ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION PROJECT 2 REPORT

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

GROUP MEMBERS:

150210060	:	MEHMET MERT ERGIN
150210055	:	TAHA TEMIZ

SPRING 2024

Contents

1 INTRODUCTION

We designed a hardwired control unit for the given architecture. We used the structure that we have designed in Part 4 of Project 1. We started the project by calculating how many clock cycles the operations requested from us would take place. We analyzed every operation that was executed in possible clock cycles for every opcode. While making our calculations, we also took into account the loading of the instruction register in the project at 2 clock cycles.

1.1 TASK DISTRIBUTION

Since we were roommates, we had the opportunity to complete all the parts of the project together. We did not distribute a specific task. We were together at all stages.

2 METHODS AND ANALYSIS

2.1 TIMING

In this project, a structure was needed to control the time. We designed a basic counter. This counter has an enable input, a reset input, a count input, and a three-bit output. The counter works with the clock signal. If the count input is enabled, it counts. If the reset input is enabled, it resets its own value. Moreover, we designed a three-to-eight decoder. Temporary data is collected from the counter. Then the temporary data is sent to the SC decoder. The output of the SC decoder is T. T is an 8-bit value. So we can reach T[x] values. The schematics of the counter and the SC decoder can be seen below.

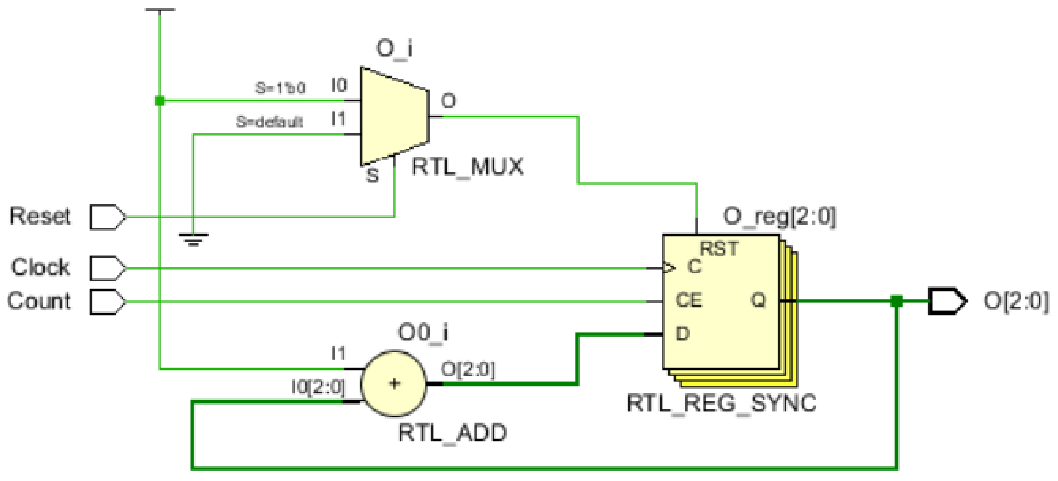


Figure 1: Counter

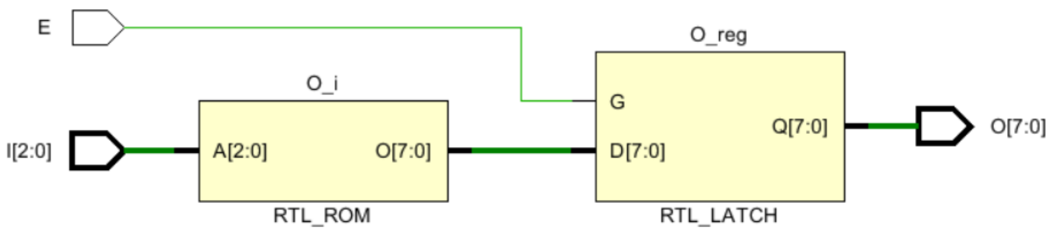


Figure 2: SC Decoder

2.2 FETCH AND DECODE

The instruction register has a 16-bit value. The size of the data in the memory is 8 bits. Fetching lasts for two clock cycles. In T[0], the LSB of the IR is loaded from memory. In T[1], the MSB of the IR is loaded from memory. Our control unit sets IR-Write as 1 and adjusts the IR-LH according to Tx. In T[0] and T[1], the control unit increments PC by one after reading the memory.

There are 34 operations. So, we designed a 6-to-64 decoder to decode the opcodes. In T[2], the control unit decodes the opcode and assigns the values for SREG1, SREG2, S, DSTREG, and RSEL. The schematic of the instruction register can be seen below. So, we can check Dx for operations. According to the value of D, the control unit assigns the function to the ALU.

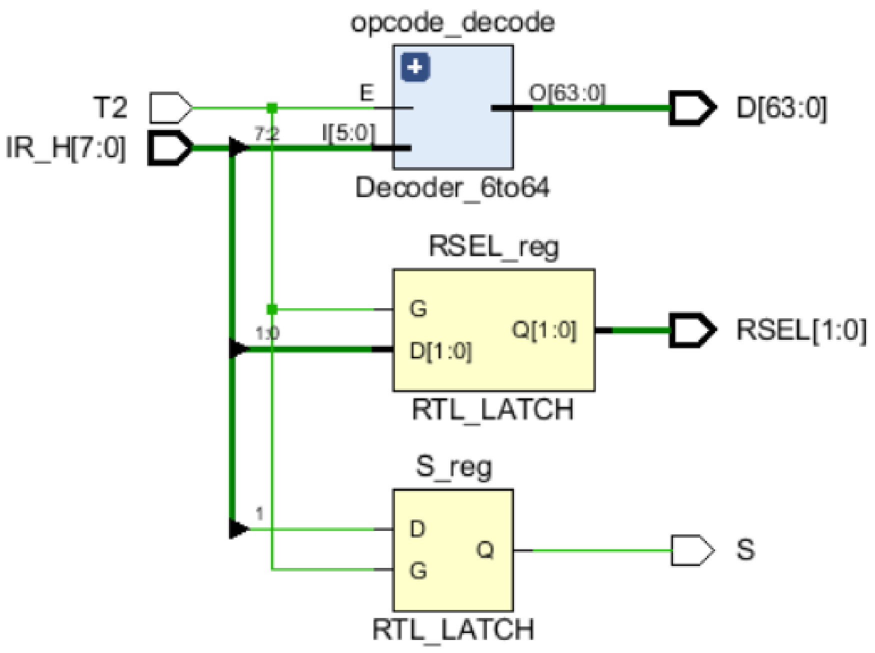


Figure 3: Instruction Register

2.3 BASIC OPERATIONS WITH ONE SOURCE

INC, DEC, LSL, LSR, ASR, CSL, CSR, NOT,and MOVS are the basic operations with one source. The number of clock cycles can be changed according to SREG1. If SREG1 is in RF, we can execute the operation in one clock cycle, and it ends after T2 (except INC and DEC). If SREG1 is in ARF, we need to store the data in S1 in RF. So, we need one more clock cycle.

INC and DEC operations need two clock cycles to be completed. In T2, SREG1 is loaded into DSTREG. In T3, the value of DSTREG is incremented or decremented, respectively.

2.4 BASIC OPERATIONS WITH TWO SOURCES

AND, ORR, XOR, NAND, ADD, ADC, SUB, ADDS, SUBS, ANDS, ORRS, and XORS are the basic operations with two sources. According to the combination of the source locations, the number of clock cycles is changing. If SREG1 and SREG2 are in RF, executing ends at the end of T2. If just one of them is from ARF, executing ends at the end of T3. If SREG1 and SREG2 are both in ARF, executing ends at the end of T4. Because one more clock cycle is needed to store the data from the ARF into S2 in the RF.

If the flag can be changed, the control unit assigns the S value to the write flag of the ALU. The only thing that differs from other operations is that.

2.5 SPECIAL OPERATIONS

BRA, BNE, BEQ, POP, PSH, MOVH, LDR, STR, MOVL, BX, BL, LDRIM, and STRIM are the special operations. We wanted to call these operations special. Because they are more complicated than the basic ALU operations.

2.5.1 BRA

D0T2: RF.S1 $\leftarrow$ PC
D0T3: RF.S2 $\leftarrow$ VALUE
D0T4: PC $\leftarrow$ ALUOut(PC+VALUE)

In T2, ALU function is ADD, MuxAOut is ARFOutC, RF function is load, and S1 in RF is enabled.

In T3, MuxAOut is IROut[7:0], RF function is load, and S2 in RF is enabled.

In T4, OutA is S1, OutB is S2, MuxBOut is AluOut, PC is enabled, and ARF function is load.

2.5.2 BNE

BNE is the same as the BRA. But, first we need to check if Z is 1 or not. If Z is 0, then we can execute.

2.5.3 BEQ

BNE is the same as the BRA. But, first we need to check if Z is 1 or not. If Z is 1, then we can execute.

2.5.4 POP

D3T2: SP ← SP + 1, Rx[7:0] ← M[SP]
D3T3: SP ← SP + 1, Rx[15:8] ← M[SP]

Registers in RF are 16 bit. When we are executing the POP operation, we need to load the data from the memory part by part.

In T2, ARF function is increment, SP is enabled, selected register in the register file is enabled, register file function is only write low, MuxAOut is MemOut.

In T3, ARF function is increment, SP is enabled, selected register in the register file is enabled, register file function is only write high, MuxAOut is MemOut.

2.5.5 PSH

D4T2: SP ← SP - 1, M[SP] ← Rx[15:8]
D4T3: SP ← SP - 1, M[SP] ← Rx[7:0]

Registers in RF are 16 bit. When we are executing the PSH operation, we need to load the data from the Rx to memory part by part.

In T2, ARF function is decrement, SP is enabled, OutA is Rx, ALU function is A, MuxCOut is ALUOut[15:8], the write signal of the memory is enabled,

In T3, ARF function is decrement, SP is enabled, OutA is Rx, ALU function is A, MuxCOut is ALUOut[7:0], the write signal of the memory is enabled.

2.5.6 MOVH

D17T2: DSTREG[15:8] $\leftarrow$ IROut[7:0]

In T2, MuxAOut is IROut[7:0], selected register in register file is enabled,and register file function is load.

2.5.7 LDR

D18T2: AR $\leftarrow$ AR + 1, Rx[7:0] $\leftarrow$ M[AR]
D18T3: AR $\leftarrow$ AR - 1, Rx[15:8] $\leftarrow$ M[AR]

In T2, ARF function is increment, AR is enabled, MuxAOut is MemOut, selected register in register file is enabled,and register file function is only write low.

In T3, ARF function is decrement to keep the AR value the same as before the execution. AR is enabled, MuxAOut is MemOut, selected register in register file is enabled, and register file function is only write high.

2.5.8 STR

D19T2: AR $\leftarrow$ AR + 1, M[AR] $\leftarrow$ Rx[7:0]
D19T3: AR $\leftarrow$ AR - 1, M[AR] $\leftarrow$ Rx[15:8]

In T2, ARF function is increment, AR is enabled, ALU function is A, MuxCOut is ALUOut[7:0], write signal of the memory is enabled, OutD is AR, and OutA is the selected register in register file.

In T3, ARF function is decrement, AR is enabled, ALU function is A, MuxCOut is ALUOut[15:8], write signal of the memory is enabled, OutD is AR, and OutA is the selected register in register file. We are incrementing and decrementing the AR to keep the value same as before execution.

2.5.9 MOVL

D20T2: DSTREG[7:0] $\leftarrow$ IROut[7:0]

In T2, MuxAOut is IROut[7:0], register file function is only write low, selected register in register file is enabled.

2.5.10 BX

D30T2: RF.S1 ← PC, PC ← Rx

D30T3: SP ← SP - 1, M[SP] ← RF.S1[15:8]

D30T4: SP ← SP - 1, M[SP] ← RF.S1[7:0]

First, we store the PC value into a scratch register. Then, we write the stored data into memory part by part.

In T2, register file function is load, ARFOutC is PC, ARF function is load, PC is enabled, S1 in RF is enabled, RFOutA is Rx MuxBOut ALUOut, ALU function is A, and MuxAOut is ARFOutC.

In T3, SP is enabled, ARF function is decrement, write signal of memory is enabled, ARFOutD is SP, RFOutA is S1, and MuxCOut is ALUOut[15:8].

In T4, SP is enabled, ARF function is decrement, write signal of memory is enabled, ARFOutD is SP, RFOutA is S1, and MuxCOut is ALUOut[7:0].

2.5.11 BL

D31T2: SP ← SP + 1, RF.S1[7:0] ← M[SP]

D31T3: SP ← SP + 1, RF.S1[15:8] ← M[SP]

D31T4: PC ← RF.S1

We load the data from the memory to S1 part by part, then we load the S1 to PC. Because memory has 8-bit data but registers have 16-bit value.

In T2, SP is enabled, ARF function is increment, register file function is only write low, S1 in register file is enabled, MuxAOut is MemOut.

In T3, SP is enabled, ARF function is increment, register file function is only write high, S1 in register file is enabled, MuxAOut is MemOut.

In T4, PC is enabled, ARF function is load, RFOutA is S1, ALU function is A, MuxBOut is ALUOut,

2.5.12 LDRIM

D32T2: Rx ← IROut[7:0](Sign Extended)

We decided to use sign extension for this operation. Data is 8-bit but Rx is 16-bit.

In T2, selected register in register file is enabled, register file function is write low with sign extension, MuxAOut is IROut[7:0].

2.5.13 STRIM

D33T2: RF.S1 ← AR
D33T3: RF.S2 ← IROut[7:0]
D33T4: AR ← ALUOut(S1 + S2)
D33T5: M[AR] ← Rx[7:0], AR ← AR + 1
D33T6: M[AR] ← Rx[15:8], AR ← AR - 1

First, we need to calculate the sum of AR and OFFSET value. So we store them in the scratch registers. Then load into AR. The data in the memory is 8-bit. So we need to write Rx part by part. We incremented and decremented AR by one to keep the AR value same.

In T2, S1 register in register file is enabled, register file function is load, MuxAOut is ARFOutC, ARFOutC is AR.

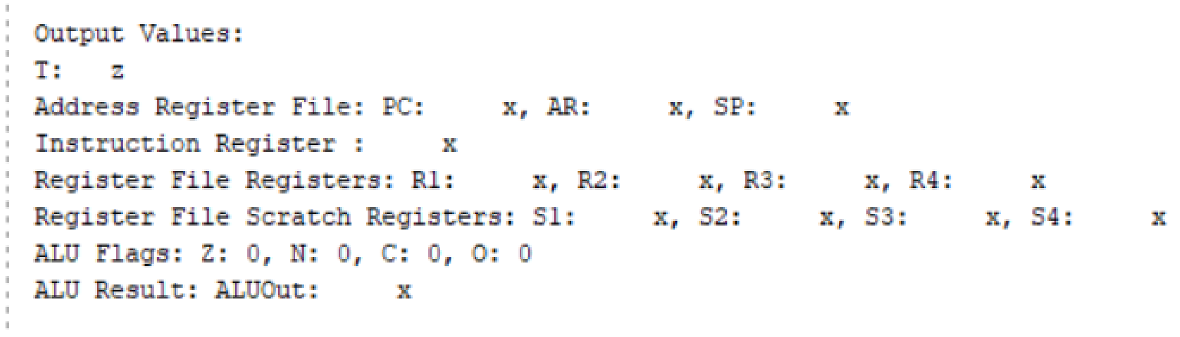
In T3, S2 register in register file is enabled, register file function is load, MuxAOut is IROut[7:0].

In T4, AR is enabled, ARF function is load, ALU function is ADD, RFOutA is S1, RFOutB is S2, MuxBOut is ALUOut.

In T5, write signal of memory is enabled, MuxCOut is ALUOut[7:0], ALU function is A, RFOutA is Rx, ARFOutD is AR, ARF function is increment, AR is enabled.

In T6, write signal of memory is enabled, MuxCOut is ALUOut[15:8], ALU function is A, RFOutA is Rx, ARFOutD is AR, ARF function is decrement, AR is enabled.

3 RESULTS



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

Figure 4: Simulation Results

Our design is mistaken, we did not get proper outputs.

4 DISCUSSION

In this project, we tried to design a hardwired control unit. We designed a counter module, two decoder modules, and a control unit module. We analyzed all of the given operations with possible scenarios. The memory has 8-bit data, and all of the registers have 16-bit values. This was a problem for some operations. So, we sometimes separate operations, and we sometimes do extensions for the MSB parts. We think that the control conditions of our control unit are correct. But we think there is a problem with connecting wires between the ALU system, the control unit, and the CPU system. We could not get proper outputs. Our design is probably mistaken. We tried to find the problem and solve it. It was not successful. Although our design did not work correctly, we explained what we thought while writing our codes by using comment lines as much as possible.

5 CONCLUSION

Although Project 2 was a continuation of Project 1, it was more challenging than Project 1. We had a hard time understanding the working logic of some operations, for example, pop and push. We did not have a lot of test files related to the project, so it was challenging for us to test the accuracy of what we were doing. Some of the operations in the operation table given to us were not clear, so we did extra research for the operations.The most difficult part for us was that there was no direct connection from ARF to ALU. This meant that the processing time was extended in operations with ARF. Another problem was connecting the control unit module with the ALU system module in the CPU system module. We think our control mechanism and timing algorithm in the control unit are correct. But we could not make the design work as expected. We could not get the expected output values. Although it was challenging, we learned how to design a hardwired control unit throughout the project.