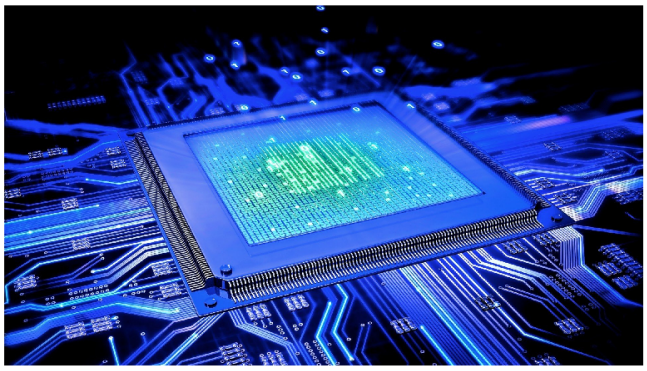
METU EE 446
Computer Architecture
Laboratory

Single Cycle
Processor Design



Laboratory Work 2 - Single Cycle Processor Design

Objectives

This laboratory work aims 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 in the given restricted 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 Nexys A7 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 other useful information exist, is available on the ODTUClass course page. Read that manual thoroughly. If you feel unfamiliar with Verilog HDL programming and single-cycle processor design, please refer to the corresponding lecture notes of the EE445 and EE446 courses, 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 that executes the instruction in only one clock cycle. First, you will design its datapath and then 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.

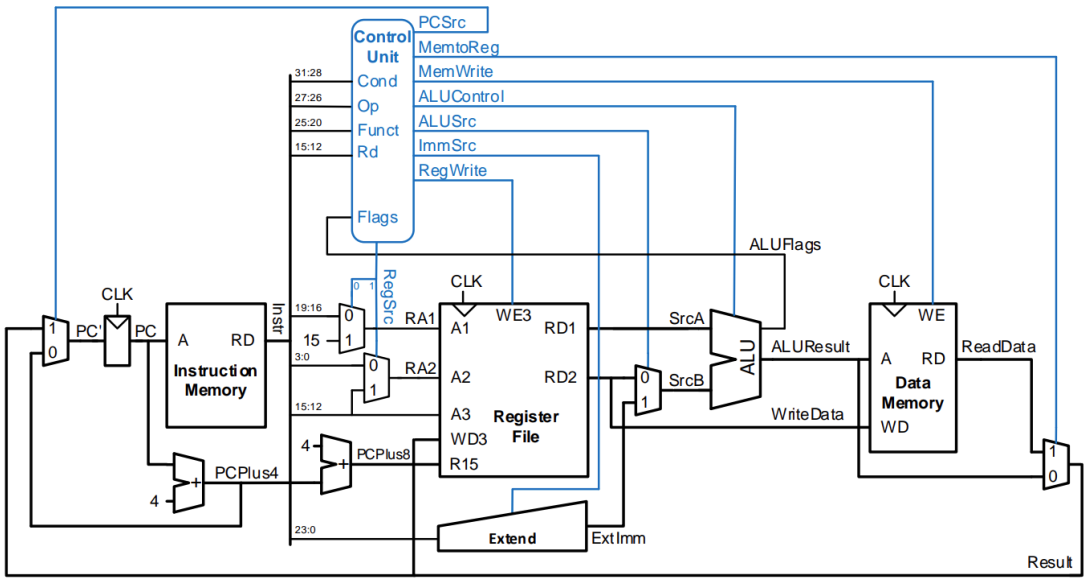


Figure 1: A complete single cycle processor implementation

The processor you 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. Thus, you only need the "Zero" flag. Condition codes defined in ARM standards are shown in Figure 3. You will implement a shifting functionality for the second operand for data processing.

Note: LSL=2'b00, LSR=2'b01, ASR=2'b10, RR=2'b11;

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

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 where instructions are stored
- Data memory where data is stored
- Register file
- Registers
- Seven Segment Display Converter
- Debouncer
- ALU
- Adders
- Immediate Extender
- Multiplexers
- Combinational Shifter

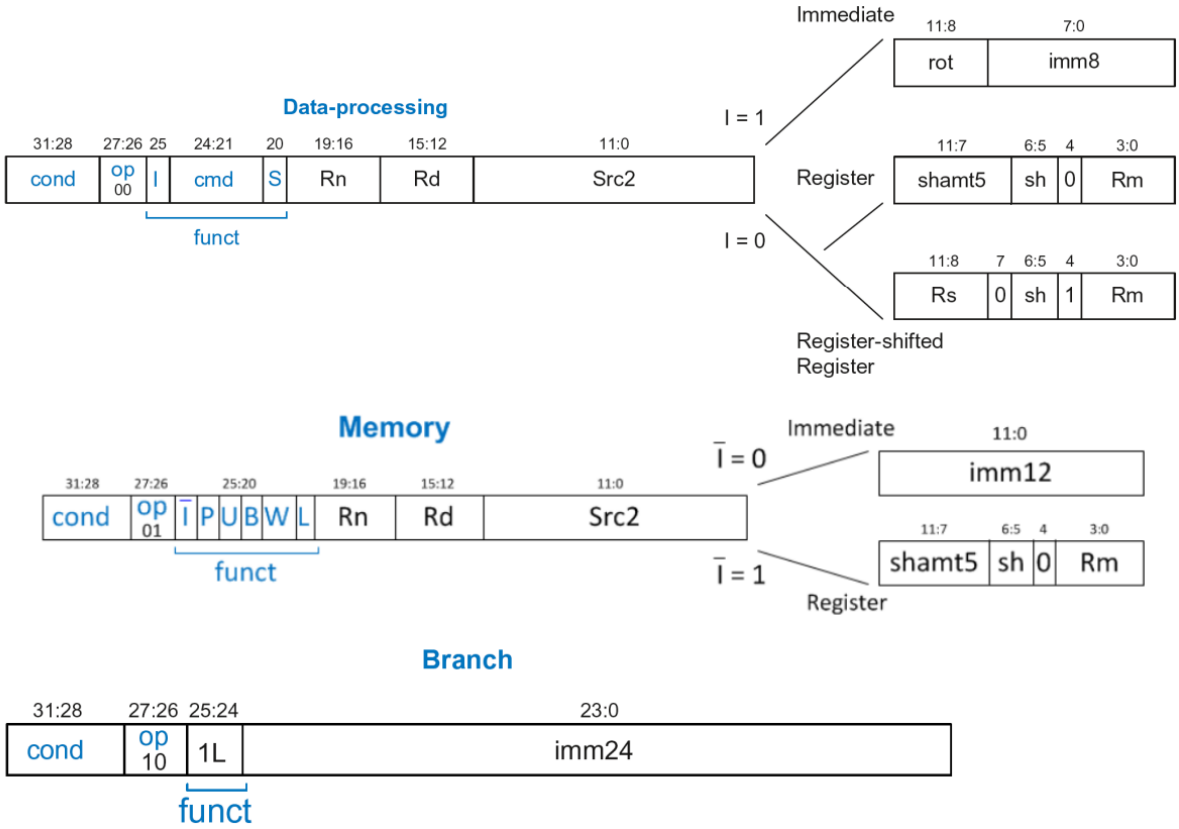


Figure 2: ARM ISA Format

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

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)

You are given an example architecture in Figure 1. The building block modules are shared through Odtuclass. You will implement a datapath with these modules. **You must use the provided files instead of the ones you designed in first laboratory work.**

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-in that functionality to 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.

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

1. (20% Credits) Using Verilog HDL, implement the Datapath using only modules and wires; no additional logic is allowed. In your report, show the synthesized Datapath's RTL Schematic view. No I/O signal should be floating or have a constant value, as that indicates an error.
2. (10% Credits) In your report, explain how you added the functionalities not discussed in the lecture notes. The register shifted immediate operations, both MOV operations, BL and BX.

For the operation of the computer, a certain external signal, namely **RESET**, is also necessary. Reset resets the PC register at the next positive clock edge.

NOTE: To obtain synthesized RTL schematic in Vivado, locate the Flow Navigator in the left of your

screen. Then under RTL ANALYSIS, press Run Linter. Then expand Open Elaborated Design, and at the bottom, find Schematic and click on it. This will show the overall schematic, you need to find the module you want to show and expand it by clicking the plus on its top-left corner.

1.2.2 Controller Design (40% Credits)

In this step, the controller for the single-cycle processor is to be designed. It will look like the controller in the lecture slides (Figure 4) with support for new instructions and addressing modes. **Note that, controller should not be inside the datapath module. The top module should contain datapath and controller instances.**

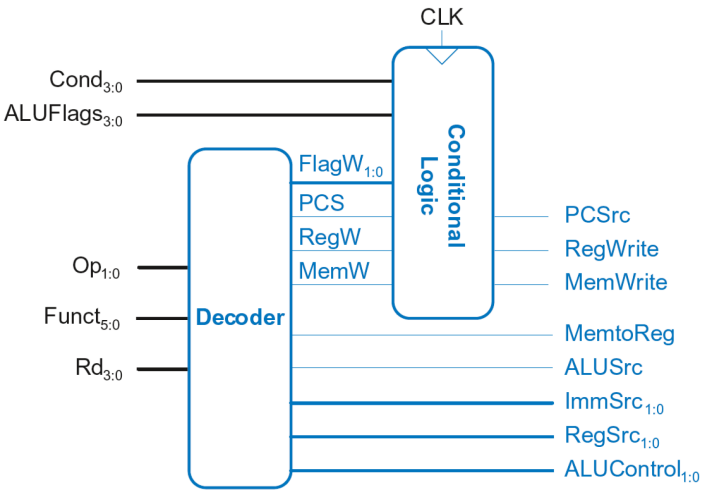


Figure 4: Controller

Perform the following steps:

1. (30% Credits) Using Verilog HDL, implement the Controller. There is no restriction, and you can write it however you like. In your report, show the synthesized Controller RTL view. No I/O signal should be floating or have a constant value, indicating an error.
2. (10% Credits) In your report, 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 Top level for Tests (10% Credits)

Connect your Datapath and Controller in another module. This top module must have clock, reset and debug register select inputs. It must have outputs which are program counter and the value of the register being debugged.

1.2.4 Testbench (20% Credits)

Now that you have completed the implementation of the single-cycle CPU, it is required to verify its operation through some light programming. You will use the supplied testbench and Instructions.hex file for this. **If the computer cannot execute at least the MOV immediate instruction in the testbench, you will not be admitted to the lab.**

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!

2 Experimental Work

To upload your designs to the FPGA, you will use Vivado to create a project, with proper pin assignments using the master XDC file, and module initialization.

2.1 Single Cycle Processor (100% Credits)

Load your processor designed in the Preliminary Work Part 1.2 to the Nexys A7 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 major help decreases your performance grade.

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

3 Parts List

`Nexys A7 100T Board`