ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 2 REPORT

CRN	:	21336
LECTURER	:	Prof. Dr. Mustafa Ersel Kamaşak

GROUP MEMBERS:

150210022	:	HARUN YAHYA DEMİRPENÇE
150210030	:	EMRE BOZKURT

SPRING 2024

Contents

1 INTRODUCTION

In this project, using the framework created in Part 4 of Project 1, we are entrusted with creating a hardwired control unit for a particular architecture. The instructions that are stored in memory will be executed by this control unit in accordance with a predetermined instruction format.

2 MATERIALS AND METHODS

2.1 Design and Development Tools

The project was developed using the Verilog hardware description language. Code development, simulation, and Verilog module synthesis were carried out using the Vivado Design Suite.

2.2 Design Approach

2.2.1 Fetch and Decode

The initial value of PC is 0.

1. $IR[7:0] \leftarrow M[PC], PC \leftarrow PC + 1$
2. $IR[15:8] \leftarrow M[PC], PC \leftarrow PC + 1$
3. For instruction with an address reference: $OpCode \leftarrow IR[15:10], RSel \leftarrow IR[9:8], Address \leftarrow IR[7:0]$

   For instruction without an address reference: $OpCode \leftarrow IR[15:10], S \leftarrow IR[9], DstReg \leftarrow IR[8:6], SReg1 \leftarrow IR[5:3], SReg2 \leftarrow IR[2:0]$

2.2.2 Execution

- BRA
  1. $S1 \leftarrow Address$
  2. $S2 \leftarrow PC$
  3. $ALUOut \leftarrow S1 + S2, PC \leftarrow ALUOut$
- BNE
  1. $if(Z == 0)$ then BRA

- BEQ
  1. $if(Z == 1)$ then BRA
- POP
  1. $Rx[7:0] \leftarrow M[SP], SP \leftarrow SP + 1$
- PSH
  1. $M[SP] \leftarrow Rx[7:0], SP \leftarrow SP + 1$
  2. $M[SP] \leftarrow RX[15:8], SP \leftarrow SP - 1$
- INC
  1. $DstReg \leftarrow SReg1$
  2. $DstReg \leftarrow DstReg + 1$
- DEC
  1. $DstReg \leftarrow SReg1$
  2. $DstReg \leftarrow DstReg - 1$
- LSL
  1. if($SReg1 == PC||SReg1 == SP||SReg1 == AR$) then $S1 \leftarrow Sreg1$ else $DstReg \leftarrow LSLSReg1$
  2. $if(SReg1 == PC||SReg1 == SP||SReg1 == AR)$ then $DstReg \leftarrow LSLS1$
- LSR
  1. if($SReg1 == PC||SReg1 == SP||SReg1 == AR$) then $S1 \leftarrow Sreg1$ else $DstReg \leftarrow LSRSReg1$
  2. $if(SReg1 == PC||SReg1 == SP||SReg1 == AR)$ then $DstReg \leftarrow LSRS1$
- ASR
  1. if($SReg1 == PC||SReg1 == SP||SReg1 == AR$) then $S1 \leftarrow Sreg1$ else $DstReg \leftarrow ASRSReg1$
  2. $if(SReg1 == PC||SReg1 == SP||SReg1 == AR)$ then $DstReg \leftarrow ASRS1$

- CSL
    1. if($SReg1 == PC || SReg1 == SP || SReg1 == AR$) then $S1 \leftarrow Sreg1$ else $DstReg \leftarrow CSLSReg1$
    2. $if(SReg1 == PC || SReg1 == SP || SReg1 == AR)$ then $DstReg \leftarrow CSLS1$
- CSR
    1. if($SReg1 == PC || SReg1 == SP || SReg1 == AR$) then $S1 \leftarrow Sreg1$ else $DstReg \leftarrow CSRSReg1$
    2. $if(SReg1 == PC || SReg1 == SP || SReg1 == AR)$ then $DstReg \leftarrow CSRS1$
- AND
    1. the case where SReg1 and Sreg2 from ARF:
        (a) $S1 \leftarrow SReg1$
        (b) $S2 \leftarrow SReg2$
        (c) $DstReg \leftarrow S1$ AND $S2$
    2. the case where SReg1 from ARF, SReg2 from RF:
        (a) $S1 \leftarrow SReg1$
        (b) $DstReg \leftarrow S1$ AND $SReg2$
    3. the case where SReg1 from RF, SReg2 from ARF:
        (a) $S1 \leftarrow SReg2$
        (b) $DstReg \leftarrow SReg1$ AND $S1$
    4. the case where SReg1 and Sreg2 from RF:
        (a) $DstReg \leftarrow SReg1$ AND $SReg2$
- ORR
    1. the case where SReg1 and Sreg2 from ARF:
        (a) $S1 \leftarrow SReg1$
        (b) $S2 \leftarrow SReg2$
        (c) $DstReg \leftarrow S1$ OR $S2$
    2. the case where SReg1 from ARF, SReg2 from RF:
        (a) $S1 \leftarrow SReg1$
        (b) $DstReg \leftarrow S1$ OR $SReg2$
    3. the case where SReg1 from RF, SReg2 from ARF:

    (a) $S1 \leftarrow SReg2$
    (b) $DstReg \leftarrow SReg1$ OR $S1$
4. the case where SReg1 and Sreg2 from RF:
    (a) $DstReg \leftarrow SReg1$ OR $SReg2$

- NOT
  1. the case where SReg1 from ARF:
      (a) $S1 \leftarrow SReg1$
      (b) $DstReg \leftarrow$ NOT $S1$
  2. the case where SReg1 from RF:
      (a) $DstReg \leftarrow$ NOT $SReg1$

- XOR
  1. the case where SReg1 and Sreg2 from ARF:
      (a) $S1 \leftarrow SReg1$
      (b) $S2 \leftarrow SReg2$
      (c) $DstReg \leftarrow S1$ XOR $S2$
  2. the case where SReg1 from ARF, SReg2 from RF:
      (a) $S1 \leftarrow SReg1$
      (b) $DstReg \leftarrow S1$ XOR $SReg2$
  3. the case where SReg1 from RF, SReg2 from ARF:
      (a) $S1 \leftarrow SReg2$
      (b) $DstReg \leftarrow SReg1$ XOR $S1$
  4. the case where SReg1 and Sreg2 from RF:
      (a) $DstReg \leftarrow SReg1$ XOR $SReg2$

- NAND
  1. the case where SReg1 and Sreg2 from ARF:
      (a) $S1 \leftarrow SReg1$
      (b) $S2 \leftarrow SReg2$
      (c) $DstReg \leftarrow S1$ NAND $S2$
  2. the case where SReg1 from ARF, SReg2 from RF:
      (a) $S1 \leftarrow SReg1$

   (b) $DstReg \leftarrow S1$ NAND $SReg2$

3. the case where SReg1 from RF, SReg2 from ARF:
   (a) $S1 \leftarrow SReg2$
   (b) $DstReg \leftarrow SReg1$ NAND $S1$
4. the case where SReg1 and Sreg2 from RF:
   (a) $DstReg \leftarrow SReg1$ NAND $SReg2$

- MOVH
  1. $DSTReg[15:8] \leftarrow Address$
- LDR
  1. $Rx[7:0] \leftarrow M[AR]$
- STR
  1. $M[AR] \leftarrow Rx[7:0], AR \leftarrow AR + 1$
  2. $M[AR] \leftarrow Rx[15:8]$
- MOVL
  1. $DSTReg[7:0] \leftarrow Address$
- ADD
  1. the case where SReg1 and Sreg2 from ARF:
     (a) $S1 \leftarrow SReg1$
     (b) $S2 \leftarrow SReg2$
     (c) $DstReg \leftarrow S1 + S2$
  2. the case where SReg1 from ARF, SReg2 from RF:
     (a) $S1 \leftarrow SReg1$
     (b) $DstReg \leftarrow S1 + SReg2$
  3. the case where SReg1 from RF, SReg2 from ARF:
     (a) $S1 \leftarrow SReg2$
     (b) $DstReg \leftarrow SReg1 + S1$
  4. the case where SReg1 and Sreg2 from RF:
     (a) $DstReg \leftarrow SReg1 + SReg2$

- ADC
  1. the case where SReg1 and Sreg2 from ARF:
     (a) $S1 \leftarrow SReg1$
     (b) $S2 \leftarrow SReg2$
     (c) $DstReg \leftarrow S1 + S2 + Carry$
  2. the case where SReg1 from ARF, SReg2 from RF:
     (a) $S1 \leftarrow SReg1$
     (b) $DstReg \leftarrow S1 + SReg2 + Carry$
  3. the case where SReg1 from RF, SReg2 from ARF:
     (a) $S1 \leftarrow SReg2$
     (b) $DstReg \leftarrow SReg1 + S1 + Carry$
  4. the case where SReg1 and Sreg2 from RF:
     (a) $DstReg \leftarrow SReg1 + SReg2 + Carry$
- SUB
  1. the case where SReg1 and Sreg2 from ARF:
     (a) $S1 \leftarrow SReg1$
     (b) $S2 \leftarrow SReg2$
     (c) $DstReg \leftarrow S1 - S2$
  2. the case where SReg1 from ARF, SReg2 from RF:
     (a) $S1 \leftarrow SReg1$
     (b) $DstReg \leftarrow S1 - SReg2$
  3. the case where SReg1 from RF, SReg2 from ARF:
     (a) $S1 \leftarrow SReg2$
     (b) $DstReg \leftarrow SReg1 - S1$
  4. the case where SReg1 and Sreg2 from RF:
     (a) $DstReg \leftarrow SReg1 - SReg2$
- MOVS
  1. $DstReg \leftarrow SReg1$
- The ADDS, SUBS, ANDS, ORRS, XORS instructions applied like the above ones. Only the S bit controls whether the flags will change or not.

- BX
  1. $M[SP] \leftarrow PC[7:0], SP \leftarrow SP + 1$
  2. $M[SP] \leftarrow PC[15:8]$
  3. $PC \leftarrow Rx$
- BL
  1. $PC[7:0] \leftarrow M[SP], SP \leftarrow SP + 1$
- LDRIM
  1. $Rx \leftarrow Address$
- STRIM
  1. $S1 \leftarrow AR$
  2. $S2 \leftarrow Address$
  3. $AR \leftarrow S1 + S2$
  4. $M[AR] \leftarrow Rx[7:0], AR \leftarrow AR + 1$
  5. $M[AR] \leftarrow Rx[15:8]$

3 RESULTS

3.1 Simulation and Testing

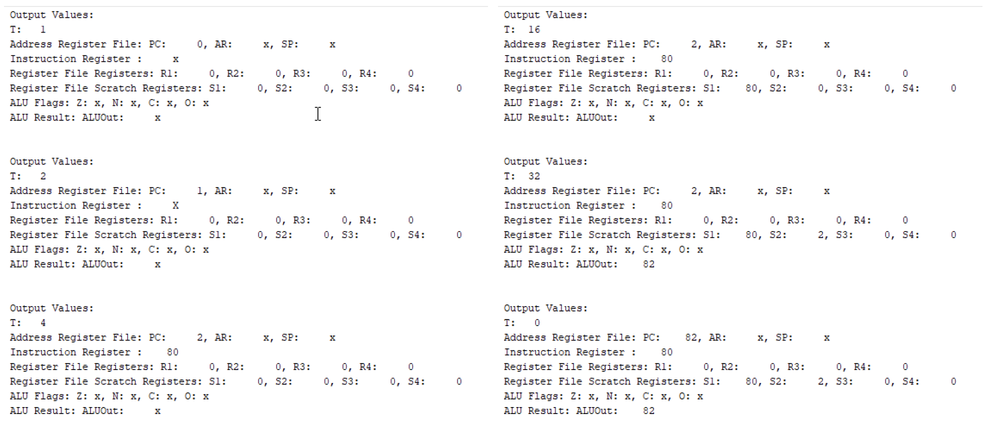
After implementing CPUSystem module, we conducted simulations using Vivado to verify its functionality. In order to fully verify the CPUSystem's functionality, we made modifications to the ram.mem file to give the CPU the right instructions across a variety of tasks. Instructions for branching, incrementing, and other related operations were included in this. We verified that the CPU system functioned as intended by the design specifications by emulating these instructions and observing the result.

3.2 Simulation Results

We conducted simulations using Vivado to verify the functionality of the CPUSystem module. We modified the ram.mem file to provide the CPU with instructions for various tasks, including BRA and INC. Below are example outputs for these operations:

3.2.1 Branch Unconditionally (BRA)

When the CPU encounters a branching instruction, it should jump to a new address based on a condition. For example, when the condition is met, the CPU should jump to a new address. In the Figure 1, output for a branching instruction where the PC is updated to PC + VALUE:



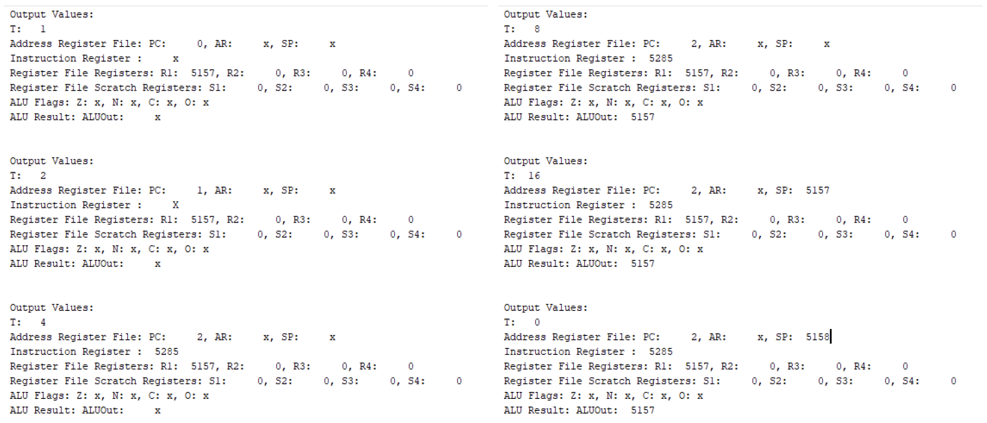
```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    80
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

```
Output Values:
T:  16
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    80
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    80, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:  32
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    80
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    80, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    82


Output Values:
T:   0
Address Register File: PC:    82, AR:     x, SP:     x
Instruction Register :    80
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    80, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    82
```

Figure 1: BRA operation

3.2.2 Increment Operation (INC)

The increment operation (INC) selects a register from SReg1, without incrementing its value, writes its value incremented by 1 in a selected register from DstReg. In the Figure 2 , SReg1 is R1 and DstReg is SP:



```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  5285
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  5285
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:  5157


Output Values:
T:  16
Address Register File: PC:     2, AR:     x, SP:  5157
Instruction Register :  5285
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:  5157


Output Values:
T:   0
Address Register File: PC:     2, AR:     x, SP:  5158
Instruction Register :  5285
Register File Registers: R1:  5157, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:  5157
```

Figure 2: INC operation

3.2.3 Elaborated Design

The design of the CPU System can be shown below in Figure 3. We wanted to upload it despite its complicatedness.

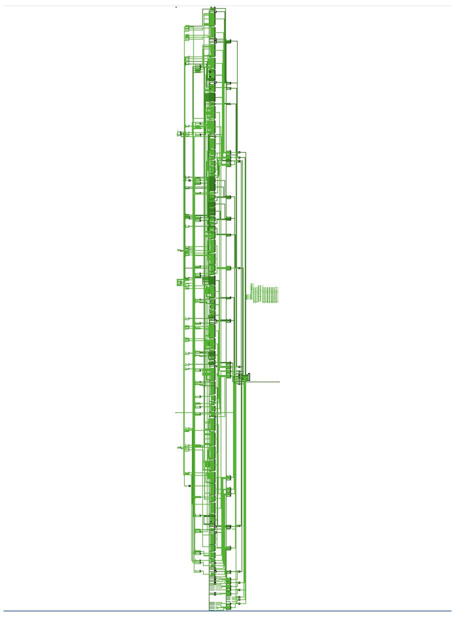


Figure 3: Elaborated Design

4 DISCUSSION

In this project, building on the framework created in Project 1's Part 4, we built and implemented a hardwired control unit for a particular architecture. The control unit was in charge of retrieving instructions from memory, interpreting them, and carrying out the necessary actions. In order to accomplish this, we incorporated the instruction structure and behavior as specified for the architecture into the Verilog HDL description of the control logic. Our approach concentrated on ensuring that instructions were appropriately decoded and executed according to their type (with or without address reference), handling little-endian order, and correctly loading instructions from memory. In order to ensure correct synchronization with the instruction execution phase, we also took into account the two-clock cycle instruction loading process as outlined in the project standards.

We learned a great deal about the architecture and operation of CPU control units from this research. We improved our comprehension of the fundamentals of computer organization, especially as they relate to the execution of instructions and control flow management. We were able to successfully demonstrate the functioning and accuracy of our hardwired control unit by following the project criteria and conducting extensive testing on our design. All in all, this project improved our knowledge of digital design and execution by giving us hands-on experience with Verilog HDL and computer organization concepts.

5 CONCLUSION

Building on the framework established in Project 1, we set out to construct a hardwired control unit for a particular architecture in this project. The project had a number of difficulties, especially when it came to putting the instruction loading procedure into practice and making sure it was properly synchronized with the instruction execution phase. Keeping the two-clock cycle instruction loading procedure under control, which necessitated close attention to detail to guarantee proper operation, was one of the main problems. Another major challenge was creating the control unit to accurately interpret instructions with and without address reference. For the CPU system to work as a whole, it was essential that the control unit could read the opcode and other instruction fields accurately in accordance with the requirements of the architecture. We overcame these obstacles by utilizing our understanding of computer organization concepts and Verilog HDL throughout the project. Working together with classmates and utilizing internet tools, such the Ninova message board, were really helpful in solving problems and deepening our comprehension.

To sum up, this project gave us hands-on experience developing and putting into use a hardwired control unit for a CPU system. It confirmed what we already knew about the fundamentals of computer structure and the value of careful testing and design. All things considered, the project played a significant role in expanding our understanding of digital design and how it is used in practical systems.

REFERENCES