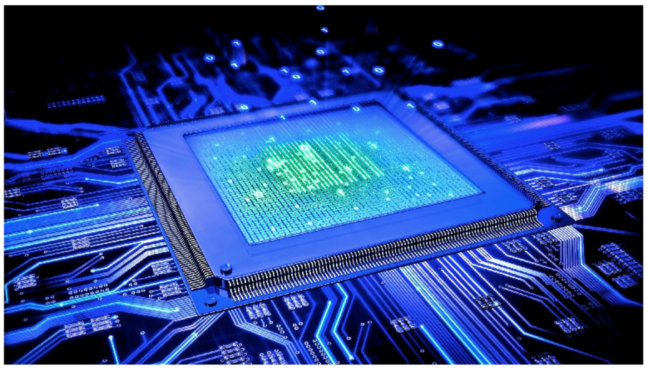
METU EE 446
Computer Architecture
Laboratory

**Multi Cycle
Processor Design**



Laboratory Work 3 - Multi Cycle Processor Design

Objectives

This laboratory work aims to practice the design of a 32-bit multi-cycle processor. You will construct a datapath and a control unit of the multi-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 multi-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 other useful information exist, is available on the ODTUClass course page. Read that manual thoroughly. If you feel unfamiliar with Verilog HDL programming and multi-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 Multi Cycle Processor Design with Verilog HDL (100% Credits)

For this laboratory work, you will design and implement a 32-bit multi-cycle processor that executes the instruction in multiple clock cycles. First, you will design its datapath and then implement the corresponding controller.

Before starting this lab, you should be familiar with the multi-cycle implementation of the processor described in lecture slides. The multi-cycle from the lecture notes is given in Figure 2. You will implement a multi-cycle processor very similar to the one in the lecture notes with a few extra instructions you should be familiar with from the previous laboratory and some design freedom.

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.

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

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

- One Instruction and Data memory (IDM)
- One Register file
- One Program Counter register
- One ALU
- One Immediate Extender
- One Combinational Shifter
- Multiplexers

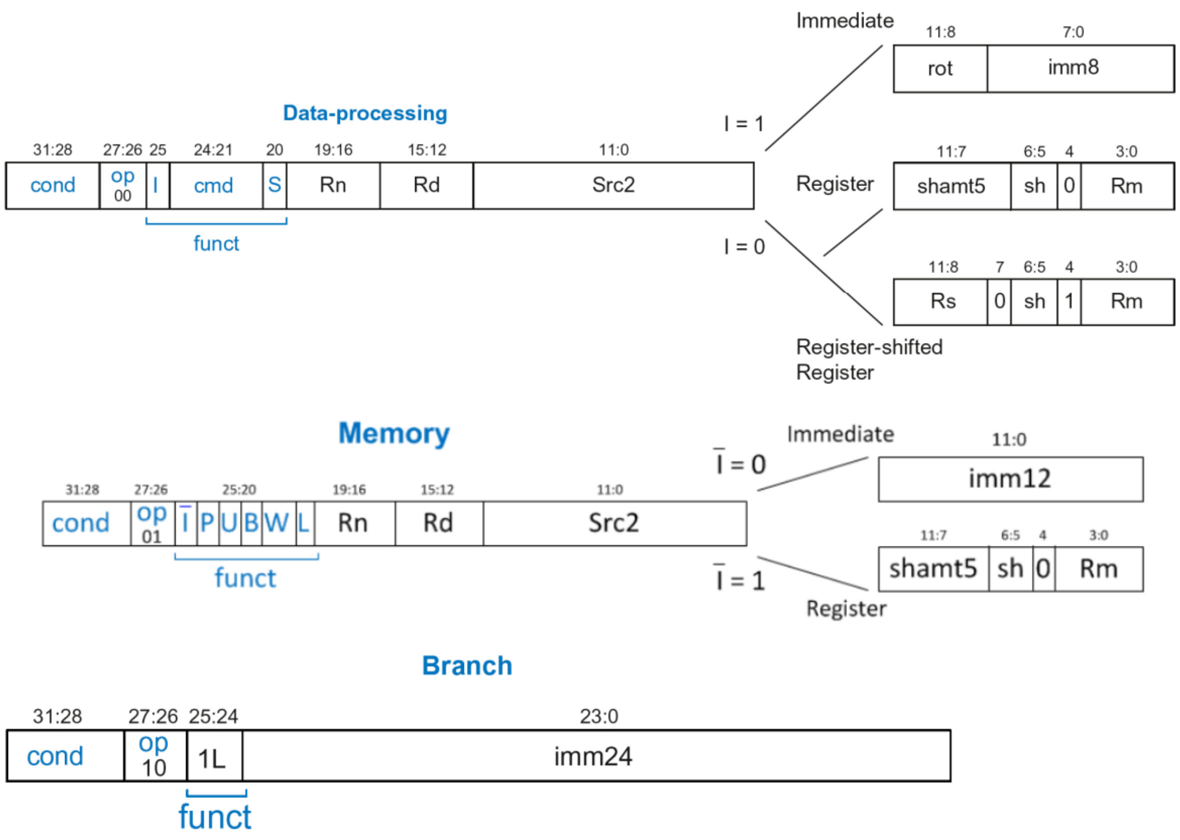


Figure 1: ARM ISA Format

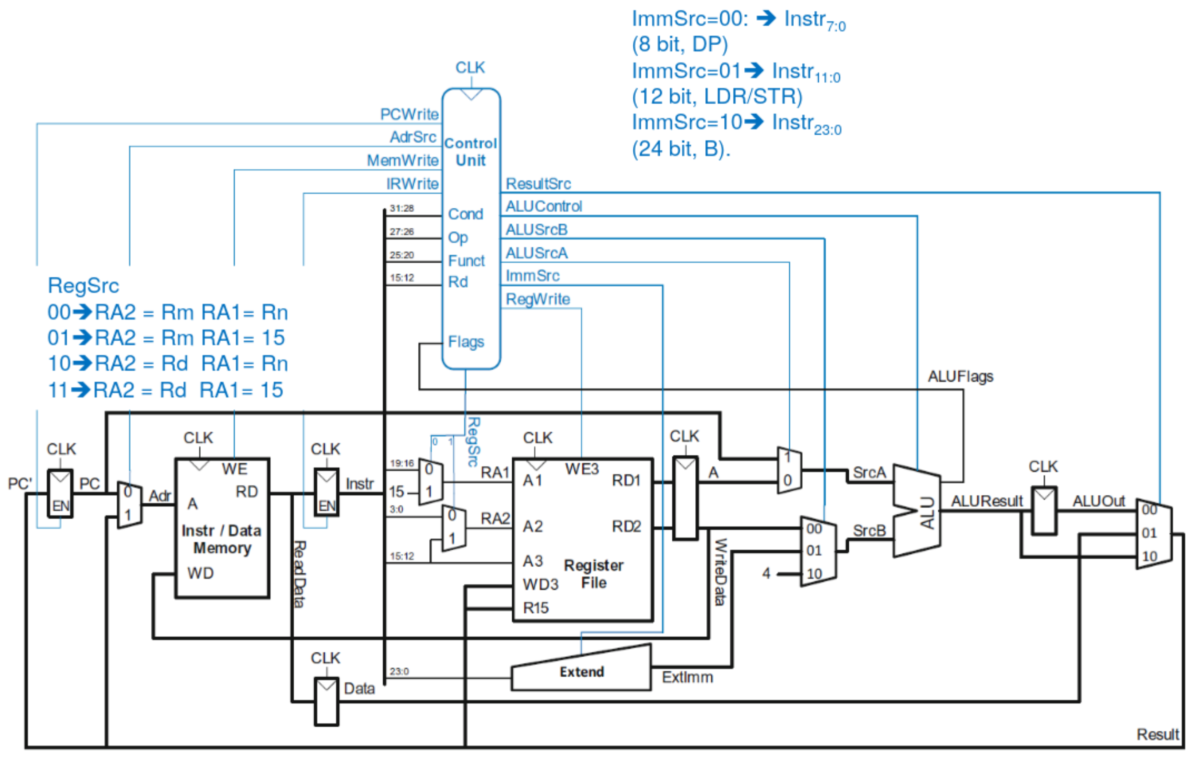


Figure 2: Multi cycle processor from the lecture notes

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 to support all the instructions in the Table 1. The instructions will be stored in a unified instruction/data memory, IDM, read from and executed. You use the memory module designed in the first lab as the IDM, also given on ODTUClass.

As stated in the previous subsection, the previous ALU can be used, without modification to operations, and control signal meanings can change (which would inherently affect the control signals required for the proper operation), although there should not be any need for it. As another design limitation, **you can use only a single arithmetic logic processor, and no wired connection between the ALU and the IDM is allowed**. Besides, you may as well use other functional components and registers for temporary data storage, provided that you support your reasoning. You are not allowed to design new modules like in the previous laboratory.,

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. Show the synthesized Datapaths RTL view. No I/O signal should be floating or have a constant value, indicating 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.

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.

1.2.2 Controller Design (40% Credits)

The design of the control unit may be considered as designing the FSM (finite-state machine) that will generate the control sequence in the correct conditional and sequential order concerning the cycles described in the lecture notes.

- **C0: Fetch Cycle:** This is the first cycle corresponding to the operation of a single instruction. The instruction is read from the instruction/data memory to be loaded to an instruction register that holds the current one. Meanwhile, the program counter (PC) is increased to point to the next instruction.
- **C1: Decode Cycle:** Within the decode cycle, the current instruction in the instruction register is decoded to obtain the conditions and the operands.
- **C2: Execute and Branches Cycle:** In this cycle, the data is processed using the ALU or branch is taken.
- **C3: MemWrite/MemRead and ALU Writeback:** In this cycle, processed data is written back to the register file or to the memory.
- **C4: Memory Writeback:** In this cycle, data read from memory is written back to the register file.

For the supplied test bench to work, you must ensure your instruction types take the same number of cycles as in the lecture notes described in Figure 4. You are heavily encouraged to follow the lecture notes.

- Data Processing and Store instructions: 4 cycles
- Branch Instructions (BL and BX included): 3 cycles

- Load instructions: 5 cycles

Regarding the above-given descriptions, with support for new instructions and addressing modes, you must determine which control signal in your design is to be utilized in which cycle.

For the operation of the FSM, a certain external signal, namely **RESET** is also necessary.

- **RESET(active high):** Terminates the operation and sets the FSM to the first state at the next positive clock edge.

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. 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.

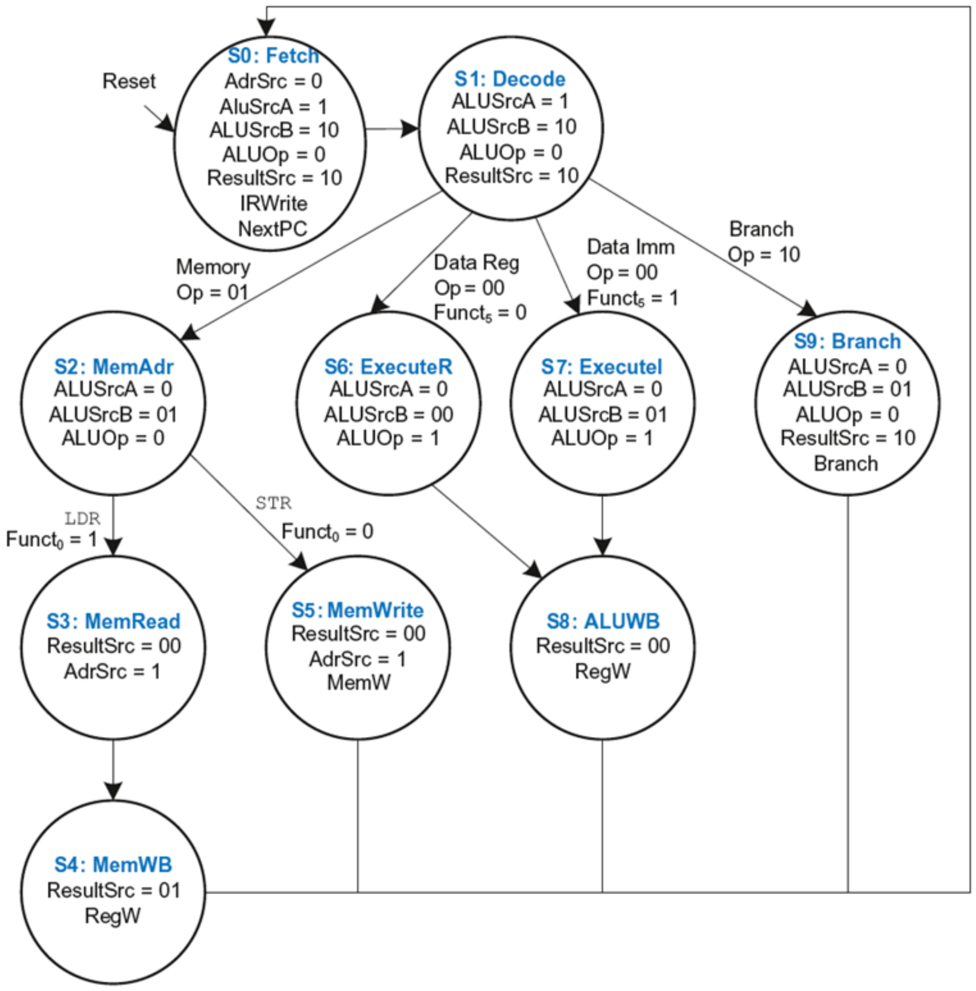


Figure 4: FSM of the Controller from Lecture Notes

1.2.3 Top level for Tests (10% 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 connects to the switches and debug register output connects to 4 seven segments.
2. PC register connects to 2 seven segments.
3. FSM's state connects to LEDs for ease of debugging.

1.2.4 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. You will use the supplied testbench. **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**

Note that your design will fail the testbench if the number of clock cycles for each instruction does not match the specification given in subsubsection 1.2.2. Your report must include the test bench results as a screenshot!

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 SystemBuilder works.

2.1 Multi Cycle Processor (100% Credits)

Load your processor designed in the Preliminary Work Part 1.2 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 major 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`