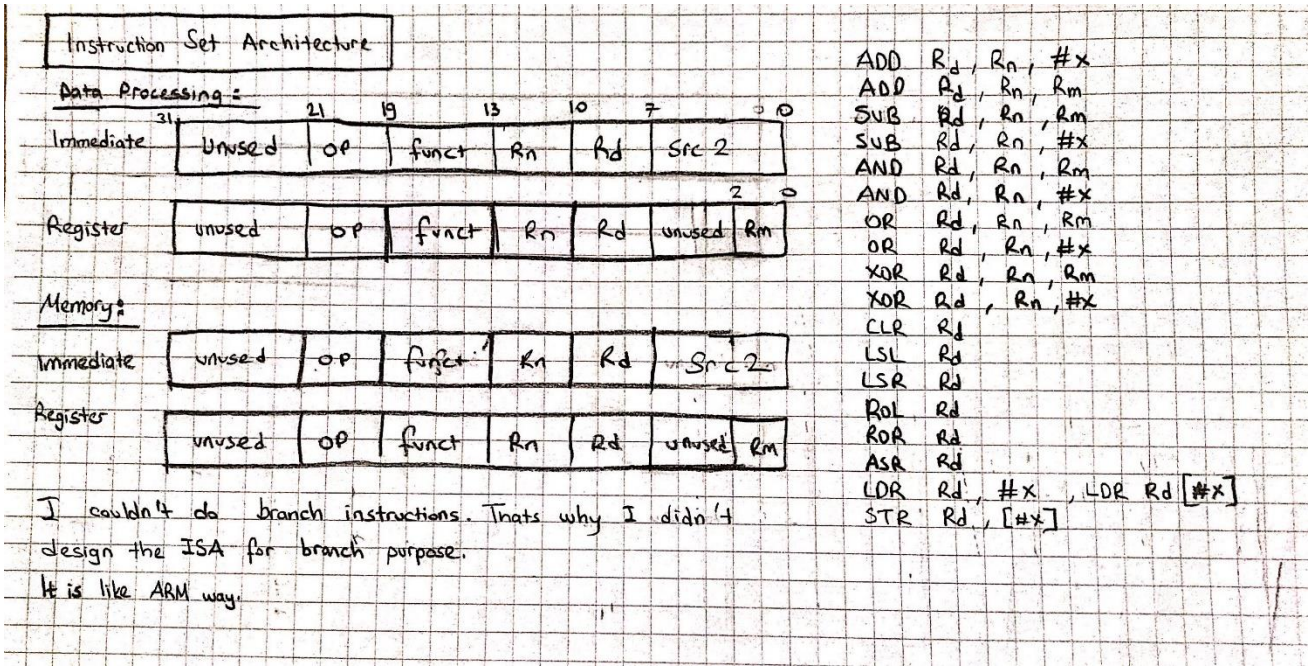


Figure 1. ISA

1.2.6 Multi-cycle CPU Datapath Design

In summary, a processor is made up of two components. The first is the datapath, which contains all of the data collection of functional units where the needed data manipulation is performed. The second, the control unit, which is outside the scope of current laboratory activity, is the entity that guides the processor's operation. The von Neumann architecture with fetch-decode-execute phases is to be studied in this multi-cycle processor implementation.

Given my ISA, I am required to construct a comprehensive datapath that will handle all of the instructions presented in this section of the laboratory work. The instructions will be read from and executed from a unified instruction/data memory, IDM. Because the operations are performed on the registers, an 8-register register file is necessary. I may utilize an enhanced version of my earlier code or rebuild it entirely. The design will have particular registers such as PC and LR in addition to the multi-purpose register. However, I was unable to incorporate these registers.

As indicated in the preceding subsection, the prior ALU can be utilized with or without modification, which may impact the control signals necessary for proper operation. Another design constraint is that I am only permitted to utilize a single arithmetic logic processor, and no wired connection between the ALU and the IDM is permitted. In addition, various functional components and registers might be used for temporary data storage. Although I am required to give the codes for the self-defined complicated modules, I may also use Quartus' schematic design capability for inter-module bus connections and wiring.

In a nutshell, I did

1. Modify or rebuild the appropriate modules in Verilog HDL for the registerfile, ALU, shift register, and other functional components.

2. Design and implement the instruction/data memory module in accordance with the design restrictions.

3. Extract their schematics as described in the LaboratoryManual.pdf on ODTUClass.

4. If any changes are made, test their functioning using a test bench module.

5. Design and build the whole datapath using the schematics retrieved from the functional modules, and supply the necessary wiring and bus connections.

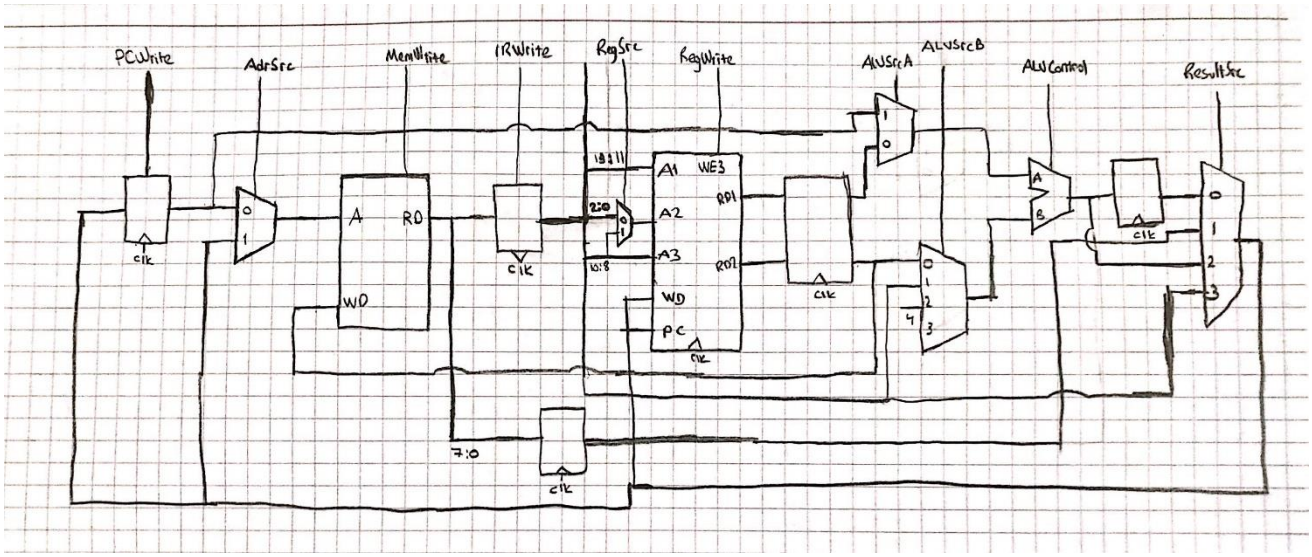


Figure 2. Datapath

1.2.7 Validation of Operation via TestBench

As previously indicated, the process would use a multi-cycle design with fetch-decode-execute stages. The control unit's signals govern the transition between these cycles and the data flow inside the datapath. These control signals must be delivered manually due to the lack of a designed control unit for the current laboratory job. As a result, I am required to develop a testbench that will synchronize the appropriate operation when control signal timing is crucial.

I examine each of the three phases independently, namely retrieve, decode, and execute, and change the control signals to simulate their sequential functioning. Aside from the right flow of data and its processing, the change in functional registers is required for the correct execution over a succession of instructions. To validate the datapath's functioning,

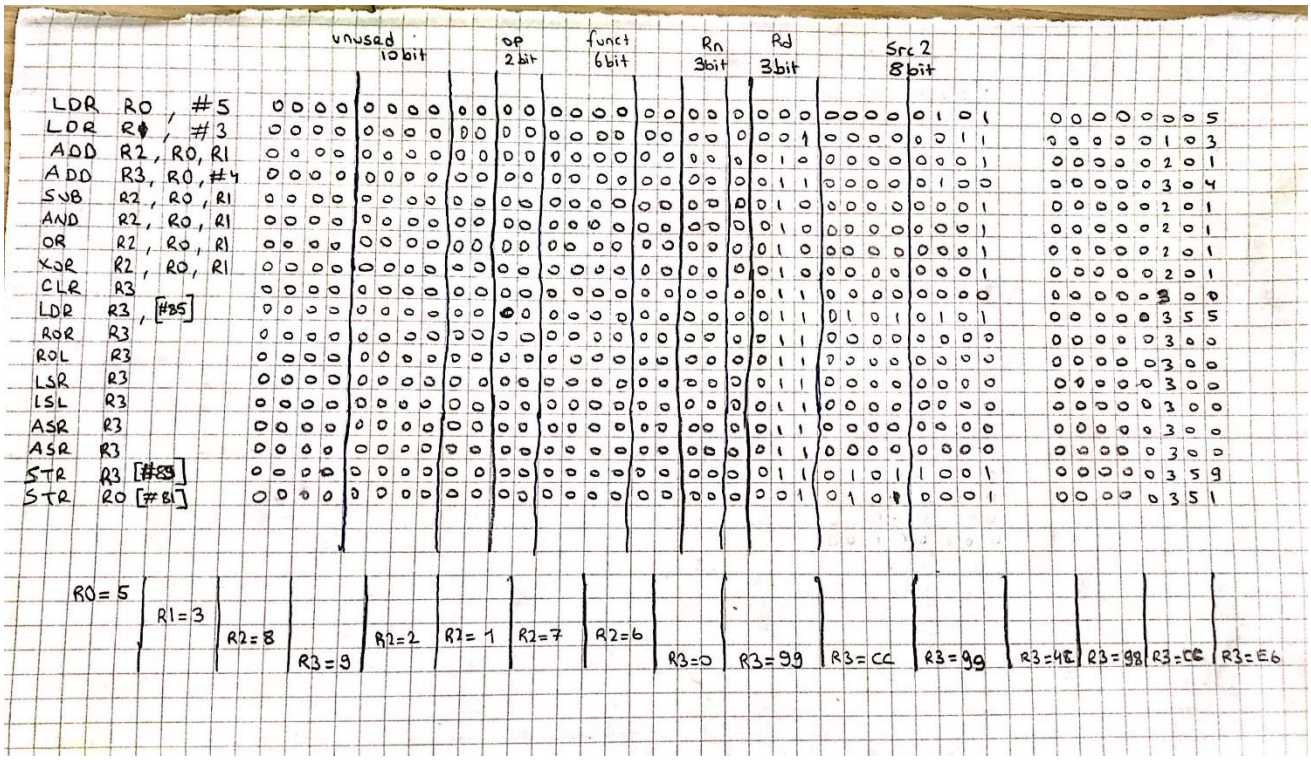


Figure 3. Example Instructions

If you look at the memory.txt file you will see the above instructions and memory.

Also, if you look at the tb_MultiCycleDatapath.v file you will see the required data flow within each cycle and state the control signals required to maintain the validity of operation.

I verify that my implementation is correct by providing the external signals with the proper timing that would obey the multi-cycle operation. As you can see from Figure 4 and Figure 5 all instructions are completed correctly.

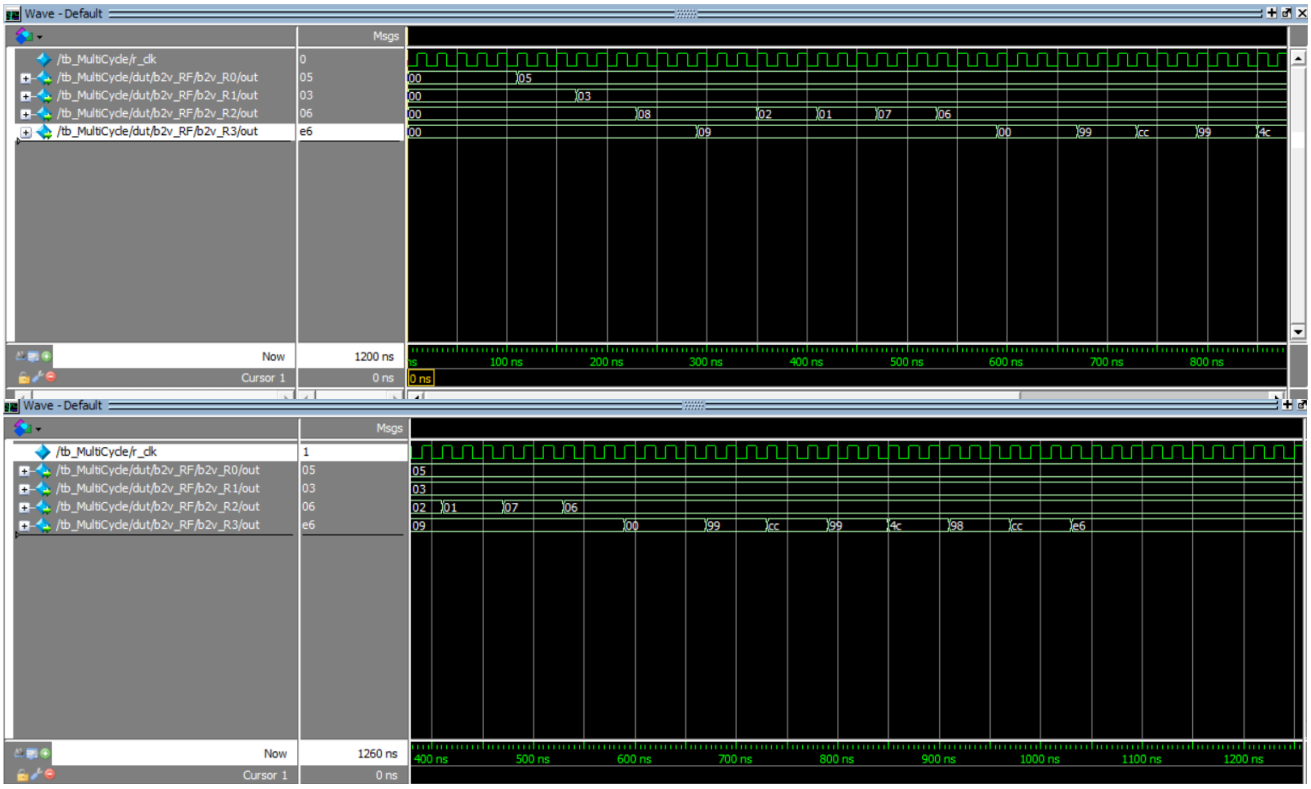


Figure 4. Testbench result for the registers

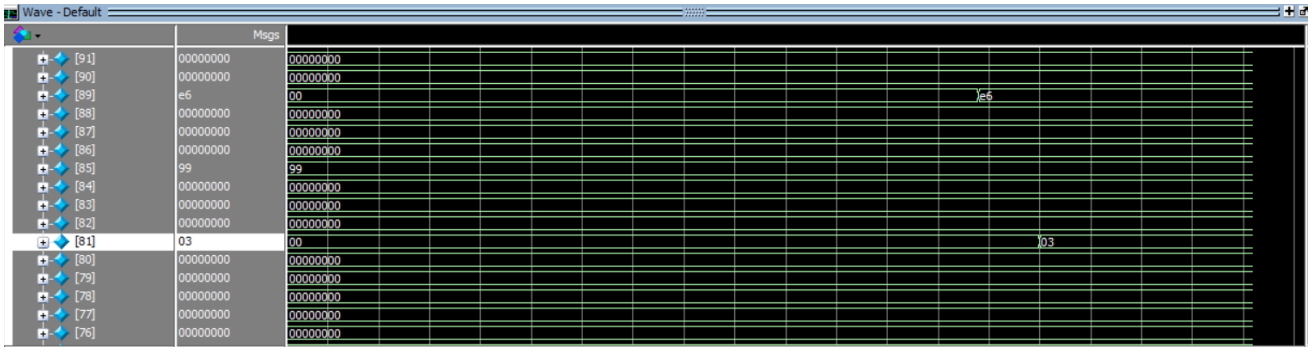


Figure 5. Testbench result for the memory

APPENDIX:

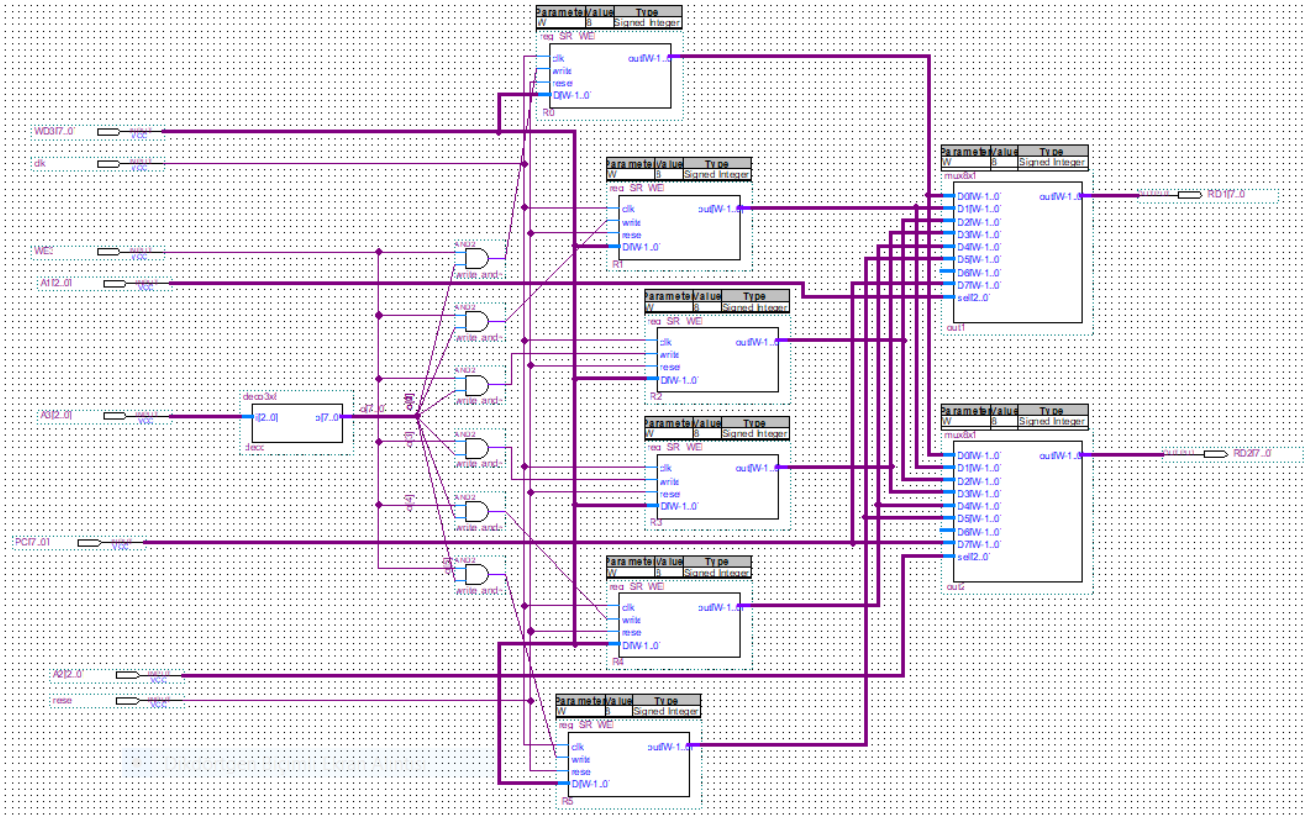


Figure 6. BDF Schematic of Register file

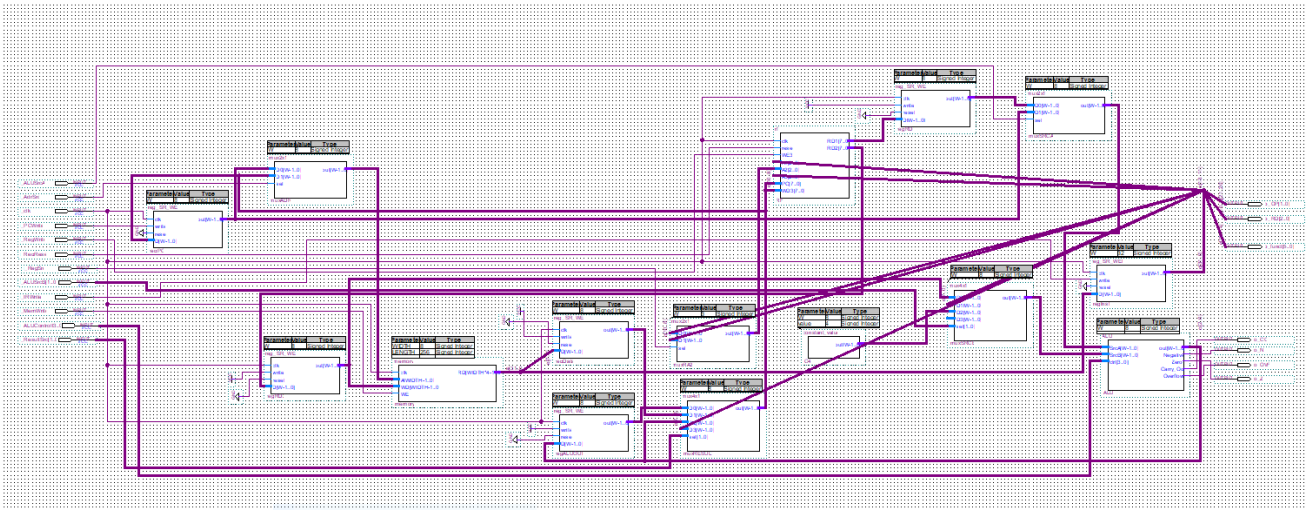


Figure 7. BDF Schematic of datapath