Mehmet Ataş & 2304020 07.05.2022

EE446 PRELIMINARY WORK #3

Laboratory Work 3 - Single Cycle Processor Design

Introduction:

The goal of this laboratory exercise is to practice designing a 32-bit single cycle processor. I will build a single cycle processor datapath and control unit similar to the one covered in class. The developed processor will be able to carry out all of the instructions in the instruction set.

During this laboratory work, I will improve your hard-wired controller design skills by designing the single-cycle processor's controller unit. Finally, I will test my design by embedding it in the FPGA of the -DE0-Nano board.

Preliminary Work:

Single-cycle: Each instruction executes in a single cycle

- One instruction at a time

- Separate instruction, data memories (Harvard)

- Simple control, no need for non-architectural state

- Cycle time is limited by the slowest instruction

The datapath:

- operates on words of data

- Contains memories, registers, ALUs, and multiplexers.

- 32-bit ARM architecture has a 32-bit datapath.

The control unit.

- receives the current instruction from the datapath

- control the operation of the datapath tells the datapath to execute that instruction.

- produces multiplexer select, register enable, and memory write signals

Combinational Read: If the address changes, then the new data appears at RD after some propagation delay; no clock is involved.

Synchronous Write: only on the rising edge of the clock.

The state of the system is changed only at the clock edge.

The address, data, and write enable must setup before the clock edge and must remain stable until a hold time after the clock edge.

The instruction memory :

- has a single read port.

- Input (A): 32-bit instruction address input from PC

- Output (RD): 32-bit data (i.e., instruction) from that address

The data memory:

- has a single read/write port.

- 32 bit address, 32 bit data word

- If write enable, (WE)=1➔ writes data WD into address A on the rising edge of the clock.

- If WE=0➔reads address A onto RD

Register File (RF):

• 15-element × 32-bit Registers: R0–R14

– Read ports:

- • A1 and A2: 4-bit address inputs, specifying one of 24 = 16 registers as source operands.
- • RD1 and RD2: read data outputs, read the 32-bit register

– The write port:

• A3: 4-bit address input • WD3: 32-bit write data input • WE3: a write enable input

– If WE3=1➔then the register file writes the data into the specified register on the rising edge of the clock.

• Input port R15:

– The data presented on the INPUT port R15 is the value of the register R15 and not the actual register R15 in the register file

– When A1/A2=15 RD1/RD2 are connected to input R15 but not the internal register R15

– PC + 8 is read from RD1/RD2

– Reading R15 returns PC+8 is because of the three-stage pipeline, so the PC value read by an instruction is two instructions (or eight bytes) ahead

• Writes to R15 must be specially handled. Many instructions restrict R15 to be the destination register.

• When an instruction writes R15 the value written to R15 is treated as an instruction address and a branch occurs to that address

• PC: The address of the current instruction is kept in a 32-bit register called the program counter (PC), which is register R15.

• PC is an alternative name for R15

– logically part of the register file

– read and written on every cycle independent of the normal register file operation➔RF read/write ports are not used

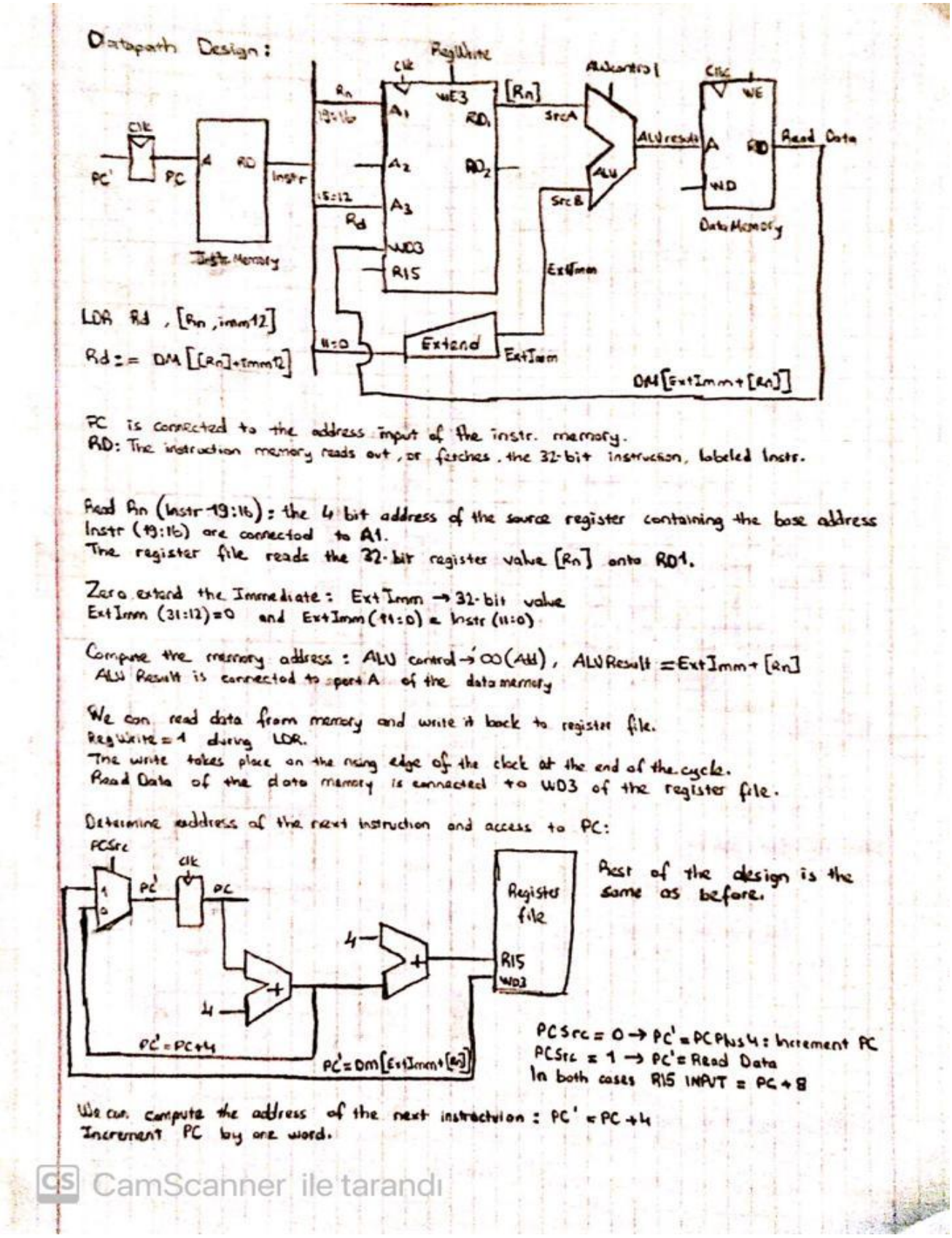
– more naturally built as a stand-alone 32-bit register.

– Output (PC): address of current instruction

– Input (PC'): address of the next instruction

1.2.1 Datapath Design:

1. For the instructions in the CPU that I am going to design, I list all the steps that are needed for the execution of each instruction.



PC is connected to the address input of the instr. memory.
RD: The instruction memory reads out, or fetches, the 32-bit instruction, labeled Instr.

Read Rn (Instr 19:16): the 4 bit address of the source register containing the base address
Instr (19:16) are connected to A1.
The register file reads the 32-bit register value [Rn] onto RD1.

Zero extend the Immediate: ExtImm → 32-bit value
ExtImm (31:12)=0 and ExtImm(11:0) = Instr(11:0)

Compute the memory address: ALU control → 00 (Add), ALUResult = ExtImm + [Rn]
ALU Result is connected to port A of the data memory

We can read data from memory and write it back to register file.
RegWrite = 1 during LDR.
The write takes place on the rising edge of the clock at the end of the cycle.
Read Data of the data memory is connected to WD3 of the register file.

Determine address of the next instruction and access to PC:

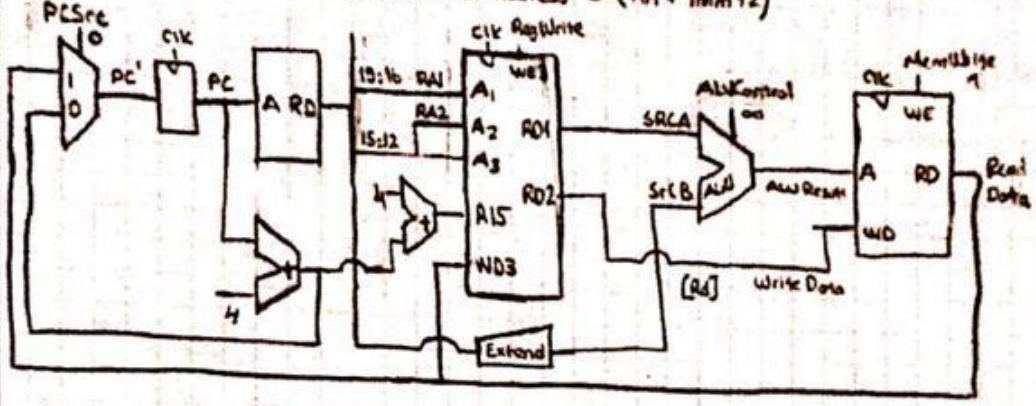
Rest of the design is the same as before.

PCSrc = 0 → PC' = PCPlus4: Increment PC
PCSrc = 1 → PC' = Read Data
In both cases R15 INPUT = PC+8

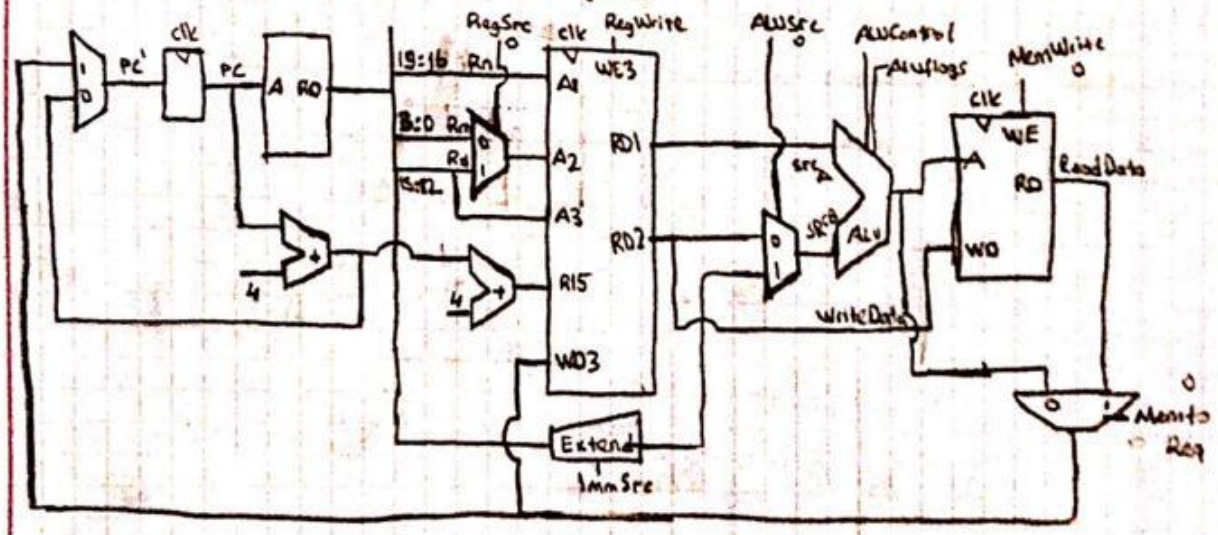
We can compute the address of the next instruction: PC' = PC + 4
Increment PC by one word.

STR Immediate instruction: STR Rd, [Rn, imm12]

The content of Rd is stored in address = (Rn + imm12)



Address computation is already implemented.
Reads Rd from the memory by also connecting it to the read port A2 [Rd] is on RD2.
[Rd] is written in the Data Memory at address ExtImm + [Rn]
RegWrite=0 , MemWrite=1, Read Data is ignored.



Expand datapath to handle Data-processing. Data processing instructions → ADD - SUB - OR - AND
Read a source register Rn from the register file and Operand.
Perform same ALU operation on them , Write the result back Rd register.
ALU control signals select which operation to be executed.
ALU flags (3:0) → Zero, Negative, Carry, Overflow, that are sent back to the controller.
RegSrc = 0 → RA2 = Rm (Data Processing)", RegSrc = 1 → RA2 = Rd (STR)
ImmSrc = 0 → ExtImm is zero extended from Instr (7:0)
ImmSrc = 1 → ExtImm is zero extended from Instr (11:0) for LDR or STR
ALUSrc = 0 → SrcB = [Rm] , ALUSrc = 1 → SrcB = ExtImm: imm8
MemtoReg = 0 → Result = ALUResult , MemtoReg = 1 → Result = ReadData

A combinational shifter is added to the RD1 output of the RF.
RD1 output and shifter output are mixed together and connected to the SrcA input of ALU.
Shifter has 2 inputs. One of them is the output of RD1, the other one is Ext Imm.
5 bits determine the shift amount.
SrcB input of the ALU changed with 4:1 MUX. "0" is connected to this as a 3. input.

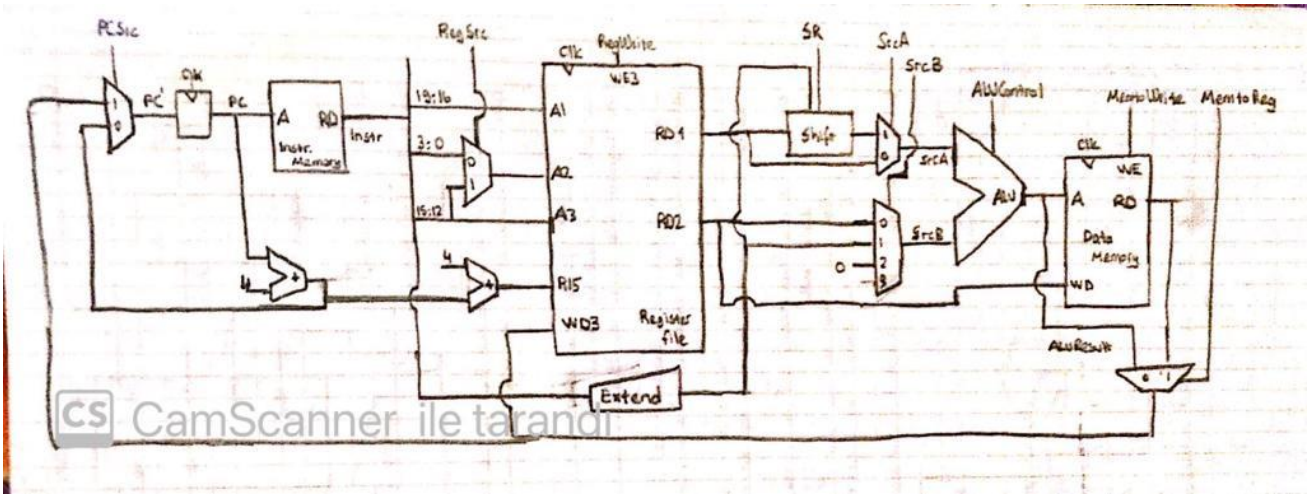
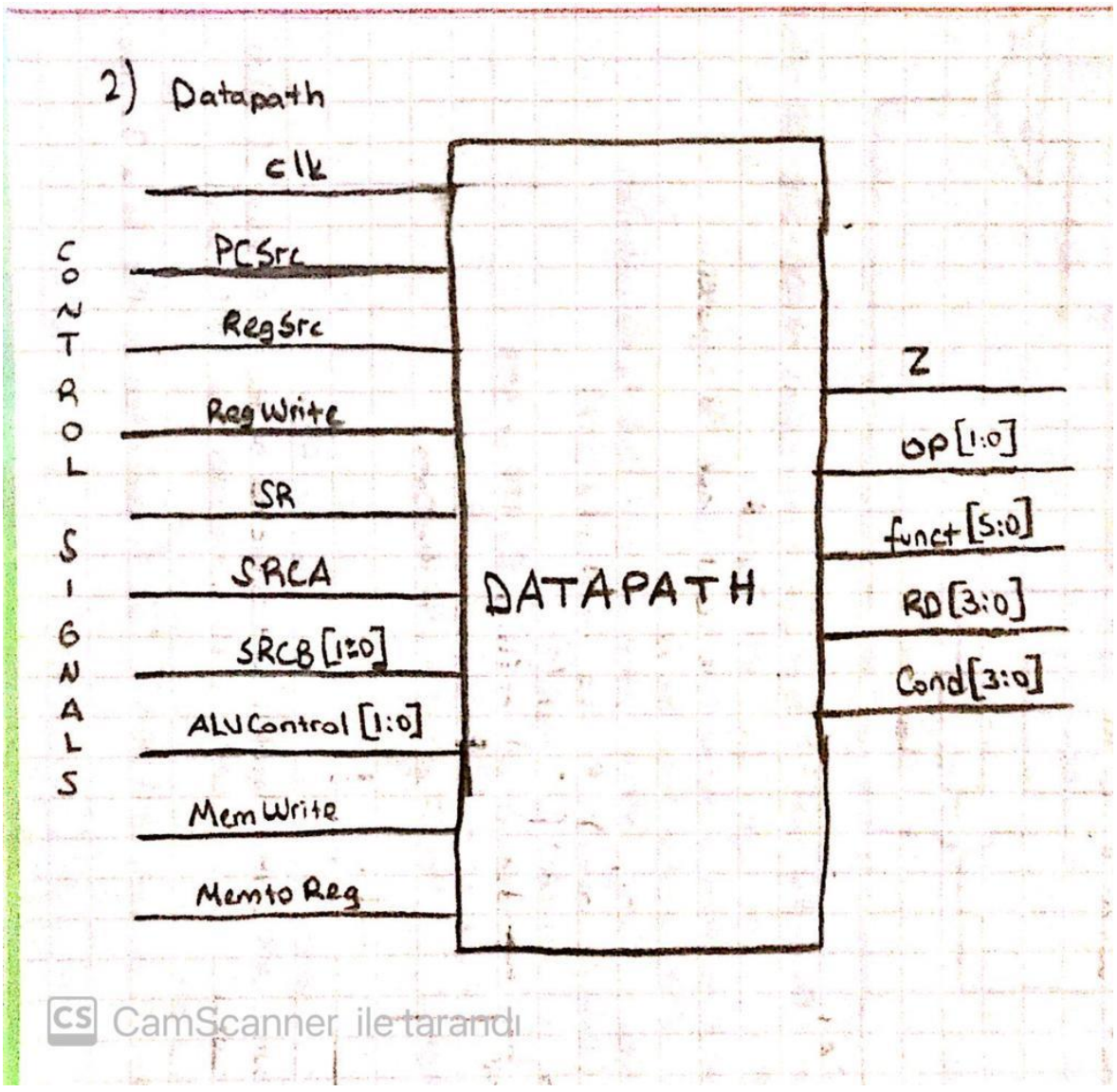


Figure 1. Overall Datapath

2. I state the control signal inputs of my overall design in previous part. Now, I draw a black box diagram of my architecture by indicating the inputs and outputs.



3.

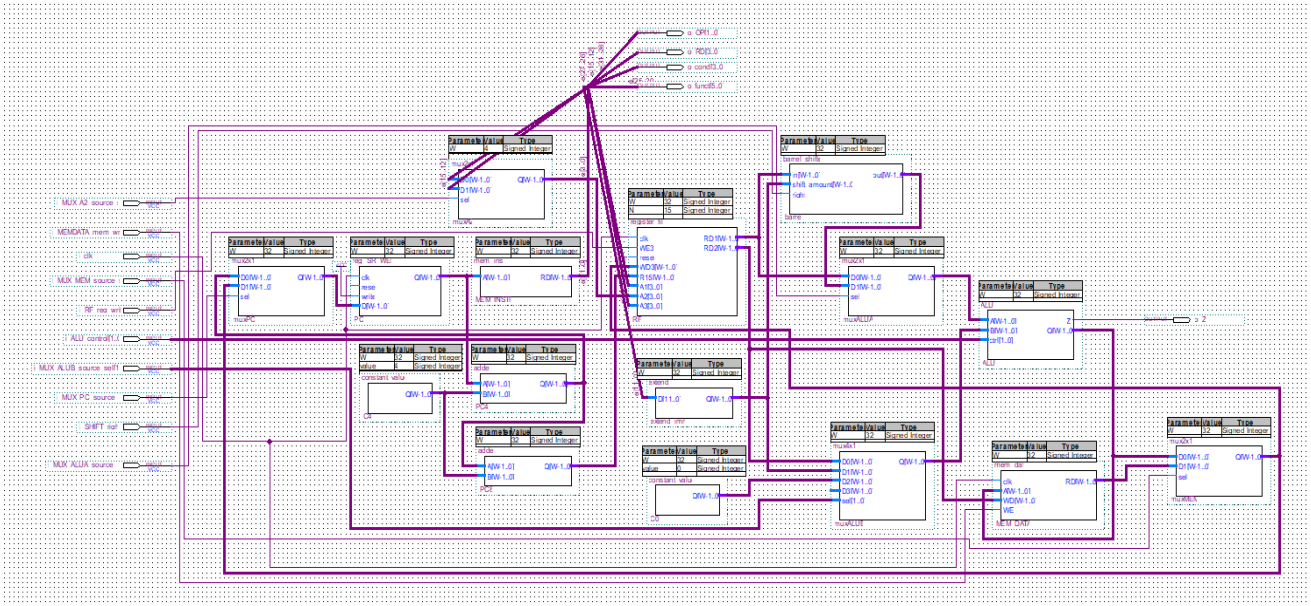


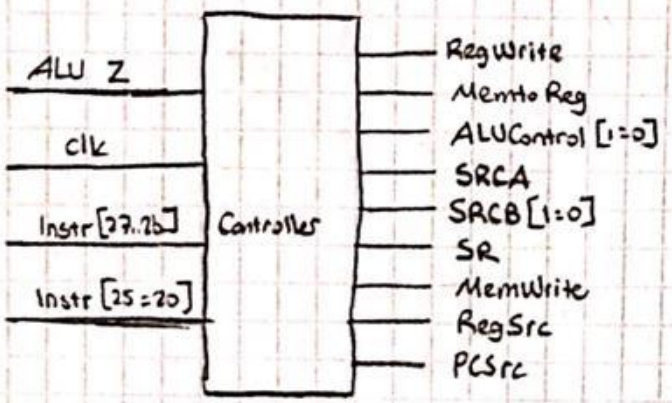
Figure 2. Implementation of my datapath design in Schematic Editor of Quartus

1.2.2 Controller Design:

1.2.

Controller:

1)



2)

We don't need to access to memory in the ADD, SUB, OR, AND instructions.
The only difference btw. them is the ALUControl signal.
PC has a mux at its inputs
This MUX has a select (PCSrc).
ALU result is mixed with Read Data.
This MUX has a select (MemtoReg)

When we consider LSL, LSR instructions, we understand SR signal is necessary.
SR signal of the shifter is handled by controller.

For CMP, the write enable of Z register to 1,

For STR, 2x1 Mux is added to the A2 input of the RF.
It has a select signal called RegSrc
MemWrite is added.

For LDR, RegWrite is added.

2x1 Mux is added to SRCA input of the ALU.
It has a select signal called SrcA.
It selects RD1 or output of shifter.

4x1 Mux is added to SRCB input of the ALU.
It has a select signal called SrcB.
It selects RD2 or ExtImm or 0.

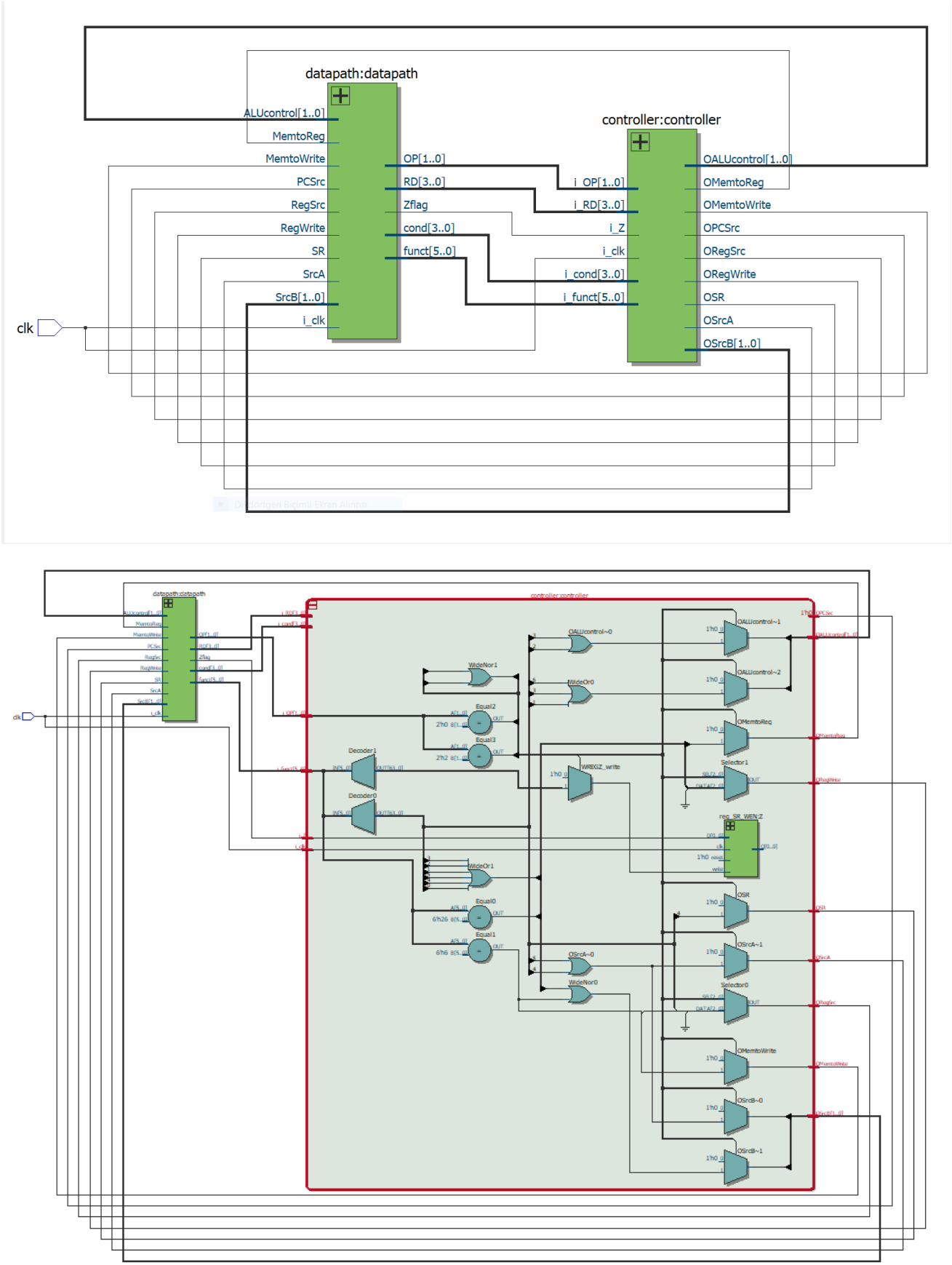
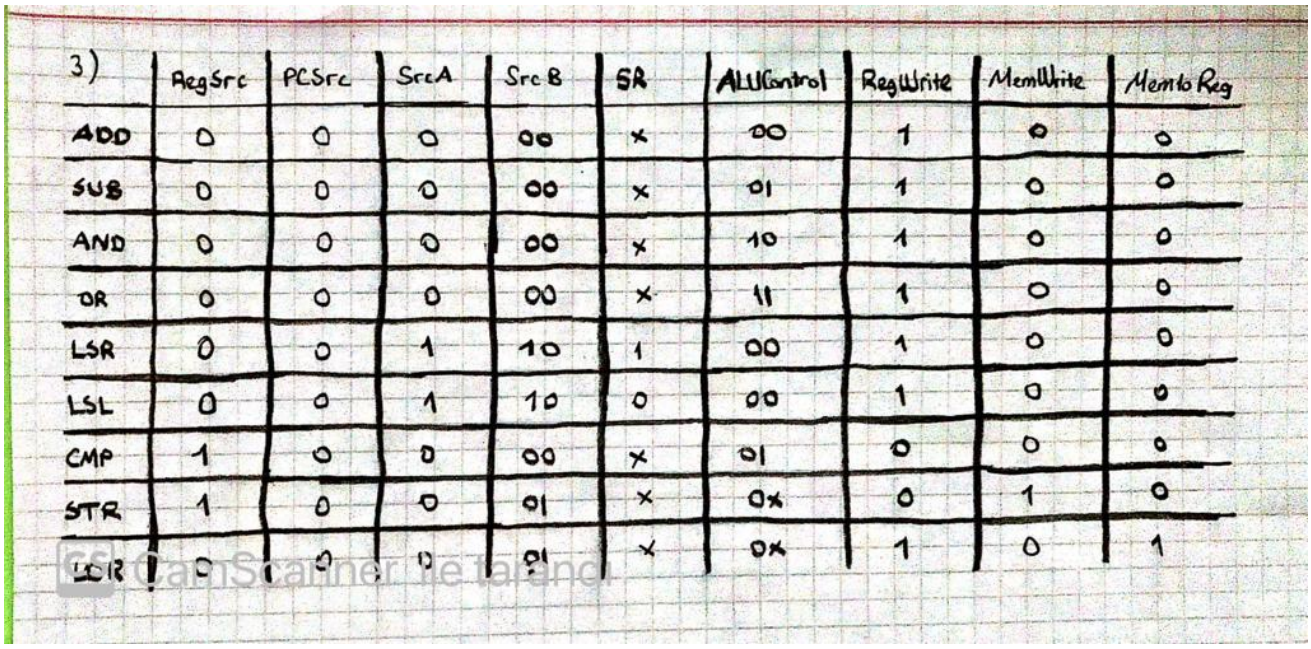


Figure 3. RTL Viewer of the controller

3.



3)	RegSrc	PCSrc	SrcA	SrcB	SR	ALUControl	RegWrite	MemWrite	MemtoReg
ADD	0	0	0	00	x	00	1	0	0
SUB	0	0	0	00	x	01	1	0	0
AND	0	0	0	00	x	10	1	0	0
OR	0	0	0	00	x	11	1	0	0
LSR	0	0	1	10	1	00	1	0	0
LSL	0	0	1	10	0	00	1	0	0
CMP	1	0	0	00	x	01	0	0	0
STR	1	0	0	01	x	0x	0	1	0
LDR	0	0	0	01	x	0x	1	0	1

Figure 4. The truth table for the main controller of the single-cycle processor

4.

I implemented my controller in Verilog HDL. It is uploaded to ODTUCLASS.

Simulation Results: I will show a simulation of the instructions below.

//LDR R1,[R6,#0] ***R1 = 6***

//LDR R2, [R6,#4] ***R2 = 2***

//ADD R3,R2,R1 ***R3 = 8***

//SUB R4,R1,R2 ***R4 = 4***

//LSL R5,R4,#1 ***R5 = 8***

//CMP R5,R3 ***Z_Reg = 1***

//CMP R5,R2 ***Z_Reg = 0***

//STR R5,[R5,#8] ***memory 16 -> 08***

//LDR R7,[R6,#24] ***R7 = FF00FF00***

//LDR R8,[R6,#28] ***R8 = 00FFFF00***

//AND R9,R8,R7 ***R9 = 0000FF00***

//ORR R10,R8,R7 ***R10 = FFFFFF00***

//LSR R11,R9,#8 ***R11 = 000000FF***

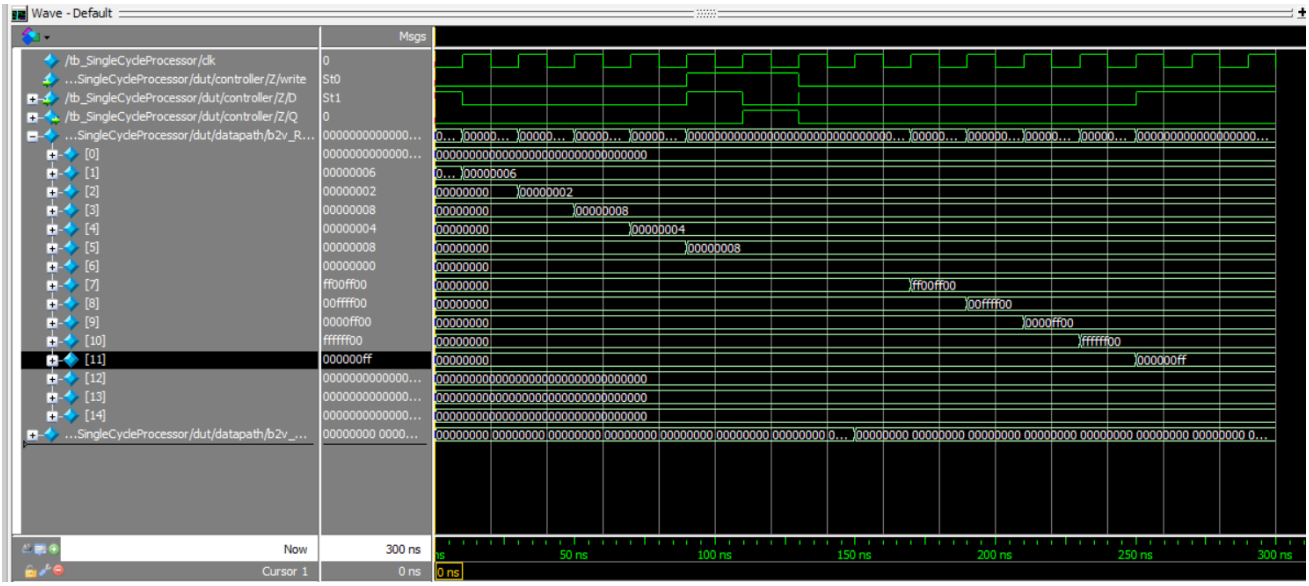


Figure 5. Simulation showing register values and z flag.

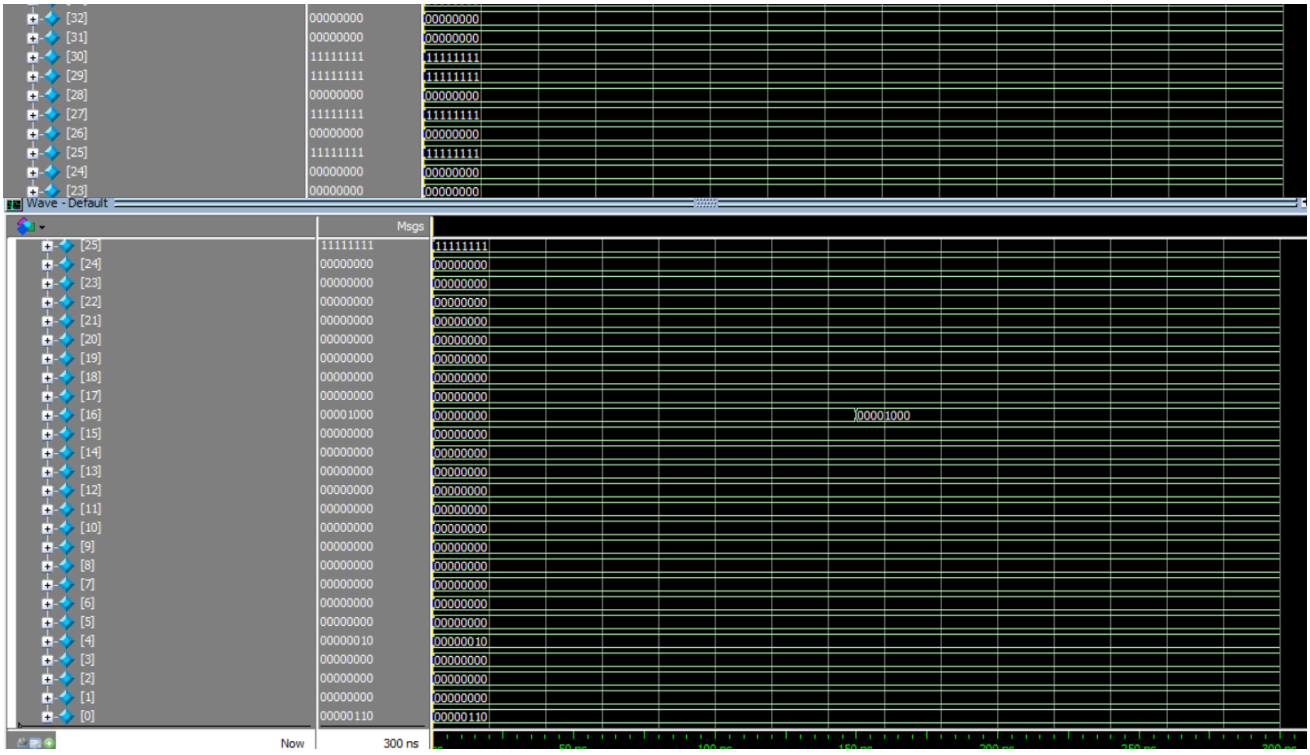


Figure 6. Memory

Initially, all register values are 0.

1. ***Clock cycle: LDR R1,[R6,#0] → R1 = 6 since memory location 0. is 6 as you can see from Figure 4***
2. ***Clock cycle: LDR R2, [R6,#4] → R2 = 2 since memory location 4. is 2 as you can see from Figure 4***
3. ***Clock cycle: ADD R3,R2,R1 → R3 = 8 since 6+2 is 8 as you can see from Figure 4.***
4. ***Clock cycle: SUB R4,R1,R2 → R4 = 4 since 6-2 is 4 as you can see from Figure 4.***
5. ***Clock cycle: LSL R5,R4,#1 → R5 = 8 since 4*2 is 8 as you can see from Figure 4.***
6. ***Clock cycle: CMP R5,R3 → Z = 1 since 8-8 is 0 as you can see from Figure 4.***
7. ***Clock cycle: CMP R5,R2 → Z = 0 since 8-2 is not 0 as you can see from Figure 4.***
8. ***Clock cycle: STR R5,[R5,#8] → memory location 16. is 08 as you can see from Figure 5.***
9. ***Clock cycle: LDR R7,[R6,#24] → R7 = FF00FF00 since memory location 27.26.25.24 is FF00FF00 as you can see from Figure 5.***
10. ***Clock cycle: LDR R8,[R6,#28] → R8 = 00FFFF00 since memory location 27.26.25.24 is FF00FF00 as you can see from Figure 5.***
11. ***Clock cycle: AND R9,R8,R7 → R9 = 0000FF00 as you can see from Figure 4.***
12. ***Clock cycle: OR R10,R8,R7 → R10 = FFFFFF00 as you can see from Figure 4.***
13. ***LSR R11,R9,#8 → R11 = 000000FF as you can see from Figure 4.***