ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT 2 REPORT

CRN : 21334 / 21336

LECTURER : Gökhan İnce / M. Ersel Kamaşak

GROUP MEMBERS:

150230736 : Umut Özil (Group Representive)

150230737 : Ege Keklikçi

SPRING 2024

Contents

1 INTRODUCTION [10 points]

In this project, we designed hardwired control unit for the ALUSytem that we created in the first project. The task of the control unit that we created is fetching instructions from memory one by one and executing them. There are two types of instructions, instructions with an address reference and Instructions without an address reference.

1.1 Task Distribution

- Fetching, decoding and instructions without an address reference was done by Ege Keklikçi.
- Instructions with an address reference was done by Umut Özil.

And debugging the whole system and writing the report were done by both of us. Also, we discussed each others parts and exchanged ideas about our parts.

2 MATERIALS AND METHODS [40 points]

2.1 Fetching

Although our registers are 16 bits, we cannot perform the fetch operation in a single clock cycle because the word length of memory is 8-bits. Consequently, we performed the fetch operation in two clock cycle by loading LSB 8-bits in T0 and loading MSB 8-bits T1 to instruction register (IR).

2.2 Decoding

Decoding instructions were not mandatory, but we aimed to enhance code readability by avoiding expressions like "_ALUSystem.ARF.IROut[7:0]" during execution. Therefore, we stored the reserved bits (OPCODE, S, RSEL, DSTREG, SREG1, SREG2) of the instructions in 'reg's in the start of T2.

2.3 Execution

When writing the execution codes for the instructions, we segmented operations into Verilog Tasks, aiming to minimize code complexity and to avoid writing the same procedures over and over again. Using Tasks has made our code much simpler, especially for the instructions without an address reference.

2.3.1 Execution of Instructions without an Address Reference

Instructions without an address reference consist of the DSTREG where the result will be written, along with the operands SREG1 and SREG2. Also, it contains an S bit to decide whether the ALU flags will be changed or not.

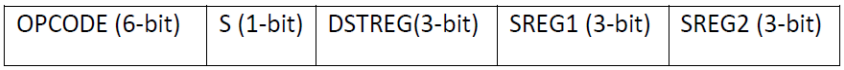

Figure 1: Instruction without an adress reference

DSTREG, SREG1 and SREG2 register selections were made according to the table below.

DSTREG /SREG1 /SREG2	**REGISTER**
000	PC
001	PC
010	SP
011	AR
100	R1
101	R2
110	R3
111	R4

Table 1: DSTREG/SREG1/SREG2 selection table

- **INC, DEC:** During the T2 clock cycle, we executed DSTREG ← SREG1 operation using "inc_dec_1" task. Subsequently, during the T3 clock cycle, we performed the DSTREG ← DSTREG +/- 1 operation using "inc_dec_2" task based on the specific operation being carried out.

- **LSL, ASR, CSL, CSR, NOT:** Since the procedure for executing these operations is exactly the same except for the FunSel to be given to the ALU, we used the same tasks.

  During the T2 clock cycle, we executed DSTREG ← SREG1 operation using "one_sreg_1" task. Subsequently, during the T3 clock cycle, we performed the chosen operation using "one_sreg_2" task and we utilized the ALU to perform operations.

- **MOVS:** Since the flags need to change depending on the S bit in the MOVS instruction, during T2 we called the "one_sreg_1" task and loaded S1 with SREG1 if SREG1 was in ARF. Subsequently, at T3, we called the "one_sreg_2" task and if SREG1 was in ARF, we passed S1 through the ALU; if it was in RF, we passed SREG1 through the ALU and loaded it into DSTREG.

- **AND, ORR, XOR, NAND, ADD, ADC, SUB, ADDS, SUBS, ANDS, ORRS, XORS:** Since the procedure for executing these operations is exactly the same except for the FunSel to be given to the ALU, we used the same tasks.

  During the T2 clock cycle, we called the "two_sreg_1" task to load S1 with SREG1 if SREG1 was in ARF. As the clock transitioned to T3, the "two_sreg_2" task was called to load S2 with SREG2 if SREG2 was in ARF. Subsequently, at T4, we concluded the execution process by invoking the "two_sreg_3" task based on the selected operation and the S bit.

  Although there were scenarios where both SREG1 and SREG2 were in RF, we did not prefer to add additional controls to complete the execution in a single clock cycle to maintain code simplicity and enhance readability.

2.3.2 Execution of Instructions with an Address Reference

Instructions with an address reference consist of the RSEL where the result will be written, along with the Address bits that reference the address in memory.

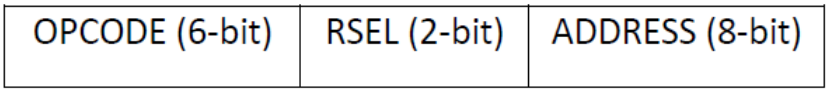

Figure 2: Instruction with an adress reference

RSEL register selection were made according to the table below.

RSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

Table 2: RSEL selection table

- **BRA, BNE, BEQ:** To execute the PC + VALUE operation within branch operations, we first loaded the PC to S1 scratch register in T2.

  Then, in T3 we loaded the VALUE from the Address bits in the instruction into the S2 scratch register by extending its sign (we assumed that VALUE is signed).

  Finally, using the ALU at T4, we performed the PC + VALUE operation and loaded the result into the PC.

  Also, in BNE and BEQ operations, we checked whether the Z flag at T2 met the condition and started the execution accordingly.

- **POP:** During the POP operation, in T2, we made SP to point the value at the top of the stack by performing SP $\leftarrow$ SP + 1, since SP points to the first free space in the stack.

  Then we accessed memory address SP+1 in T3, fetched the value stored there, and loaded it into LSB bit 8 of the RSEL register. Simultaneously, within the same clock cycle, we executed the operation SP $\leftarrow$ SP + 1. This simultaneous execution result no issue since reading data from Memory doesn't rely on the clock.

  Finally, in T4, we retrieved the value at Memory's SP+2 address and loaded it into the MSB 8-bits of RSEL.

- **PUSH:** During the PUSH operation, in T2, we stored the MSB 8-bit of the RSEL to the SP address of the Memory, and since we would also save the LSB 8-bit to the memory, we performed the SP ¡- SP - 1 operation.

  Finally, in T3, we stored the LSB 8-bit of RSEL in the SP-1 address of the Memory. Simultaneously, we performed the SP ¡- SP - 1 operation because SP should point to the first free space on the stack.

- **MOVH, MOVL:** During MOVH and MOVL operations, in T2, we loaded the value from the address bit of the instruction fetched into either the MSB 8-bits or LSB 8-bits of the RSEL register, based on the operation being performed.

- **LDR:** During the LDR operation, in T2, we loaded the LSB 8-bits from the AR address of the memory to the RSEL, and since we would also load the MSB 8-bits from the memory to the RSEL, we performed the AR $\leftarrow$ AR + 1 operation.

  Then, in T3, we loaded the MSB 8-bits from the AR+1 address of the memory to the RSEL and at the same time performed the AR $\leftarrow$ AR - 1 operation to protect the address inside AR.

- **STR:** During the STR operation, in T2, we stored the LSB 8 bits of the RSEL to the AR address of the memory, and since we would also save the MSB 8-bit to the memory, we performed the AR $\leftarrow$ AR + 1 operation.

  Then, in T3, we stored the MSB 8-bit of the RSEL to the AR+1 address of the memory and at the same time performed the AR $\leftarrow$ AR - 1 operation to protect the address inside AR.

- **BX:** Branch and save return address operation works similar to the PUSH operation. However, since the PC is in ARF, we cannot directly transfer the data in it to Memory. Therefore, in T2, we loaded the PC into the S1 scratch register. In T3 and T4, we applied the same procedures as for PUSH. Finally, We loaded the value in RSEL into the PC in T5.

- **BL:** The BL operation operates similarly to the POP operation. However, due to both PC and SP being in ARF, simultaneous execution of PC ¡- M[SP] and SP ¡- SP + 1 is not possible. Thus, we initially stored the data from Memory pointed by SP into the S1 scratch register, following the same process as the POP operation, during T2 and T3. Subsequently, at T4, we loaded the value from S1 into the PC.

- **LDRIM:** During the LDRIM operation, we loaded the 8-bit data in the address bits of the instruction into the RSEL in T2.

- **STRIM:** To execute the AR + OFFSET operation within the STRIM operation, we first loaded the data from AR into the S1 scratch register at T2.

  Subsequently, we loaded the OFFSET data from the address bits of the instruction into the S2 scratch register at T3.

  Then, at T4, we added the values of S1 and S2 using ALU and stored the result back into AR.

  Finally, during T5 and T6, we stored the LSB 8 bits of RSEL to AR (AR+OFFSET) and the MSB 8 bits of RSEL to AR+1 (AR+OFFSET+1) in memory respectively.

3 RESULTS [15 points]

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
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: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
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: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :     8
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: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :     8
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:  16
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :     8
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     8, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:    10


Output Values:
T:   1
Address Register File: PC:    10, AR:     0, SP:   255
Instruction Register :     8
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     8, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:    10
```

Figure 3: Results of BRA 0x08

```
T:   1
Address Register File: PC:     6, AR:     x, SP:    25
Instruction Register :  5408
Register File Registers: R1:    16, R2:     x, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    16

Output Values:
T:   2
Address Register File: PC:     7, AR:     x, SP:    25
Instruction Register :  5472
Register File Registers: R1:    16, R2:     x, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    16


Output Values:
T:   4
Address Register File: PC:     8, AR:     x, SP:    25
Instruction Register :  6496
Register File Registers: R1:    16, R2:     x, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    16


Output Values:
T:   8
Address Register File: PC:     8, AR:     x, SP:    25
Instruction Register :  6496
Register File Registers: R1:    16, R2:    16, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    16


Output Values:
T:   1
Address Register File: PC:     8, AR:     x, SP:    25
Instruction Register :  6496
Register File Registers: R1:    16, R2:    15, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
```

Figure 4: Results of DEC R2 R1

```
T:   1
Address Register File: PC:     8, AR:     x, SP:   255
Instruction Register : 20745
Register File Registers: R1:     6, R2:     9, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     9, AR:     x, SP:   255
Instruction Register : 20908
Register File Registers: R1:     6, R2:     9, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:    10, AR:     x, SP:   255
Instruction Register : 15788
Register File Registers: R1:     6, R2:     9, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   8
Address Register File: PC:    10, AR:     x, SP:   255
Instruction Register : 15788
Register File Registers: R1:     6, R2:     9, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:  16
Address Register File: PC:    10, AR:     x, SP:   255
Instruction Register : 15788
Register File Registers: R1:     6, R2:     9, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    15


Output Values:
T:   1
Address Register File: PC:    10, AR:     x, SP:   255
Instruction Register : 15788
Register File Registers: R1:     6, R2:     9, R3:    15, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
```

Figure 5: Results of XOR R3 R1 R2

```
Output Values:
T:   1
Address Register File: PC:     4, AR:     0, SP:   255
Instruction Register : 20486
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   2
Address Register File: PC:     5, AR:     0, SP:   255
Instruction Register : 20480
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     6, AR:     0, SP:   255
Instruction Register :  4096
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:   8
Address Register File: PC:     6, AR:     0, SP:   254
Instruction Register :  4096
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:   1
Address Register File: PC:     6, AR:     0, SP:   253
Instruction Register :  4096
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6
```

(a) Results of PUSH R1

```
Output Values:
T:   1
Address Register File: PC:     6, AR:     0, SP:   253
Instruction Register :  4096
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:   2
Address Register File: PC:     7, AR:     0, SP:   253
Instruction Register :  4096
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:   4
Address Register File: PC:     8, AR:     0, SP:   253
Instruction Register :  3328
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:   8
Address Register File: PC:     8, AR:     0, SP:   254
Instruction Register :  3328
Register File Registers: R1:     6, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6


Output Values:
T:  16
Address Register File: PC:     8, AR:     0, SP:   255
Instruction Register :  3328
Register File Registers: R1:     6, R2:     6, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:     6
```

(b) Results of POP R2

Figure 6: Results of consecutive push and pop operations

```
Output Values:
T:   1
Address Register File: PC:    10, AR:    16, SP:   255
Instruction Register : 25312
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:    16


Output Values:
T:   2
Address Register File: PC:    11, AR:    16, SP:   255
Instruction Register : 25088
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut:    16


Output Values:
T:   4
Address Register File: PC:    12, AR:    16, SP:   255
Instruction Register : 19712
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535


Output Values:
T:   8
Address Register File: PC:    12, AR:    16, SP:   256
Instruction Register : 19712
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535
```

(a) Results of STR R2

```
Output Values:
T:   1
Address Register File: PC:    12, AR:    16, SP:   255
Instruction Register : 19712
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535


Output Values:
T:   2
Address Register File: PC:    13, AR:    16, SP:   255
Instruction Register : 19712
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535


Output Values:
T:   4
Address Register File: PC:    14, AR:    16, SP:   255
Instruction Register : 18944
Register File Registers: R1:    16, R2: 65535, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535


Output Values:
T:   8
Address Register File: PC:    14, AR:    16, SP:   256
Instruction Register : 18944
Register File Registers: R1:    16, R2: 65535, R3:   255, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535


Output Values:
T:   1
Address Register File: PC:    14, AR:    16, SP:   255
Instruction Register : 18944
Register File Registers: R1:    16, R2: 65535, R3: 65535, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, C: 0, N: 0, O: 0
ALU Result: ALUOut: 65535
```

(b) Results of LDR R3

Figure 7: Results of consecutive str and ldr operations

4 DISCUSSION [25 points]

In this project we implemented a hardwired unit by adding on top of our previous assignment, arithmetic logic unit. Control unit decodes and runs the proper codes and tasks with the help of the sequence counter in order to achieve the intended result. The sequence counter in the control unit determines which tasks are being ran. For the sake of complexity reduction, throughout the code we used tasks (functions) to not write what we have already written. Tasks are separated by the sequence counters current status. In our design sequence counter is checked by an always block at posedge. Fetch process lasts two clock cycles. Decode process does not need additional clock cycle. After these process resetSel() function is called to disable the registers in ARF and RF. After calling resetSel() the desired opcode is executed by the help of case keyword.

We've had some problems while storing data to memory. Also, the hardest part of our implementation was about stack pointer. Our concerns were about if the sp was going to point to the data or the free space after the data. Finally, we implemented our system to point to the free space. In the end we have solved all of our problems.

5 CONCLUSION [10 points]

While this project was very informative and taught us a lot, it was also very challenging and we had some trouble coding and designing.

At first, we were writing separate codes for each operation without using the tasks. Then we realized that we were repeating the same codes in some operations, and at that moment we thought that by dividing our code into tasks, we could reduce these code repetitions and increase the readability of the code. And as we expected, our final code is much more readable than the first version.

We struggled the most with writing data to memory and reading data from memory in the project. However, thanks to the extension of the project deadline, we overcame this problem. We also noticed that some of our modules in the first project were working incorrectly and we fixed them as well.

As a result, this project was a difficult but instructive assignment that both summarized what we learned throughout the course and enabled us to put our theoretical knowledge into practice.