Mehmet Ataş & 2304020 05.06.2022

EE446 PRELIMINARY WORK #5

Laboratory Work 4 - ISA & Datapath Design for Multi-Cycle CPU

Introduction:

The goal of this laboratory activity is to construct a multi-cycle CPU with a control unit that extends the previously defined datapath.

The previously developed draft instruction set will be finished with the machine code format, i.e. the binary representations in machine language, and will be put in the instruction/data memory to serve for the entire execution of a micro-controlled program. The control unit's design will be based on the previously built datapath architecture, which is capable of running sample code written using this ISA.

Preliminary Work:

Datapath

– architectural state elements, memories combinational logic

– nonarchitectural state elements to hold intermediate results between the steps

Controller

– produces different signals on different steps during execution of a single instruction

– finite state machine rather than combinational logic

Replace Instruction and Data memories with a single unified memory – more realistic

Step 1: Instruction Fetch:

Read instruction in one cycle, read/write data in another cycle

New non-architectural instruction register (IR): read the instruction in IR, store it for future cycles.

IRWrite: asserted when the IR should be loaded with a new instruction.

Step 2: LDR Register Read:

Read one source (Rn) from register file into non-architectural register A

Extend the second source from the immediate field

Step 3: LDR Address:

Add base address to offset

ALUOut: Non-architectural Register

Step 4: LDR Read data from memory:

Memory address: Adr

AdrSrc selects Adr from either the PC or ALUOut

Data: Non-architectural register

Step 5: LDR Write-back to Register File:

Other instructions will need to write a result from the ALU

Multicycle Data path for LDR

Non architectural registers Instr (IR), A, ALUOut, Data to store

intermediate results

Reuse of ALU

Go over the instruction execution together with PC updates

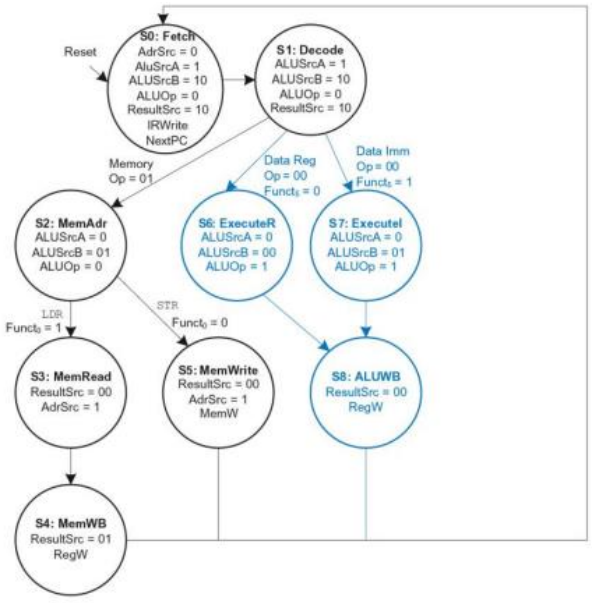
Data Processing Immediate

Read the first source from Rn and extend the second source from an 8-bit immediate.

The datapath already contains all the connections necessary

The ALU uses the ALUControl signal to determine the type of data processing instruction to execute.

The ALUFlags are sent back to the controller to update the Status register.



State	Datapath µOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset

1.2 ISA Configuration: From Mnemonics to Machine Code

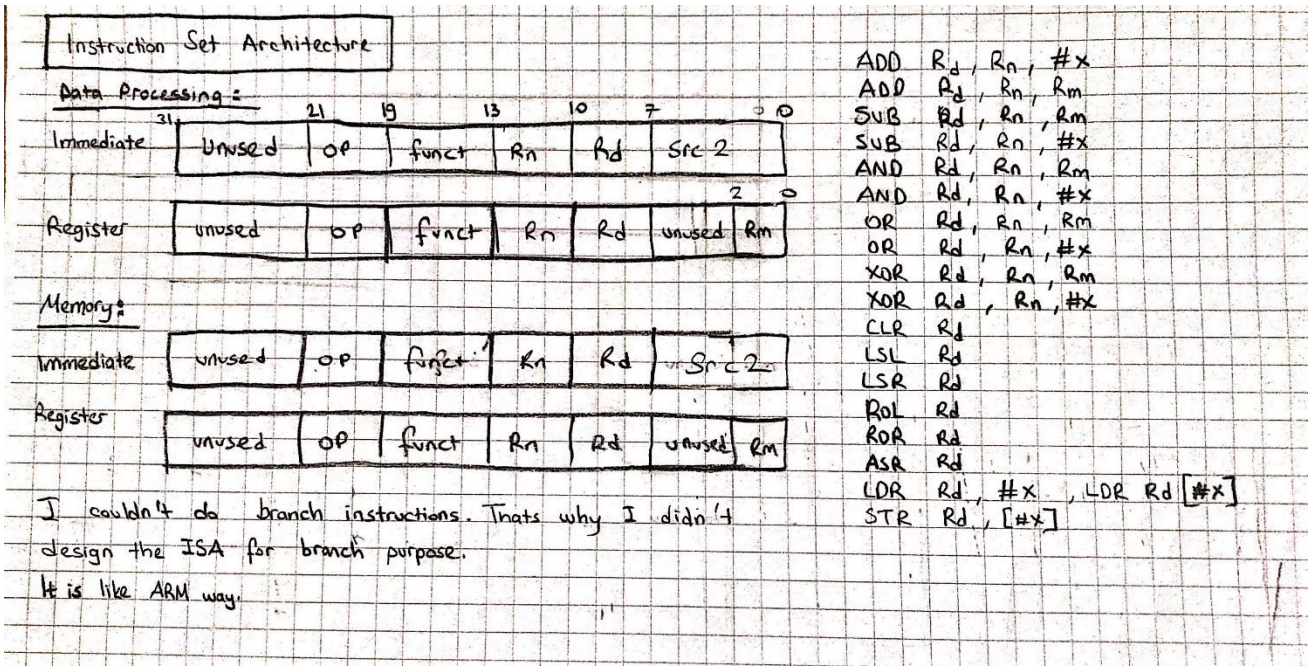


Figure 1. ISA

I built the instruction set architecture (ISA) that will run on the specified CPU for this section. Because the ISA must allow general-purpose computing, it includes instructions for arithmetic-logic, memory load-store, and program control activities. There should be enough addressing modes to accommodate both basic constants and variables, as well as more complicated data structures like indexed arrays. The ISA setup, like the ISA in Laboratory Work #3, is anticipated to include the following instructions as well as possible additions of my choice.

1. Arithmetic Operations • addition • addition indirect (bonus) • subtraction • subtraction indirect (bonus)

2. Logic Operations • and • or • xor • clear

3. Shift Operations • rotate left • rotate right • shift left • arithmetic shift right • logical shift right

4. Branch Operations • branch unconditional • branch with link • branch indirect • branch if zero • branch if not zero

(I couldn’t do the branch operations)

5. Memory Operations • load to register from memory • load immediate to register • store from register to memory

For example, the MSB 6 bits are useless for each operation type, therefore I might use fewer bits for the instructions. However, in order to maintain compatibility with the ARM ISA, I've chosen to utilize 32-bit long instructions.

I have used similar ISA that I implemented on the LAB4.

1.2.1 Multi-cycle CPU Controller Unit Design

My datapath is the same as LAB4.

• 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 desired data manipulation is realized and the corresponding registers and/or memory locations are updated accordingly.

RUN: The external signal that incites the execution of the code from the current instruction (active high).

RESET: Terminates the operation and sets the PC to the very first slot in the instruction/data memory (active high).

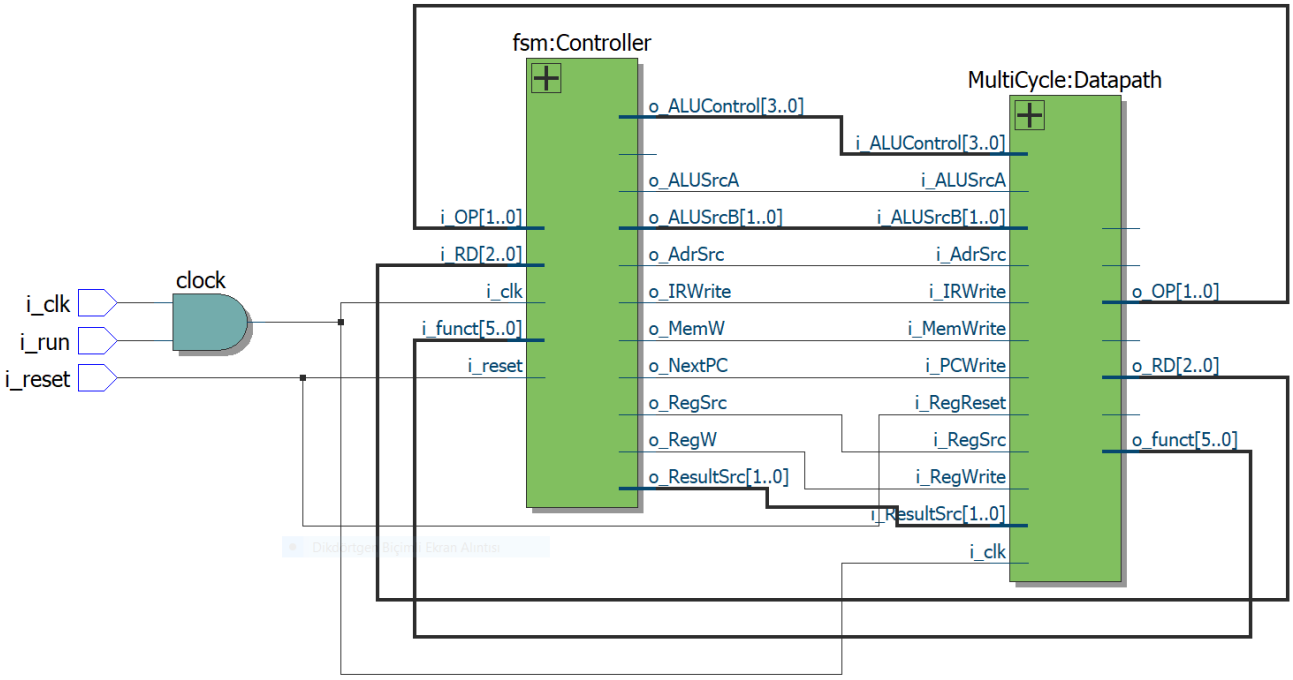


Figure 1. RTL View of the overall design

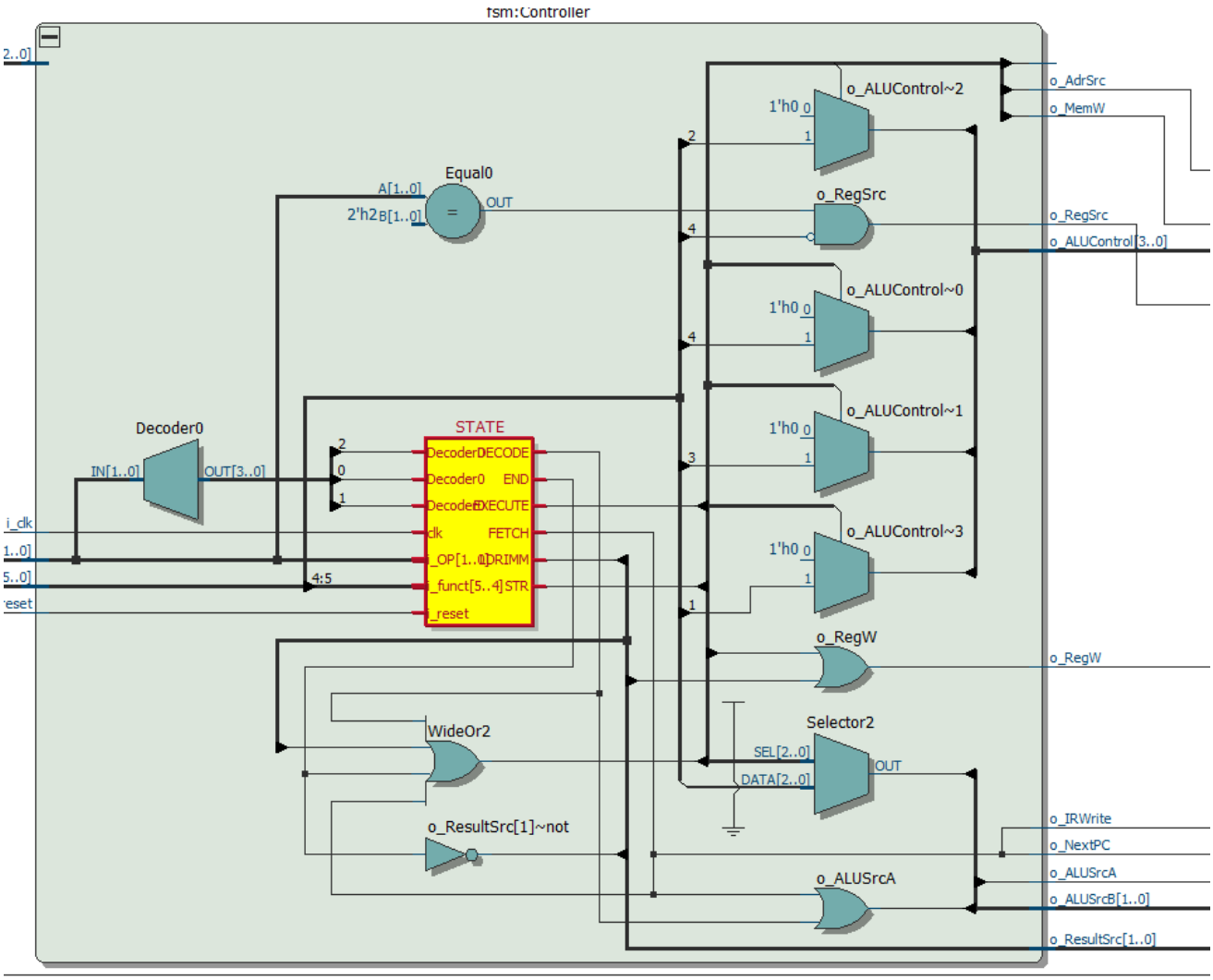


Figure 2. RTL View of controller

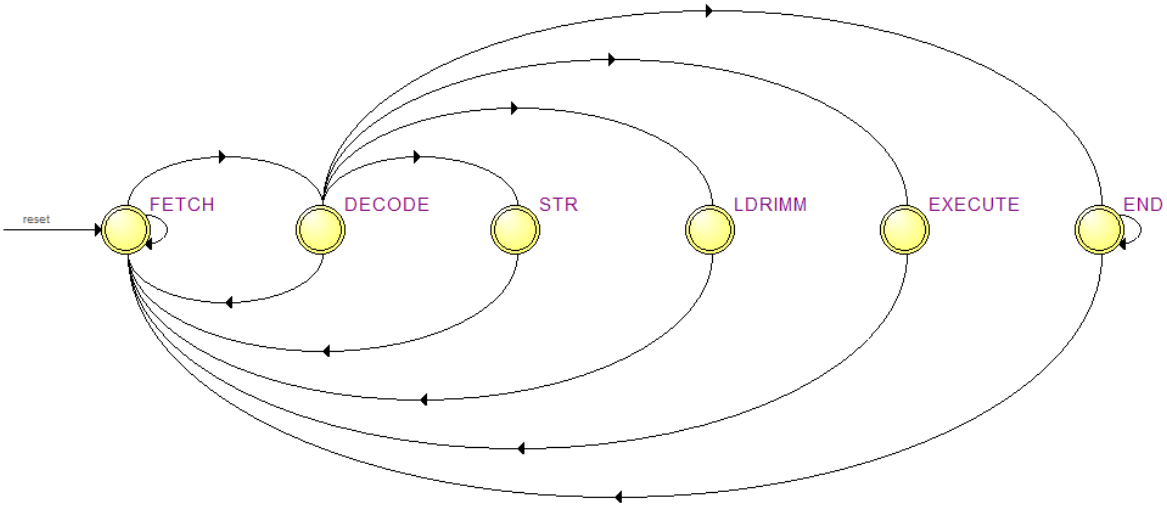


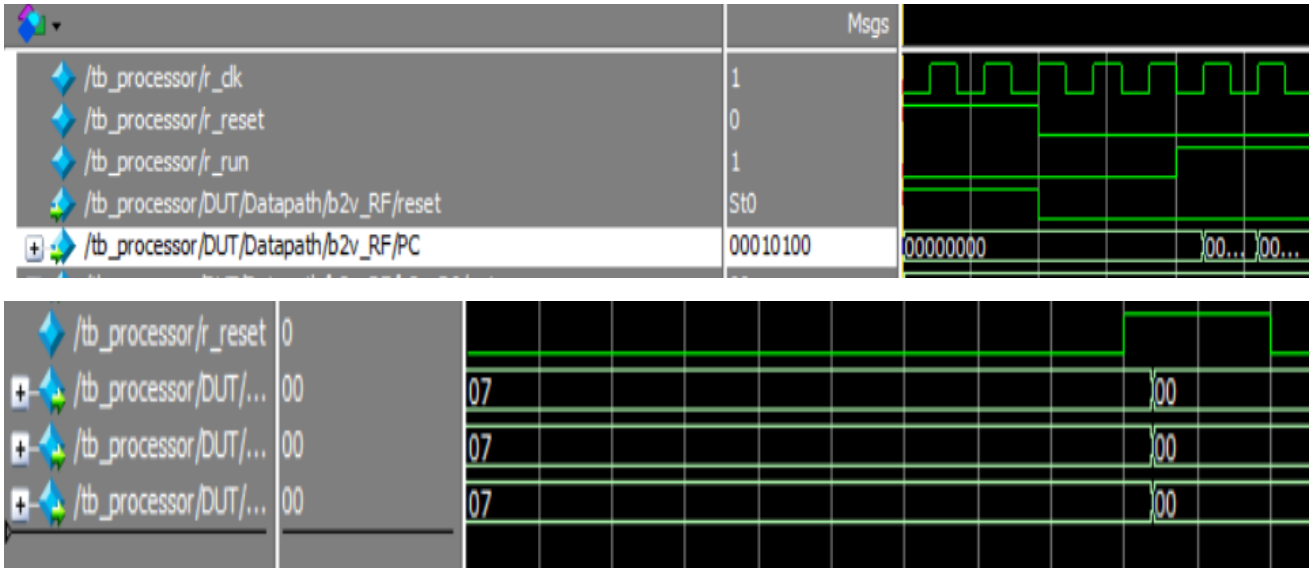
Figure 3. RTL View of the state

```
   FETCH    :  begin

                  o_RegW            = 1'b0;
                  o_MemW            = 1'b0;
                  o_IRWrite         = 1'b1;
                  o_NextPC          = 1'b1;
                  o_AdrSrc          = 1'b0;
                  o_ResultSrc [1:0] = 2'b10;
                  o_ALUSrcA         = 1'b1;
                  o_ALUSrcB   [1:0] = 2'b10;
                  o_ALUOp           = 1'b0;
               end

   DECODE   :  begin

                  o_RegW            = 1'b0;
                  o_MemW            = 1'b0;
                  o_IRWrite         = 1'b0;
                  o_NextPC          = 1'b0;
                  o_AdrSrc          = 1'b0;
                  o_ResultSrc [1:0] = 2'b10;
                  o_ALUSrcA         = 1'b1;
                  o_ALUSrcB   [1:0] = 2'b10;
                  o_ALUOp           = 1'b0;

               end

 EXECUTE   :  begin

                  o_RegW            = 1'b1;
                  o_MemW            = 1'b0;
                  o_IRWrite         = 1'b0;
                  o_NextPC          = 1'b0;
                  o_AdrSrc          = 1'b0;
                  o_ResultSrc [1:0] = 2'b10;
                  o_ALUSrcA         = 1'b0;
                  if (i_funct[5] == 1'b1)
                     o_ALUSrcB[1:0] = 2'b01;
                  else
                     o_ALUSrcB   [1:0] = 2'b00;
                  o_ALUOp           = 1'b1;
               end


 LDRIMM    :  begin

                  o_RegW            = 1'b1;
                  o_MemW            = 1'b0;
                  o_IRWrite         = 1'b0;
                  o_NextPC          = 1'b0;
                  o_AdrSrc          = 1'b0;
                  o_ResultSrc [1:0] = 2'b11;
                  o_ALUSrcA         = 1'b0;
                  o_ALUSrcB   [1:0] = 2'b00;
                  o_ALUOp           = 1'b0;
               end
```

```
STR      :  begin

                o_RegW            = 1'b0;
                o_MemW            = 1'b1;
                o_IRWrite         = 1'b0;
                o_NextPC          = 1'b0;
                o_AdrSrc          = 1'b1;
                o_ResultSrc [1:0] = 2'b10;
                o_ALUSrcA         = 1'b0;
                o_ALUSrcB   [1:0] = 2'b01;
                o_ALUOp           = 1'b0;
            end
```

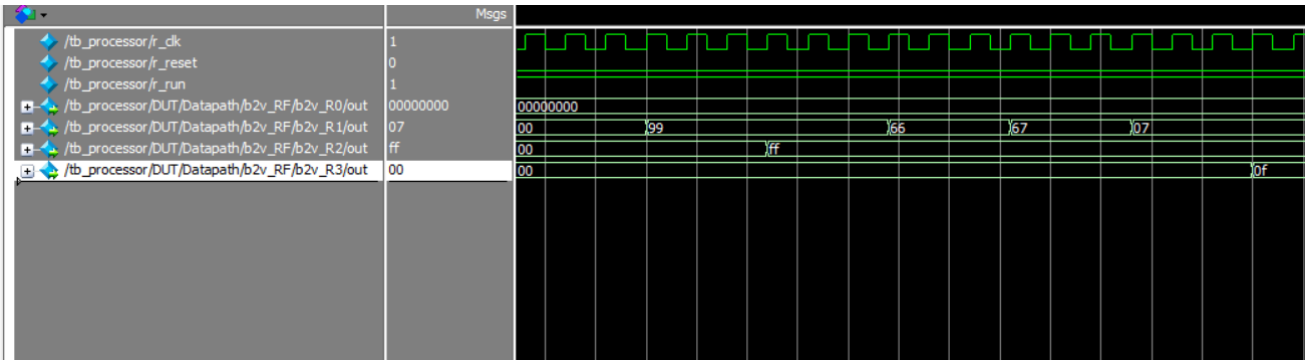
The FSM generates each instruction's control signal, which is seen above.

1.2.2 Validation of Operation via Microprogrammed Control



The above results are showing that the RESET and the RUN signals work as intended.

SUBROUTINE 1:



I write a subroutine that get an 8-bit number and computes it 2's complement.

As you can see, first I load 0x99 to R1. Then I take 1's complement which is 0x66. Next, I take 2's complement by adding 1 to the 1's complement. The result becomes 0x67.

In addition to the subroutine 1, I want to show that I can use AND operation.

AND R1, R1 #0x0F which result with 0x07

Based on the simulation findings, this multi-cycle CPU appears to be in good working order.

This laboratory does not test every potential instruction. All of them, however, were tested and simulated in the prior laboratory. As a result, I've opted to simply replicate the subroutine that have been provided.

//LDR R1, #99

039C0199

//LDR R2, #FF

039C02FF

//SUB R1,R2,R1

03809101

//ADD R1, R1, #0x01

03880901

//AND R1, R1 #0x0F

0009090F