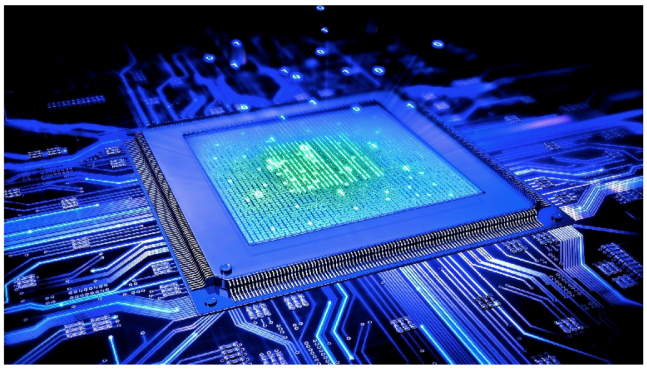
METU EE 446
Computer Architecture
Laboratory

Multi Cycle
Processor Design



Laboratory Work 3 - Multi Cycle Processor Design

Objectives

The purpose of this laboratory work is 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 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 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 some other useful information exist is available at the ODTUClass course page. Read that manual thoroughly. If you feel unfamiliar with multi-cycle CPU architecture, please refer to the corresponding **lecture notes of EE446** course

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 which executes the instruction in multiple clock cycles. First, you will design its datapath then you will 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 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 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
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 are allowed to use the following components to construct the CPU:

- Instruction and Data memory (IDM)
- Register file
- Program Counter register
- ALU
- Immediate Extender
- Multiplexers
- Combinational Shifter
- Intermediate registers for multi cycle operation

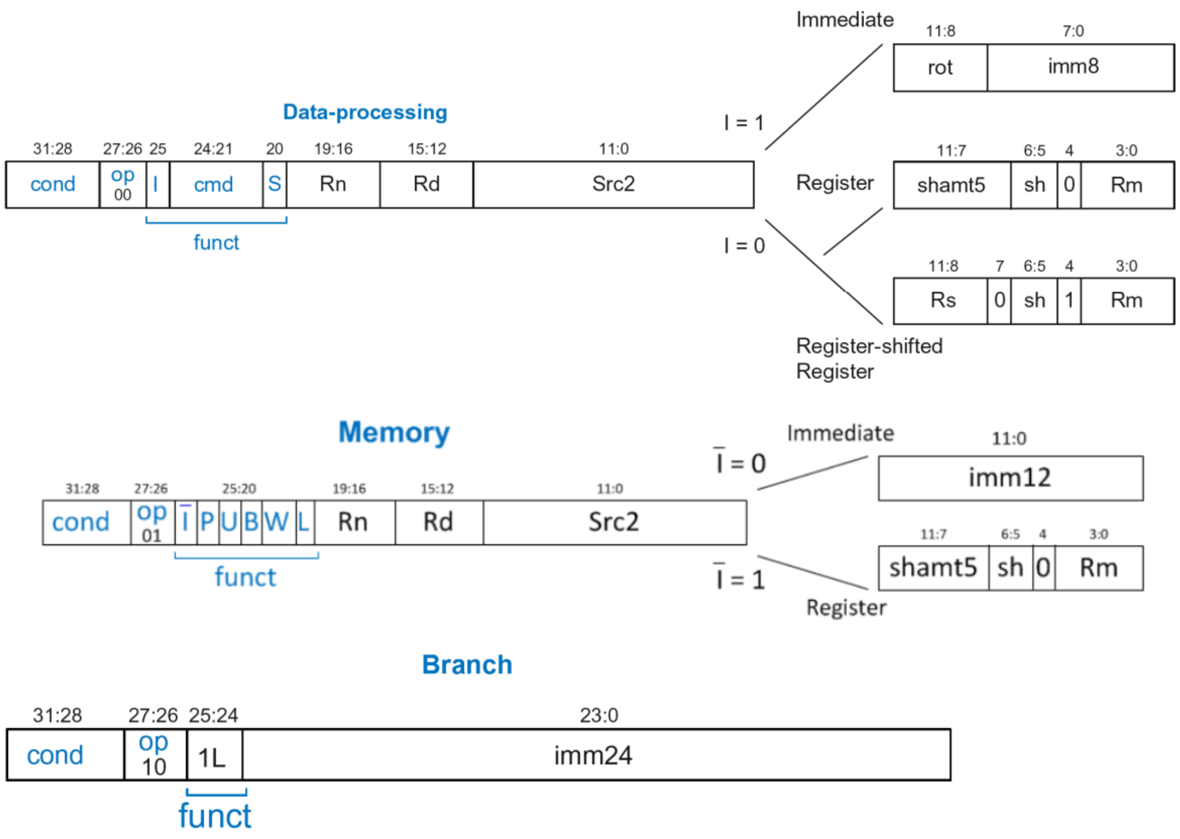


Figure 1: ARM ISA Format

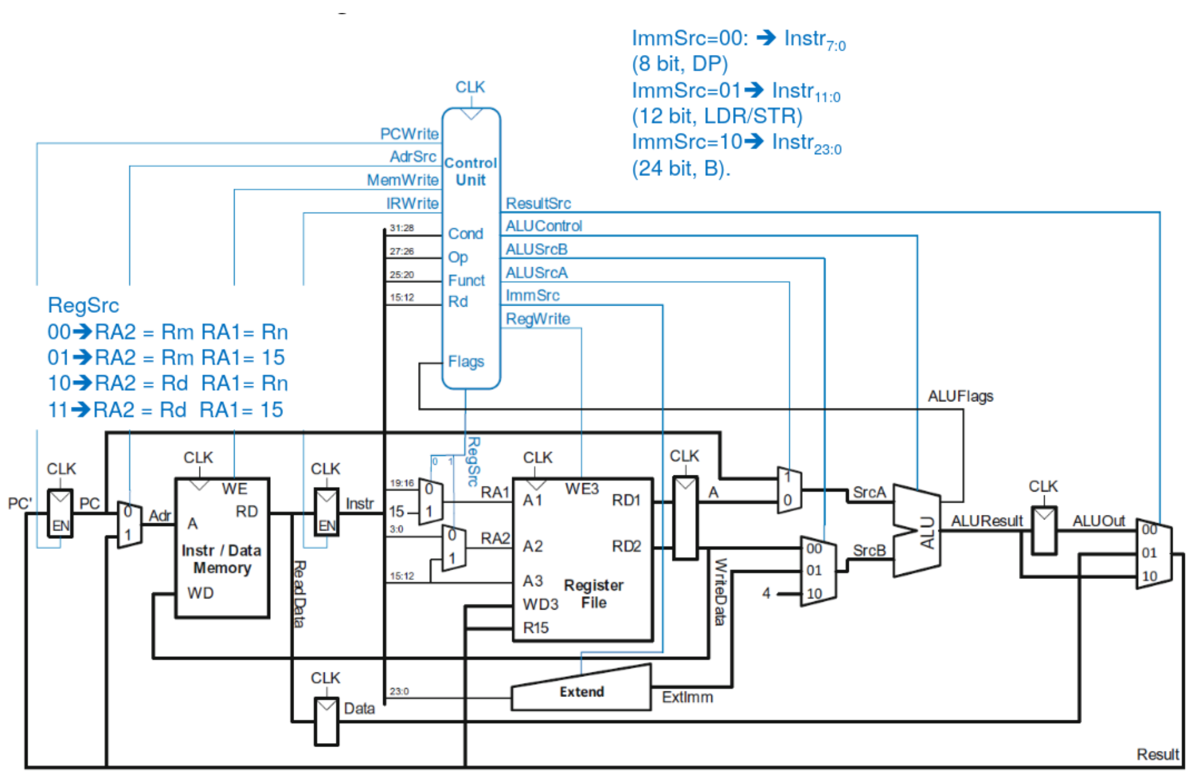
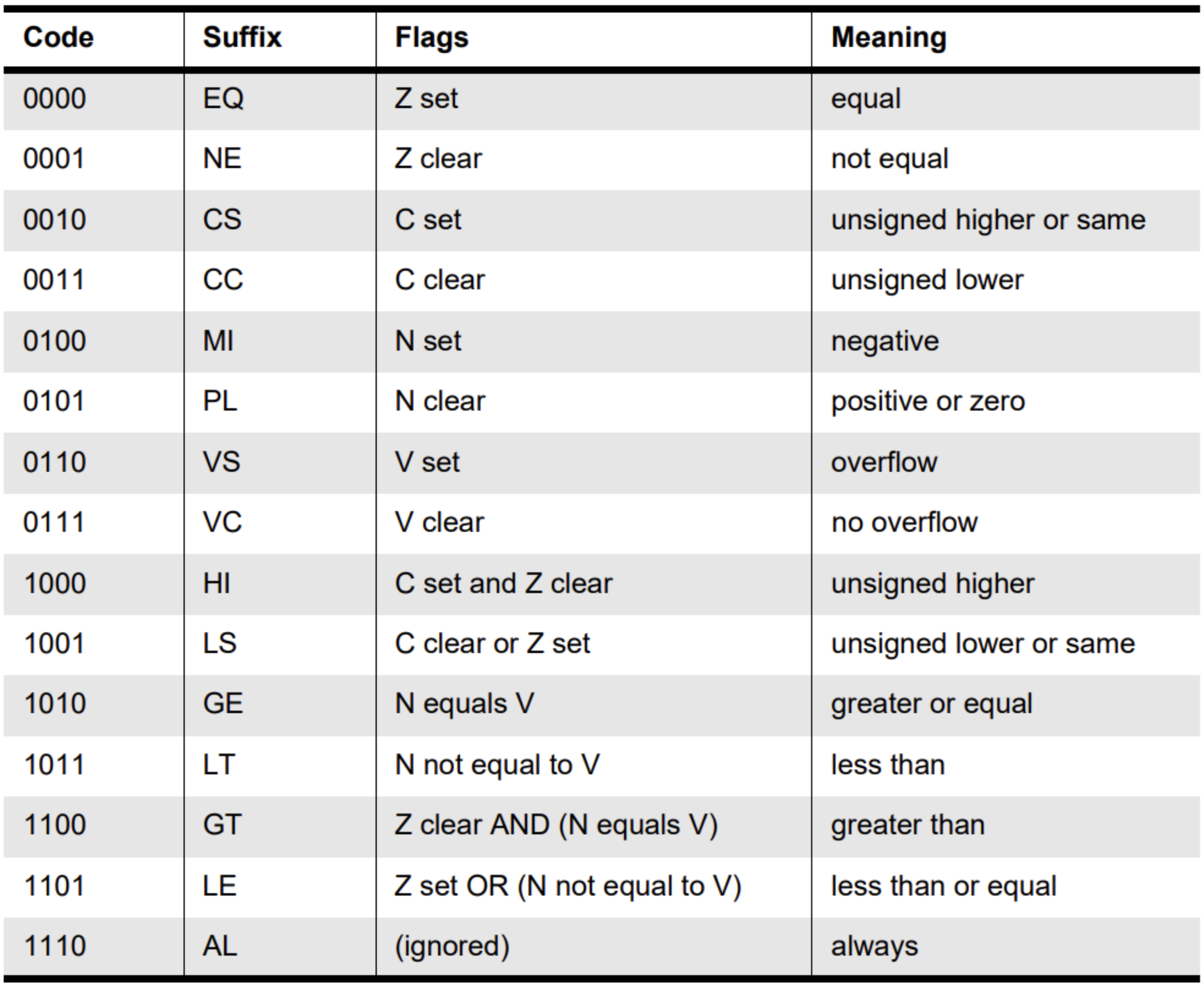


Figure 2: Multi cycle processor from the lecture notes



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)

A processor, in brief, is composed of two parts, the former of which is the datapath where all the data collection of functional units where the desired data manipulation is conducted. The latter, the control unit is the entity that directs the operation of the processor. In this multi-cycle processor implementation, the von Neumann architecture with fetch-decode-execute phases is to be considered.

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. The instructions will be stored in a unified instruction/data memory, IDM, read from, and executed. You can use the memory module designed in the first lab as the IDM. Alternatively, SDRAM usage can be an option, with the required control signals, provided that you embed the code to operate on the FPGA.

As the operations are realized on the registers as stated in the ISA, a register file, of 15 registers + pseudo PC register as R15, is required. You can use the register file module from the first laboratory or the one provided on ODTUClass.

Please note that, along with the multi-purpose register, the architecture is to include specific registers such as PC.

As stated in the previous subsection, the previous ALU can be used, without modification to operations, 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 are allowed to 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 expected to provide the codes for the self-defined complex modules; however, you may utilize the schematic design feature of Quartus for inter-module bus connections and wiring.

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)

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 with respect to the fetch-decode-execute principle.

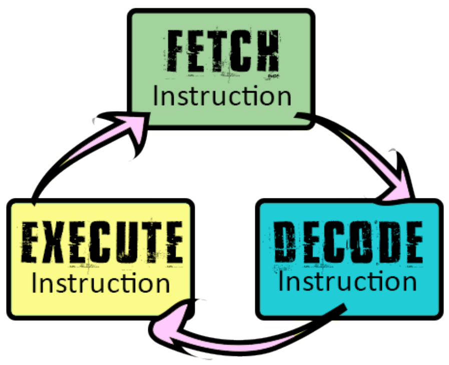


Figure 4: Fetch-Decode-Execute Sequence in Multi-Cycle Operation

In more detail, the above-mentioned cycles are:

- **Fetch Cycle:** This is the very first cycle corresponding to the operation of a single instruction and it is where 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.
- **Decode Cycle:** Within the decode cycle, the current instruction in the instruction register is decoded to obtain the conditions and the operands.
- **Execute Cycle:** In this final cycle, the data is processed and the corresponding registers and/or memory locations are updated accordingly.

You are not limited to 3 cycles per instructions, this is the minimum for an adequate multi-cycle implementation. You are encouraged to follow the lecture notes and add new cycles of execution and/or new states to the FSM such as write back, memory read, etc. However, you can also do other designs you will not lose any grades as long as you clearly explain what you did and why you did it.

Regarding the above-given descriptions, it is required that you determine which control signal in your design is to be utilized in which cycle. Intuitively, these cycles are expected to be some states of the above-mentioned FSM. For the execution of a complete program code, FSM is expected to function from the first instruction until the program ends. You can add an infinite loop to the end of your code to make the computer stop there.

For the operation of the FSM, certain external signals, namely **RUN** and **RESET** are also necessary.

- **RESET:** 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.
- **RUN:** Blocks updates to the PC if 0, that is PC is not incremented branches not taken, etc.

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 instruction.
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.

After constructing the subroutines write a cocotb testbench that checks for the correct operation of the computer.

2 Experimental Work

2.1 Multi 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 an example program that 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 in the previous labs, 1-bit signals can also be shown using on-board LEDs.

3 Parts List

`DE1-SoC Board`