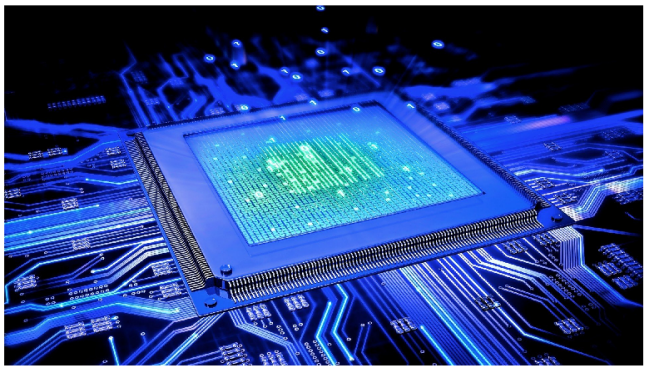
METU EE 446
Computer Architecture
Laboratory

Single Cycle
Processor Design



Laboratory Work 2 - Single Cycle Processor Design

Objectives

The purpose of this laboratory work is to practice the design of a 32-bit single-cycle processor. You will construct a datapath and control unit of the single-cycle processor like the one discussed in class. The designed processor will be able to execute all instructions given in the instruction set.

During this laboratory work, you will further improve your hard-wired controller design skills by designing the controller unit of the single-cycle processor. Finally, you will embed your design into the FPGA of the DE1-Soc board and experiment with it.

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 on the ODTUClass course page. Read that manual thoroughly. If you feel yourself unfamiliar with Verilog HDL programming and single-cycle processor design, please refer to the corresponding lecture notes of the EE446 course, which are available on the course page.

1.2 Single Cycle Processor Design with Verilog HDL (100% Credits)

For this laboratory work, you will design and implement a 32-bit single-cycle processor which executes the instruction in only one clock cycle. First, you will design its datapath then you will implement the corresponding controller.

Before starting this lab, you should be familiar with the single-cycle implementation of the processor described in the lecture slides. An example single-cycle processor schematic is shown in Figure 1. Our model of the processor divides the machine into two major units: the control and the datapath. Each unit is constructed from various functional blocks. For example, the below datapath contains the 32-bit ALU, the register file, the sign extension logic, two adders, and five multiplexers to choose appropriate operands.

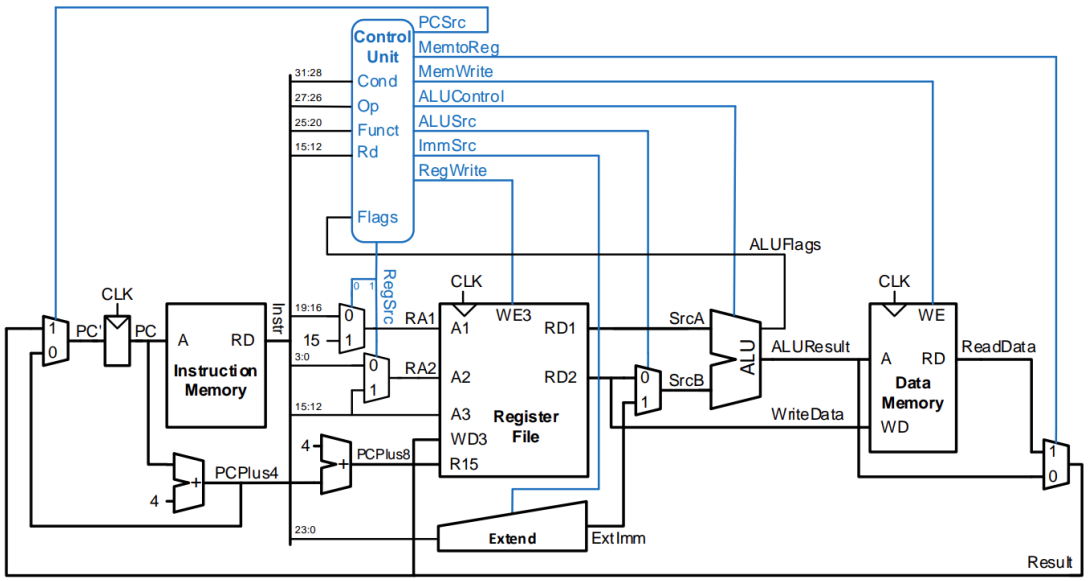


Figure 1: A complete single cycle processor implementation

The processor you will design is not going to support all ARM instructions but only a restricted set that is 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. For data processing, you will implement a shifting functionality for the second operand. Only logical shift right and left functionality is required for this lab, the "sh" value shown in Figure 2 will be 00 for LSL and 01 for LSR.

Note: You will use the 32-bit ARM ISA format as shown in Figure 2.

You will need the following components to construct the CPU.

- Storage Elements

- Instruction memory where instructions are stored
- Data memory where data is stored
- Register file
- Program Counter
- Combinational Elements
- ALU
- Adders
- Immediate Extender
- Multiplexers
- Combinational Shifter

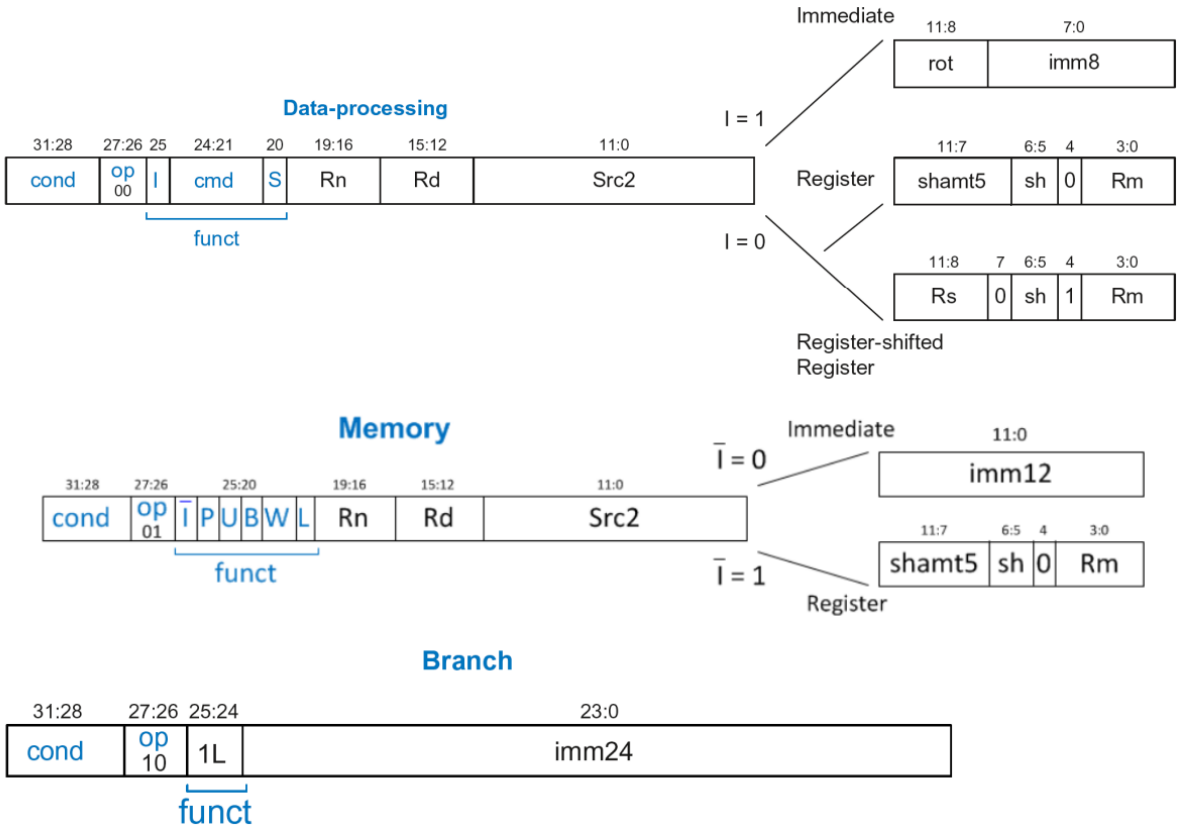


Figure 2: ARM ISA Format

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

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 ← imm24
BEQ	Branch if Equal	beq imm24	PC ← imm24 if flag = 1
BL	Branch with Link	bl imm24	PC ← imm24, R14 ← PC
BX	Branch and Exchange	bx Rm	PC ← Rm

Table 1: ISA to be implemented

1.2.1 Datapath Design (30% Credits)

You are given an example architecture in Figure 1. You will implement a datapath with modules in your library that have been constructed in the scope of the first laboratory work.

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

1. (5% Credits) State the control signal inputs of your datapath. Draw a black box diagram of your datapath by indicating the inputs and outputs.
2. (25% Credits) Implement the final datapath design which supports all the required instructions in Schematic Editor of Quartus.
   - (10% Credits) Capability of executing data processing instructions showed in Table 1
   - (8% Credits) Capability of executing memory instructions showed in Table 1, give general explanations as to how
   - (7% Credits) Capability of branch instructions showed in Table 1, give general explanations as to how

1.2.2 Controller Design (50% Credits)

In this step, the controller for the single-cycle processor is to be designed. It will look like the controller which is in the lecture slides (Figure 4).

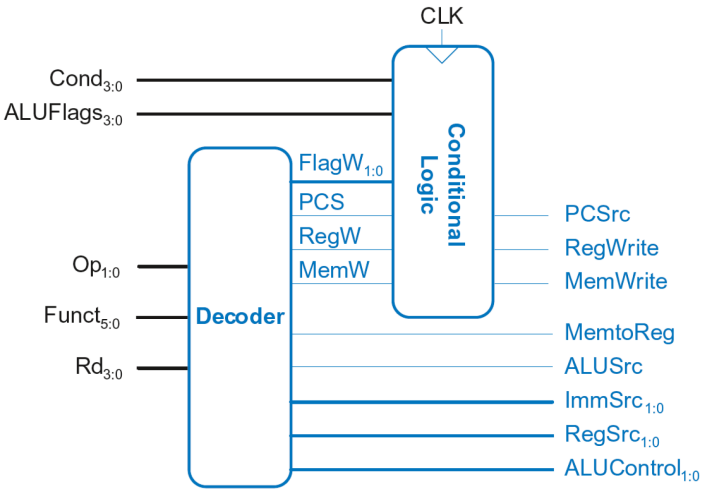


Figure 4: Controller

Perform the following steps:

1. (20% Credits) For the instructions in the CPU that you are going to design, show all of the changes in the control signals while adding each instruction. List all the steps that are needed for the execution of each instructions.
2. (30% Credits) Implement your controller which supports all the required instructions in Verilog HDL.
   - (up to 10% Credits) Arithmetic and Logical operations are fully/partially implemented
   - (up to 5% 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 multi-cycle CPU, 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.

For the above subroutines and/or others of your choice that test sufficiently many instructions, write an automated testbench with cocotb which will validate your implementation of the CPU.

Cocotb can access all of the signals in your design so specifying an output is not necessary but register file outputs together with PC are usually assigned as outputs.

2 Experimental Work

2.1 Single Cycle 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 a example program which calls the subroutines described in subsubsection 1.2.3 sequentially and prepare the machine code to upload on your CPU 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)
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 are going to assign the clock signal to a button to run your CPU. You will be using 7-segment displays to show the signals as the previous labs, 1-bit signals can also be shown using on-board LEDs.

3 Parts List

`DE1-SoC Board`