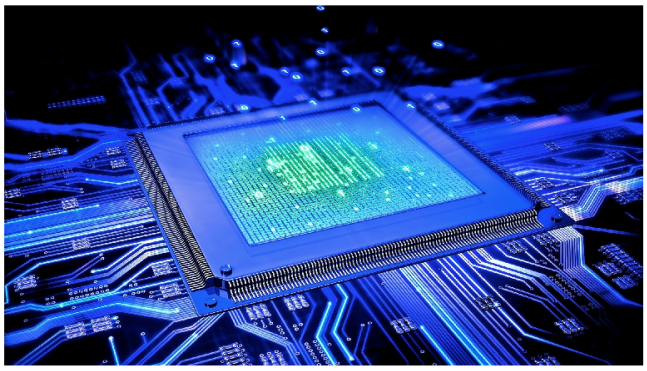
METU EE 446
Computer Architecture
Laboratory

Pipelined
Processor Design



Laboratory Work 4 - Pipelined Processor Design

Objectives

This laboratory work aims to practice the design of a 32-bit pipelined processor. You will construct a datapath and a control unit of the pipelined processor like the one discussed in class **with the hazard unit**. The designed processor will be able to execute all instructions in the instruction set.

During this laboratory work, you will improve your hard-wired controller design skills by designing the pipelined processor's controller unit, which will contain multiple stages like the datapath. Finally, you will embed your design into the FPGA of the DE1-SoC board and demonstrate your design.

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 other useful information exist, is available on the ODTUClass course page. Read that manual thoroughly. If you feel unfamiliar with pipelined CPU architecture, please refer to the corresponding **lecture notes of EE446** course.

1.2 Pipelined Processor Design with Verilog HDL (100% Credits)

For this laboratory work, you will design and implement a 32-bit pipelined processor that executes the instruction in multiple clock cycles but differs from a multi-cycle because it has an IPC of one. First, you will design its datapath and then implement the corresponding controller.

You will implement a pipelined processor very similar to the one in the lecture notes, with a few extra instructions you should be familiar with from the previous laboratories.

The processor you design will not support all ARM instructions but only a restricted set listed in Table 1. For all instructions, conditional logic of **EQ, NE, and AL** are required. Conditions codes defined in ARM standards are shown in Figure 3. You will implement a shifting functionality for the second operand for data processing.

Mnemonic	Name		Operation
ADD	Addition	add Rd,Rn,Rm	Rd← Rn + (Rm *sh* shamt5)
SUB	Subtraction	sub Rd,Rn,Rm	Rd← Rn - (Rm *sh* shamt5)
AND	Bitwise And	and Rd,Rn,Rm	Rd← Rn & (Rm *sh* shamt5)
ORR	Bitwise Or	orr Rd,Rn,Rm	Rd← Rn | (Rm *sh* shamt5)
MOV	Move to Register	mov Rd,Rm	Rd← (Rm *sh* shamt5)
MOV	Move to Register	mov Rd,rot-imm8	Rd← (imm8 *rr* rot<< 1)
CMP	Compare	cmp Rd,Rn,Rm	set the flag if (Rn - Rm =0)
STR	Store	str Rd,[Rn,imm12]	Mem[Rn + imm12] ← Rd
LDR	Load	ldr Rd,[Rn,imm12]	Rd ← Mem[Rn + imm12]
B	Branch	b imm24	PC ← (PC + 8) + (imm24<< 2)
BL	Branch with Link	bl imm24	PC ← (PC + 8) + (imm24<< 2), R14 ← PC + 4
BX	Branch and Exchange	bx Rm	PC ← Rm

Table 1: ISA to be implemented

Note: For this lab, you will use the 32-bit ARM ISA format as shown in Figure 1. You can check any resource from the web for any instructions not explained here, as we use standard ARM format.

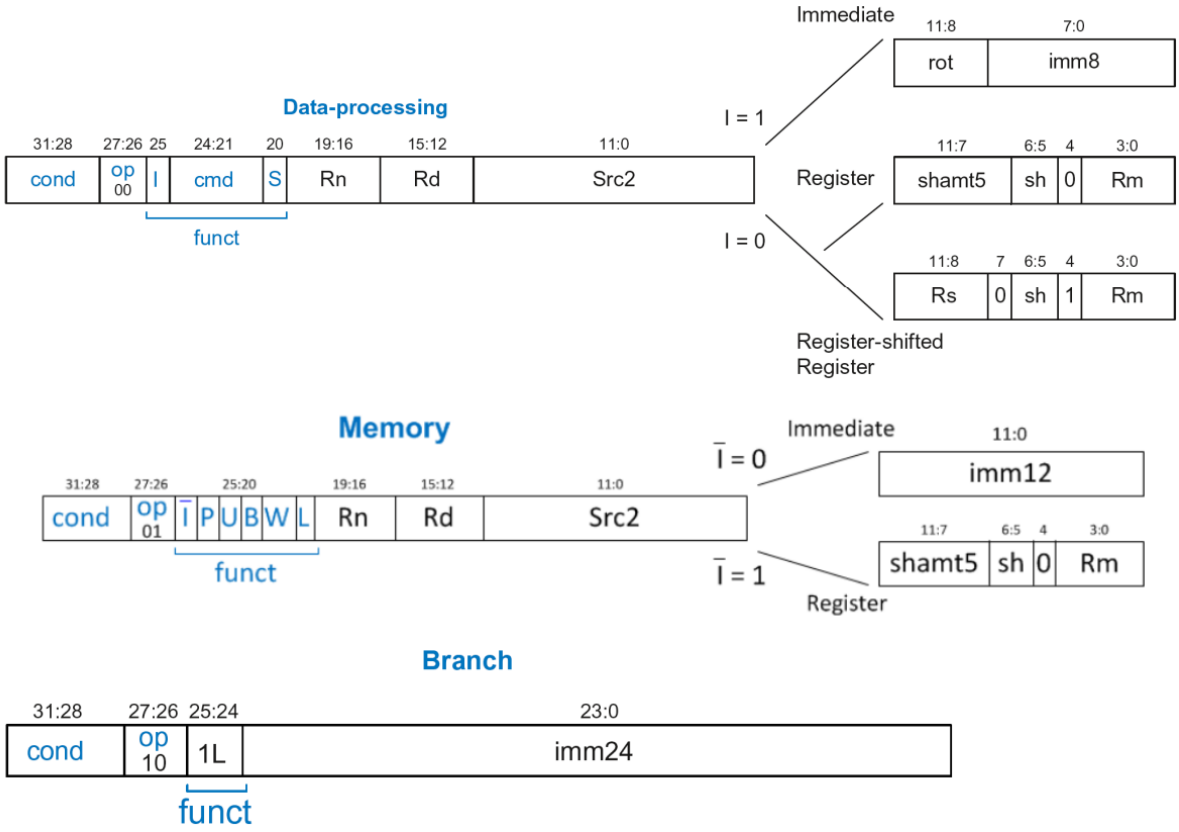


Figure 1: ARM ISA Format

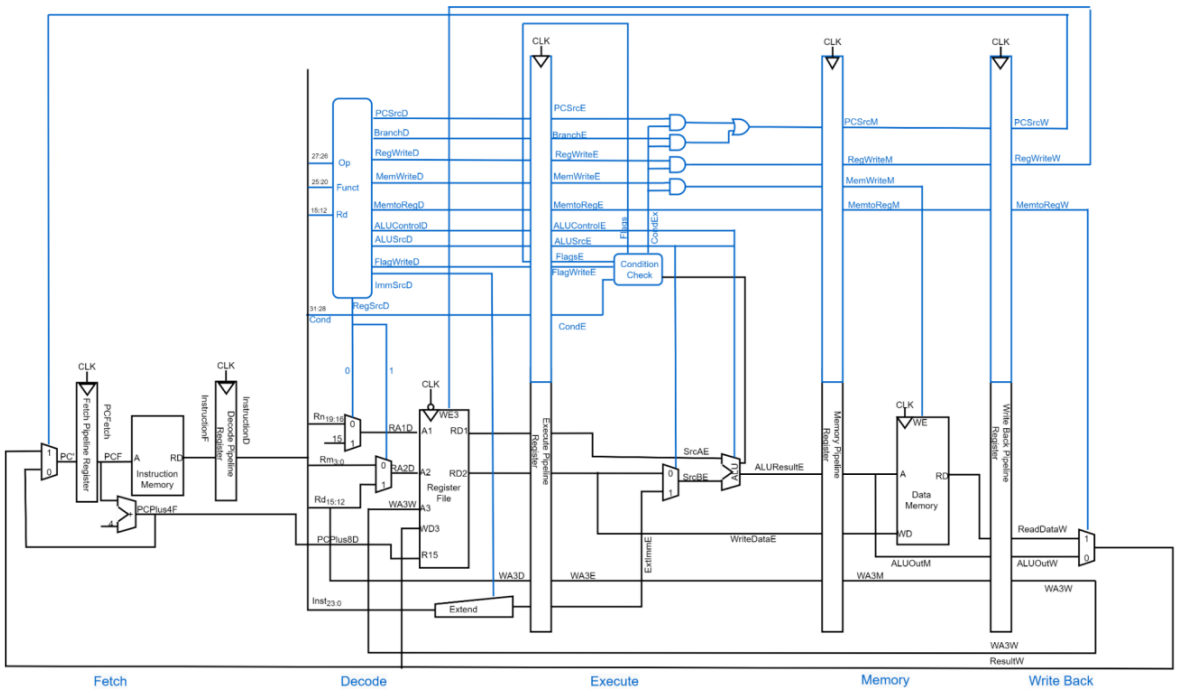


Figure 2: Pipelined processor from the lecture notes

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Figure 3: ARM Condition Codes

1.2.1 Datapath Design (20% Credits)

In this part of the laboratory work, given your ISA, you are expected to design a full datapath that would support all the instructions included in Figure 1. Using pipelined processor implementation, you will use five stages for your datapath as in lecture notes: Fetch, Decode, Execute, Memory, and Writeback. 5 Stages are important because the test bench won't work with different stages.

You will need the following components to construct the datapath, all of which are given on ODTUClass **You must use the provided modules for the testbench to work**

- Instruction Memory
- Data Memory
- Register file
- Program Counter register
- One ALU
- One Immediate Extender
- Multiplexers
- One Combinational Shifter
- Interim registers for pipelined operation
- Adders

The design will extend the datapath we discussed in the lectures. A shifter needs to be added to support data processing instructions with shift. This shifter can also be used for branches, but we built that functionality into the extender, as in lecture notes. Some modifications in the datapath connections are also needed for BL and BX instructions. ALU has a pass-through for the second operand that you can use for MOV.

Give your reasoning in the report for the changes you made to the datapath in the lecture notes (including how you used shifter and implemented the BL/BX instruction). You can change the datapath as much as you want if you give proper reasoning. As a rule of thumb, try not to needlessly forward data between stages and use the inter-stage registers as much as possible to ensure the critical path is small.

Considering the instruction set provided in Table 1, perform the following design steps:

1. (15% Credits) Using Verilog HDL, implement the Datapath using only modules and wires; no additional logic is allowed. Show the synthesized Datapaths RTL view. No I/O signal should be floating or have a constant value, as that indicates an error.
2. (5% Credits) Explain how you added the functionalities not discussed in the lecture notes. The register shifted immediate operations, both MOV operations, BL and BX.

1.2.2 Controller Design (30% Credits)

You are going to design a controller for the datapath you have designed. The pipelined computer controller will be very similar to the single-cycle controller but will forward the controller signals through the pipeline stages.

The design will extend the controller we discussed in the lectures. The shifter controller will have additional signals, and BL instruction will require you to design a new set of control signals for it. Make sure everything is consistent with your datapath.

Give your reasoning in the report for the changes you made to the controller in the lecture notes (including how you used shifter and implemented the BL instruction). As with the datapath, you can change the controller as much as you want if you give proper reasoning.

For the correct operation of the computer, you will need a **RESET** signal that terminates the operation and sets the PC to the very first slot in the instruction/data memory (active high) at the next positive clock edge. **Don't forget to reset interim registers.**

Perform the following steps:

1. (20% Credits) Using Verilog HDL, implement the Controller. There is no restriction, and you can write it however you like. Show the synthesized Controller's RTL view. No I/O signal should be floating or have a constant value, as that indicates an error.
2. (10% Credits) Explain how you added the functionalities not discussed in the lecture notes. The register shifted immediate operations, both MOV operations, BL and BX.

1.2.3 Hazard Unit (25%)

The hazard unit implemented for this project will handle two hazard types. Most of the design will be consistent with the implementation you have studied in the lectures. As a reminder, the list of the terms for hazard handling is given.

- Flush: Clearing a stage register so that the result of that stage is discarded
- Stall: Holding the value of a stage register so that a bubble can be introduced
- Forward: Sending the calculated value to a previous stage

1. (20% Credits) Using Verilog HDL, implement the Hazard Unit. There is no restriction, and you can write it however you like. Show the synthesized Hazard Unit's RTL view. No I/O signal should be floating or have a constant value, indicating an error.
2. (10% Credits) Explain how you handled the functionalities not discussed in the lecture notes. The register shifted immediate operations, both MOV operations, BL and BX.

Data Hazard Handling (15%) Data hazards happen when an instruction tries to read a register that a previous instruction has not yet written back. There can be multiple methods to handle this hazard type, even as simple as constant stalling. However, you can see that this implementation method will decrease the efficiency of your design. Hence you are required to implement your hazard unit such that:

- Hazards caused by data operations must be handled by forwarding such that no cycle is wasted.
- Hazards caused by memory operations can use a minimal amount of stalling, which is one cycle.

Control Hazard Handling (10%) Control hazards happen when the decision of what instruction to fetch next has not been made by the time the fetch takes place.

Branch operations and other operations that write to the PC (MOV R15, BX, B, BL, etc. with their conditional variants) will forward the new PC value to the fetch cycle and flush the wrong stages **when the branch is taken**. This should be implemented with a minimal amount of flushing while considering the critical path. See the lecture notes for more detailed explanations.

1.2.4 Top level for Tests (5% Credits)

Use the top-level file provided in ODTUClass to assemble the controller and datapath. This will also be used to upload your design to the DE1-SoC Board. This will make:

1. Debug register select connect to the switches and debug register output connect to one of the seven segments.
2. PC register connects to one of the seven segments.
3. You can use LEDs to connect to various hazard signals.

1.2.5 Testbench (20% Credits)

Now that you have completed the implementation of the pipelined CPU, it is required to verify its operation through some light programming. You will use the supplied testbench for this.

Don't forget to give the proper signal handles to the initialization function of the testbench class. Also, you can fill in the log_controller and log_datapath functions inside the helper library for your debugging purposes. **Do not change anything inside the TB class**

Your report must include the test bench results as a screenshot! You should submit the testbench files as well

2 Experimental Work

To upload your designs to the FPGA, you will use SystemBuilder to create a project with proper pin assignments and module initialization. DE1-SoC User manual has a short section on how the System-Builder works.

2.1 Pipelined Processor (100% Credits)

Load your processor designed in the Preliminary Work to the DE1-SoC board. Load the instructions to your instruction memory using $readmemh with the provided hex file.

Your proctoring assistant will check the design and grade you depending on how many instructions the computer can successfully execute. You can get help from the proctors, but any significant help decreases your performance grade.

You must use the supplied top-level file that will connect all the necessary signals to the board's buttons, switches, and seven-segment displays

3 Parts List

`DE1-SoC Board`