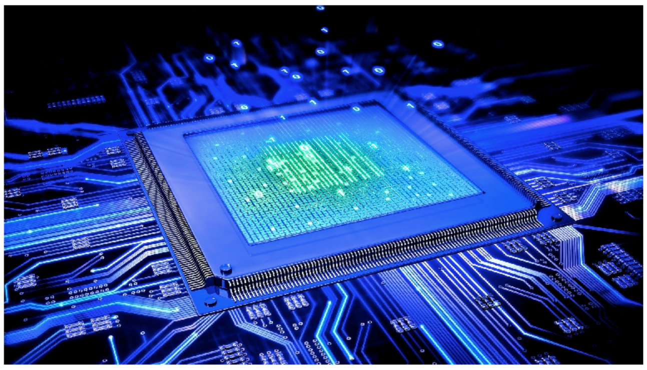
METU EE 446
Computer Architecture
Laboratory

**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 **without hazard unit and branch prediction**. 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.

Project will be an extension of this lab with at least a hazard unit and a branch predictor. Although the project will have groups, please try to understand this lab as much as possible.

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 which executes the instruction in multiple clock cycles but is different 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 will 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. Only logical shift right and left functionality is required for this lab, the ”sh” value shown in Figure 1 will be 00 for LSL and 01 for LSR.

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)
STR	Store	str Rd,[Rn,imm12]	Mem[Rn + imm12] ← Rd
LDR	Load	ldr Rd,[Rn,imm12]	Rd ← Mem[Rn + imm12]
CMP	Compare	cmp Rd,Rn,Rm	set the flag if (Rn - Rm =0)
B	Branch	b imm24	PC ← (PC + 8) + (imm24<< 2)
BEQ	Branch if Equal	beq imm24	PC ← (PC + 8) + (imm24<< 2) if flag = 1
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.

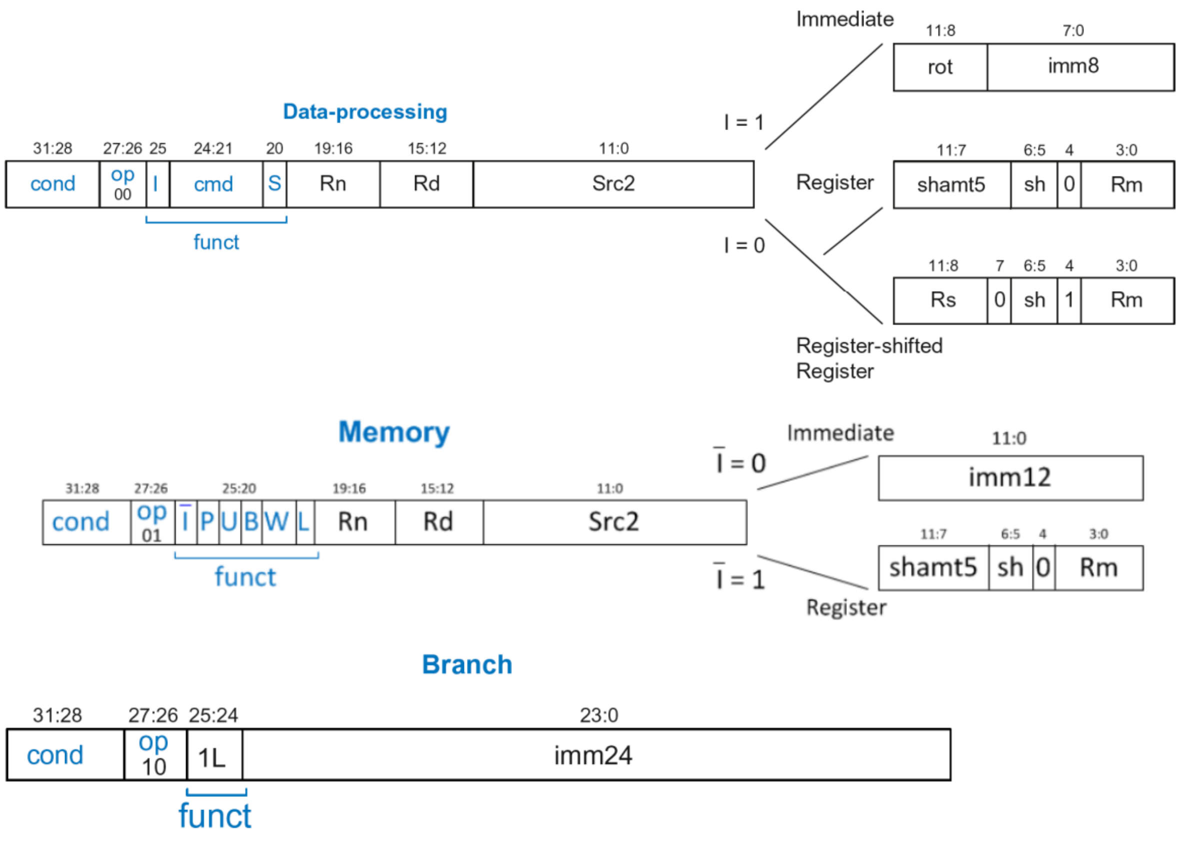


Figure 1: ARM ISA Format

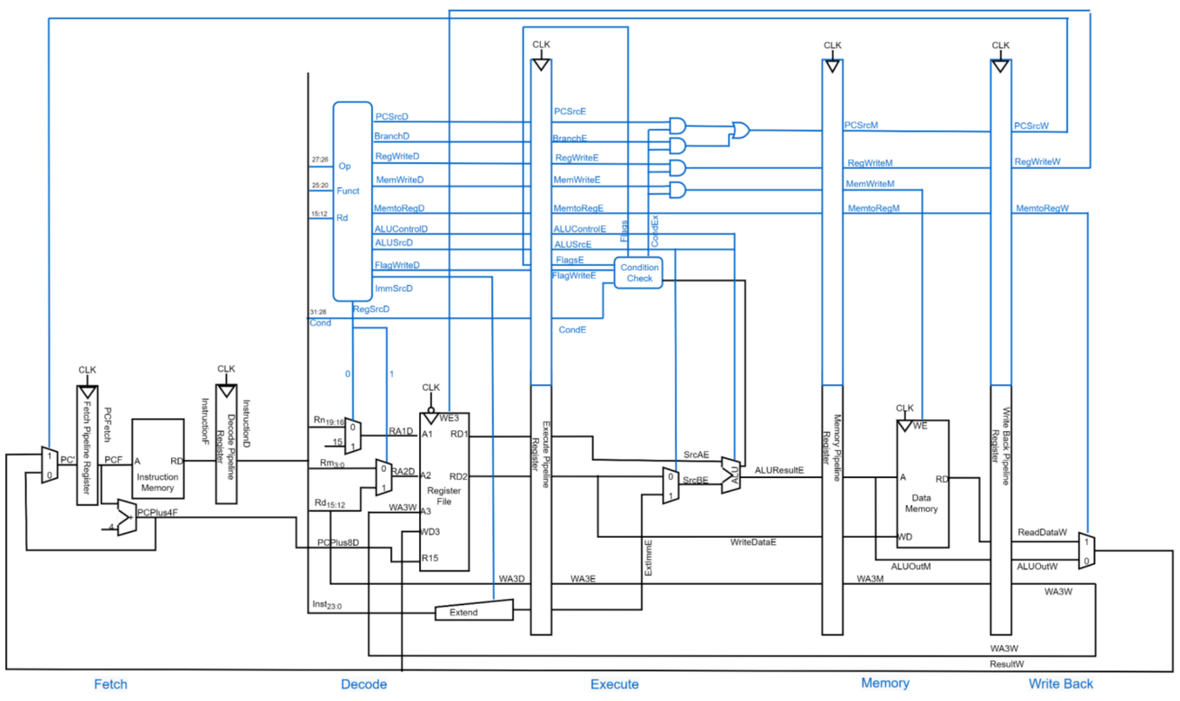
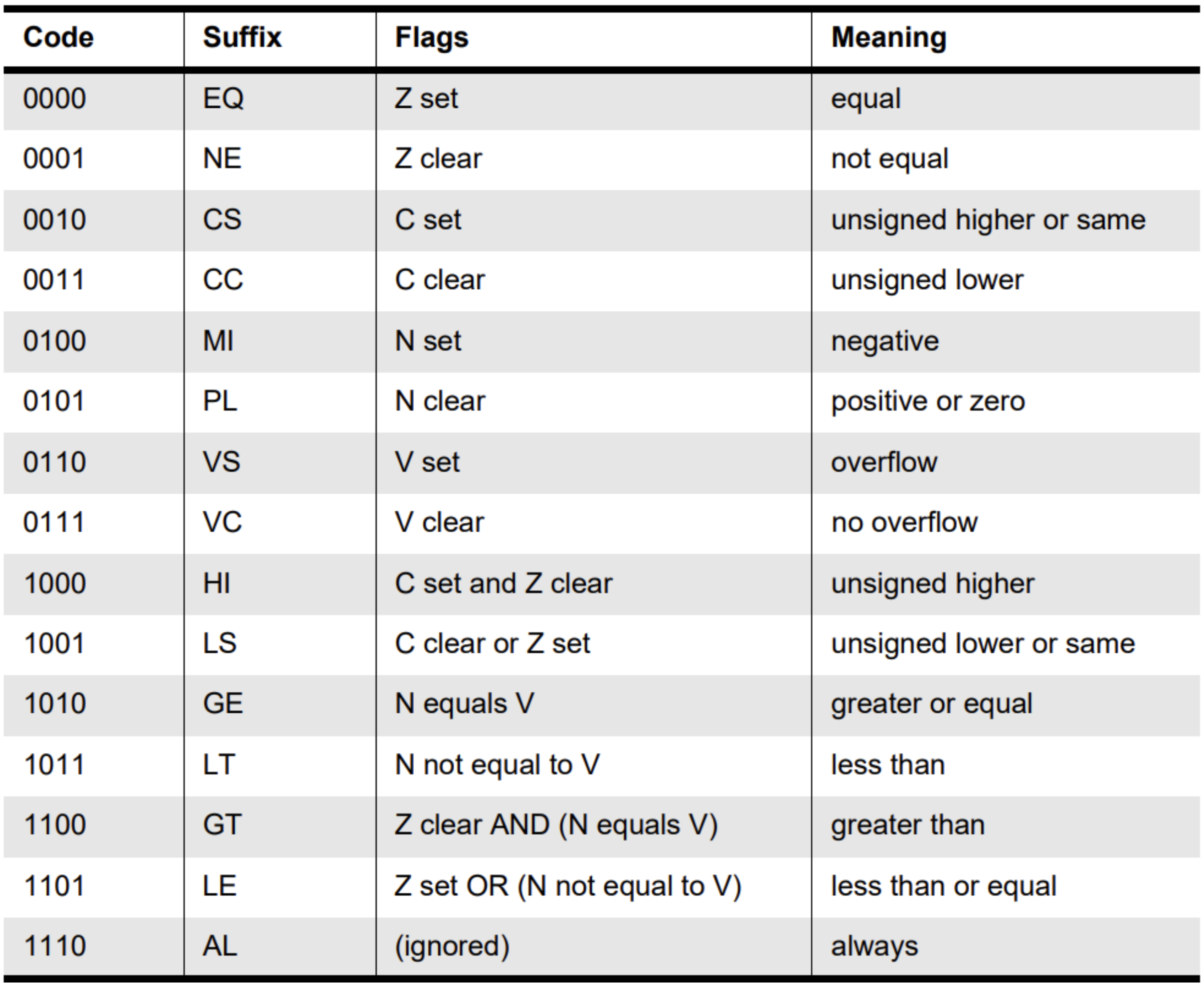


Figure 2: Pipelined processor from the lecture notes



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 (30% 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, namely Fetch, Decode, Execute, Memory, and Writeback. 5 Stages are important because, in the project, you will implement a hazard unit for this computer, which in the lecture notes is designed for five stages. **You must use the Schematic editor of Quartus to construct your datapath**.

You are allowed to use the following components to construct the datapath :

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

The design will be an extension of 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 branch, but you can also built-in that functionality to the extender as in lecture notes. Some modifications in the datapath connections are also needed for BL instruction.

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 instruction). You can change the datapath as much as you want as long as 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. (5%) Describe/explain how data processing instructions are executed in your datapath as specified in Table 1
2. (5%) Describe/explain how memory instructions are executed in your datapath as specified in Table 1
3. (5%) Describe/explain how branch instructions are executed in your datapath as specified in Table 1
4. (15% Credits) Implement the final datapath, which supports all the required instructions in **Schematic Editor of Quartus.**
   - (5% Credits) Capability of executing data processing instructions showed in Table 1
   - (5% Credits) Capability of executing memory instructions showed in Table 1
   - (5% Credits) Capability of branch instructions showed in Table 1

1.2.2 Controller Design (50% 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 be an extension of 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 as long as 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.

Perform the following steps:

1. (15% Credits) For the instructions in the CPU that you will design, show all of the control signals for each stage necessary for each instruction. To save time and space, you can group instructions (data processing, memory, branch, etc.) with similar control signals.
2. (35% Credits) Implement your controller, which supports all the required instructions, preferably in Verilog HDL. The schematic editor of Quartus is also an option.
    - (up to 10% Credits) Arithmetic and Logical operations are fully/partially implemented
    - (up to 10% Credits) Memory operations are fully/partially implemented
    - (up to 10% Credits) Branch operations are fully/partially implemented
    - (up to 5% Credits) Conditional logic is fully/partially implemented

1.2.3 Testbench (20% Credits)

Now that you have completed the implementation of the pipelined computer, it is required to verify its operation through some light programming. These programs will be some small test codes that you will write to test the full execution of a code. Along with the main codes to call these, you are supposed to:

- Write a subroutine that gets a 16-bit number and computes its 2's complement.
- Write a subroutine that computes the sum of an array of numbers and stores it in a memory location. The length of the array is initially stored in a memory location of your choice.
- Write a subroutine that gets a 16-bit number and computes its even-parity bit. This subroutine should return 0 for a number with an even number of bits with a value of 1, and it should return 1 for a number with an odd number of bits with a value of 1.

After constructing the subroutines, write a cocotb testbench that checks for the correct operation of the computer. A proper testbench is automated, so it should indicate when something fails in the design without needing any manual work. You can use prints/logs in your testbench for debug purposes.

2 Experimental Work

2.1 Pipelined Processor (100% Credits)

Load your processor designed in the Preliminary Work Part 1.2 to the DE1-SoC board. Test the correct functionality of your processor by storing all the implemented instructions in the instruction memory and verifying the correct execution of each instruction.

1. Prepare an example program that calls the subroutines described in subsubsection 1.2.3 sequentially and prepare the machine code to upload to your design.
    - **Hint 1:** If your implementation is ARM-compliant, you can use an ARM assembler to convert your assembly code into machine code.
    - **Hint 2:** The initial block is usually not synthesized; however readmemb and readmemh commands are synthesized, which you can use to upload your code to the memory). Ensure the text file you use for these commands has the same length as your memory.
2. Verify the operation of your design by embedding the design to DE1-SoC Board and running the program. Demonstrate the correct operation to your lab instructor. Your program should show the outputs of each subroutine. You will assign the clock signal to a button to run your CPU. You will be using 7-segment displays to show the signals as in the previous labs, 1-bit signals can also be shown using on-board LEDs.

3 Parts List

`DE1-SoC Board`