ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : FINAL

DUE DATE : 14.07.2020

GROUP NO : G26

GROUP MEMBERS:

150170725 : FATİH UZUNOĞLU

150170041 : UZAY DALYAN

150170068 : BERDAN ÇAĞLAR AYDIN

150170071 : YİĞİT EMRE YILMAZ

SPRING 2020

Contents

1 INTRODUCTION

As the final project, we were asked to design a *microprogrammed control unit* that is capable of executing instructions listed below. This control unit works on 16-bit fixed size instructions.

- LD (LOAD)
- ST (STORE)
- MOV (MOVE)
- PSH (PUSH)
- PUL (PULL)
- ADD
- SUB (SUBTRACT)
- DEC (DECREMENT)
- INC (INCREMENT)
- AND
- OR
- NOT
- LSL (LOGICAL SHIFT LEFT)
- LSR (LOGICAL SHIFT RIGHT)
- BRA (BRANCH)
- BEQ (BRANCH IF EQUAL)
- BNE (BRANCH IF NOT EQUAL)
- CALL
- RET (RETURN)

Logisim (v3.3.4 / Logisim-Evolution) was used for the implementations since the original logisim caused problems in our computers because of it being very old and not maintained for a very long time. Note that it requires the latest version of Java runtimes to run. Please read README.txt if you have troubles opening the project.

2 PROJECT PARTS

2.1 Microprogram Structure

Before designing the circuit, we first designed a structure for the control unit microcode. We created a structure which has 8-bit Address Bit Width and 48-bit Data Bit Width.

We found out that the ideal microprogram structure that is able to execute the required operations is as given:

- Address Field (8-bit)
  - 5-bit OpCode Field
    This field denotes which instruction to run.
  - 3-bit Time Field
    Since not all instructions are completed in only one clock cycle, we need a time bit-field to point out at which step we are in during a instruction execution.
- Content Field (48-bit)
  - 3-bit ARegSel Field
    This field contains 3-bit of information for indirectly forwarding necessary signal to Address Register File's RegSel input.
  - Enable(3-bit) + 2-bit AFunSel Field
    This field contains 3-bit of Enable information and 2-bit actual FunSel content for Address Register File's FunSel input. Enable is discussed later.
  - 3-bit RegSel Field
    This field contains 3-bit of information for indirectly forwarding necessary signal to General Purpose Register File's RegSel input.
  - Enable(3-bit) + 2-bit FunSel Field
    This field contains 3-bit of Enable information and 2-bit actual FunSel content for General Purpose Register File's FunSel input.
  - 3-bit OutASel Field
    This field has 3-bit of information for indirectly forwarding necessary signal to OutASel input of General Purpose Register File.
  - 3-bit OutBSel Field
    This field has 3-bit of information for indirectly forwarding necessary signal to OutBSel input of General Purpose Register File.

- 3-bit OutCSel Field
  This field has 3-bit of information for indirectly forwarding necessary signal to OutCSel input of Address Register File.
- 3-bit OutDSel Field
  This field has 3-bit of information for indirectly forwarding necessary signal to OutDSel input of Address Register File.
- 2-bit MuxASel Field
  The content of this field directly gets forwarded into Select input of Multiplexer A.
  Edit #09/07/2020 - Fatih: When MuxASel field is 11, 10 gets forwarded into Multiplexer A's select.
- 2-bit MuxBSel Field
  The content of this field directly gets forwarded into Select input of Multiplexer B.
- 2-bit MuxCSel Field
  This field has 2-bit of information for indirectly forwarding necessary signal to Multiplexer C.
- 1-bit IR_SelH Field
  The content of this field directly gets forwarded into H input of Instruction Register.
- 2-bit IR_FunSel Field
  The content of this field directly gets forwarded into FunSel input of Instruction Register.
- 1-bit IR_Enable Field
  The content of this field directly gets forwarded into Enable input of Instruction Register.
- Enable(3-bit) + 4-bit ALUFunSel Field
  This field contains 3-bit of Enable information and 4-bit of actual FunSel content for Arithmetic Logic Unit's FunSel input.
- 1-bit RAM_WriteEnable Field
  The content of this field directly gets forwarded into RAM's Write Enable signal input.
- 1-bit FakeSrc
  The Fake Source field is responsible for altering SRCREG1. It is used for Increment and Decrement operations. These operations use internal FunSel

signal to do increment or decrement so ALU Flags are not set according to these operations. Since we require ALU flags to be updated according to the result of these operations, increment and decrement operations first does increment or decrement on SRCREG1, then they move SRCREG1's content into DSTREG, and finally they move DSTREG into DSTREG to ensure that the result is passed through the ALU and the flags are properly updated. When Fake Source is high, it tricks the control unit to behave as if the DSTREG was SRCREG1. In other words, it discards original SRCREG1 and sets DSTREG as SRCREG1. This effectively gives an ability to move DSTREG to DSTREG even when instruction has a different SrcReg1.

- 1-bit CounterReset Field
  When high, sequence counter is reset. It is used as the last operation that is carried out during instruction execution.

2.1.1 OutSel Forwarding

OutASel, OutBSel, OutCSel, and OutDSel inputs of the organization are fed by the output of this mapping. OutSel Bit Field is the content of regarding instruction's Microcode correspondence.

Table 1: OutSel Mapping

3-bit OutSel Bit Field	**2-bit OutSel Input**
000	00
001	01
010	10
011	Forward instruction's RegSel
100	N/A
101	Forward instruction's SRCREG1_GPR
110	Forward instruction's SRCREG1_AR
111	Forward instruction's SRCREG2_GPR

2.1.2 MuxCSel Forwarding

Select input of Multiplexer C is fed by the output of this mapping. MuxCSel Bit Field is the content of regarding instruction's Microcode correspondence.

Table 2: OutSel Mapping

2-bit MuxCSel Field	**1-bit MuxCSel Select Input**
00	0
01	1
10	0 if SRCREG1 is in AR File, 1 if SRCREG1 is in GPR File
11	N/A

2.1.3 RegSel Forwarding

GPR RegSel and AR RegSel inputs of the organization are fed by the output of this mapping. RegSel/ARegSel Bit Field is the content of regarding instruction's Microcode correspondence.

Table 3: RegSel Mapping

3-bit RegSel/ARegSel Field	**3-bit ARegSel / 4-bit RegSel Input**
000	000 / 0000
001	001 / 0001
010	Forward instruction's SRCREG1_AR
011	Forward instruction's SRCREG1_GPR
100	100 / 0100
101	Forward instruction's RegSel
110	Forward instruction's DSTREG_AR
111	Forward instruction's DSTREG_GPR

2.1.4 Enable Controlling

All FunSel functions in the microcode has 3-bit Enable field which starts from the MSB. The reason of its existence is to do selections based on the operation. For example, MOV instruction may move R1 to AR but it may also move AR to R1. In order to be aware of this and to carefully drive the organization input signals, there is an additional Enable field that is integrated to the FunSel bit fields of the Microcode.

Based on the Enable bit field, content of FunSel gets either forwarded or not forwarded to FunSel inputs of the organization. When it is not forwarded, FunSel input signal becomes high impedance.

The other reason of such 'Enable Controlling' usage is due to the fact that '00' FunSel operation is used for clear operation on registers. So without enable, there

would be no way to not do any operation on registers or on the ALU when the instruction doesn't need to.

Table 4: Enable Controller

3-bit Enable Field	**2-bit RegFunSel / 4-bit ALU FunSel Input**
000	Z (high impedance)
001	Forward content.
010	Forward content if DSTREG is in GPR File
011	Forward content if DSTREG is in AR File
100	Forward content if (instruction is BEQ and Z Flag is 1) or (instruction is BNE and Z Flag is 0)
101	Forward content if (SRCREG1 is in GPR File and DSTREG is in AR File) or (SRCREG1 is in AR File and DSTREG is in AR File) or (SRCREG1 is in GPR File and DSTREG is in GPR File)
110	Forward content if SRCREG1 is in AR File
111	Forward content if SRCREG1 is in GPR File

2.2 The Microcode

A microprogrammed control unit needs a microcode file to operate. This file is generally a binary file consisting of microcodes of instruction operations.

As described before, our microprogram structure has **8-bit address width** and **48-bit content width**. However, Logisim has problems with bit-widths that are greater than 32-bit. It is impossible to use 48-bit width memory units using Logisim Evolution v3.3.4 (https://github.com/reds-heig/logisim-evolution/pull/394) because of integer overflow. The problem is fixed in the development branch but the version containing this fix is not released yet.

In order to overcome this issue, we used a **ROM** with 8-bit address width and 24-bit content width instead of 48-bit content width. We split the content width in half and used **dual line size** to overcome this issue. Split output of the ROM is merged with a splitter before processing begins.

The microcode we built and used for our microprogram structure is given below:

```
v3.0 hex words addressed
00: 00a500 000000 000000 000001 000000 000000 000000 000000
08: 000060 104084 000000 000001 000000 000000 000000 000000
10: cde9a3 0f8280 000000 000001 000000 000000 000000 000000
18: 00000c 20008c 870000 000000 000000 000001 000000 000000
```

```
20: 860000 000000 00a500 240000 000000 000001 000000 000000
28: cde9bf 0f80a0 000000 000001 000000 000000 000000 000000
30: cde9bf 0f80b0 000000 000001 000000 000000 000000 000000
38: 5b7f00 000000 cde9a3 0f8280 cde9a3 0f8282 000000 000001
40: 5a7e00 000000 cde9a3 0f8280 cde9a3 0f8282 000000 000001
48: cde9bf 0f80b8 000000 000001 000000 000000 000000 000000
50: cde9bf 0f80c0 000000 000001 000000 000000 000000 000000
58: cde9bf 0f8090 000000 000001 000000 000000 000000 000000
60: cde9bf 0f80d0 000000 000001 000000 000000 000000 000000
68: cde9bf 0f80d8 000000 000001 000000 000000 000000 000000
70: 250000 010000 000000 000001 000000 000000 000000 000000
78: 310000 010000 000000 000001 000000 000000 000000 000000
80: 310000 010000 000000 000001 000000 000000 000000 000000
88: 870000 200084 250000 010000 000000 000001 000000 000000
90: 860000 000000 250000 220000 000000 000001 000000 000000
98: 000000 000000 000000 000000 000000 000000 000000 000000
a0: 000000 000000 000000 000000 000000 000000 000000 000000
a8: 000000 000000 000000 000000 000000 000000 000000 000000
b0: 000000 000000 000000 000000 000000 000000 000000 000000
b8: 000000 000000 000000 000000 000000 000000 000000 000000
c0: 000000 000000 000000 000000 000000 000000 000000 000000
c8: 000000 000000 000000 000000 000000 000000 000000 000000
d0: 000000 000000 000000 000000 000000 000000 000000 000000
d8: 000000 000000 000000 000000 000000 000000 000000 000000
e0: 000000 000000 000000 000000 000000 000000 000000 000000
e8: 000000 000000 000000 000000 000000 000000 000000 000000
f0: 000000 000000 000000 000000 000000 000000 000000 000000
f8: 00a500 140000 000000 000001 000000 000000 000000 000000
```

Listing 1: The Microcode

This microcode provides all necessary information to carry out the instructions listed in the introduction part.

2.2.1 MOV Instruction

MOV instruction does MOVE operation. MOVE operation is an operation where the destination register's content gets replaced by source register's content. MOVE is a destructive operation because after MOVE operation, destination register's content becomes permanantly lost.

MOV operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - If SRCREG1 is in AR file and DSTREG is a general purpose register:

        * Set RegSel according to DstReg.
        * Set FunSel to load.
        * Set OutCSel to forward SRCREG1.
        * Set MuxASel to forward incoming SRCREG1 to general purpose register file input.
    - If SRCREG1 is a general purpose register and DSTREG is in AR File:
        * Set ARegSel according to DstReg.
        * Set AFunSel to load.
        * Set OutBSel to forward SRCREG1.
        * Set MuxBSel to forward ALU output to address register file input.
        * Set FunSelAlu to 1 to make ALU act as a buffer for input B.
    - If both SRCREG1 and DSTREG are general purpose registers:
        * Set RegSel according to DstReg.
        * Set FunSel to load.
        * Set MuxASel forward ALU output to general purpose register file input.
        * Set FunSelAlu to 1 to make ALU act as a buffer for input B.
        * Set OutBSel to forward SRCREG1.
    - If both SRCREG1 and DSTREG are in AR File:
        * Set ARegSel according to DstReg.
        * Set AFunSel to load.
        * Set OutCSel to forward SRCREG1.
        * Set MuxCSel to forward incoming SRCREG1 into ALU input A.
        * Set FunSelAlu to 0 to make ALU act as a buffer for input A.
        * Set MuxBSel to forward ALU output into AR file input.
- Second Cycle
    - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
=========================================================================
Address: 00010 000 Content: 110 01101 111 01001 101 000 110 000 11 11 10
    0 00 0 1010000 0 0 0
```

```
Address: 00010 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1
```

Listing 2: MOV Operation Microcode

2.2.2 ALU Operation Instructions

ALU Operations section denotes operations or instructions that requires ALU to be used.

ALU Operations are done as given below (cycles are relative to fetch):

- First Cycle
  - Set OutCSel according to SRCREG1.
  - Set OutASel according to SRCREG1.
  - Set OutBSel according to SRCREG2.
  - Set MuxCSel to low if SRCREG1 is in AR File, otherwise set it high.
  - Set FunSelAlu according to mode.
  - If DSTREG is in AR File, set MuxBSel to 3 to forward ALU output to AR File, set AR RegFile's RegSel and FunSel to load the input signal into the respective register.
  - If DSTREG is not in AR File, set MuxASel to 3 to forward ALU output to general purpose register file, set register file RegSel and FunSel to load the input signal into the respective register.
- Second Cycle
  - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
=========================================================================
ADD:
Address: 00101 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0010100 0 0 0
Address: 00101 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

```

```
SUB:
Address: 00110 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0010110 0 0 0
Address: 00110 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

AND:
Address: 01001 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0010111 0 0 0
Address: 01001 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

OR:
Address: 01010 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0011000 0 0 0
Address: 01010 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

NOT:
Address: 01011 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0010010 0 0 0
Address: 01011 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

LSL:
Address: 01100 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0011010 0 0 0
Address: 01100 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1

LSR:
Address: 01101 000 Content: 110 01101 111 01001 101 111 110 000 11 11 10
    0 00 0 0011011 0 0 0
Address: 01101 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1
```

Listing 3: ALU Operation Instructions Microcode

2.2.3 INC/DEC Instructions

INC/DEC instructions do increment or decrement operation. Increment and decrement operations are operations that increments or decrements SRCREG1 by 1, then moves the content to DESTREG.

This implementation of increment and decrement operation does the moving by simu-

lating MOV operation. And it does MOV two times (first, from SRCREG1 to DSTREG; then, from DSTREG to DSTREG) to ensure that the FLAGs are updated. By moving two times, chances of flags being not updated are eliminated completely. Consider the case where DSTREG is R0 and SRCREG is AR. If MOV was being done only one time, flags would not update because SRCREG would not pass through the ALU. When MOV is done two times, there is no way SRCREG not passed through the ALU.

INC/DEC module does the operations listed below (cycles are relative to fetch):

- First Cycle (increment or decrement)
  - If SRCREG1 is in address register file:
    - Set ARegSel according to SRCREG1.
    - Set AFunSel increment or decrement depending on the mode.
  - If SRCREG1 is a general purpose register:
    - Set RegSel according to SRCREG1.
    - Set FunSel increment or decrement depending on the mode.
- Second Cycle (move SRC to DST)
  - Do MOV (MOV microcode copy-paste)
- Third Cycle (move DST to DST to ensure FLAGS are up-to-date)
  - Set FakeSrc to high
  - Do MOV (MOV microcode copy-paste)
- Fourth Cycle
  - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
    OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
    (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
    Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
=========================================================================
INC:
Address: 01000 000 & Content: 010 11010 011 11110 000 000 000 000 00 00
    00 0 00 0 0000000 0 0 0
Address: 01000 001 & Content: 110 01101 111 01001 101 000 110 000 11 11
    10 0 00 0 1010000 0 0 0
Address: 01000 010 & Content: 110 01101 111 01001 101 000 110 000 11 11
    10 0 00 0 1010000 0 1 0
```

```
Address: 01000 011 & Content: 000 00000 000 00000 000 000 000 000 00 00
    00 0 00 0 0000000 0 0 1

DEC:
Address: 01000 000 & Content: 010 11011 011 11111 000 000 000 000 00 00
    00 0 00 0 0000000 0 0 0
Address: 01000 001 & Content: 110 01101 111 01001 101 000 110 000 11 11
    10 0 00 0 1010000 0 0 0
Address: 01000 010 & Content: 110 01101 111 01001 101 000 110 000 11 11
    10 0 00 0 1010000 0 1 0
Address: 01000 011 & Content: 000 00000 000 00000 000 000 000 000 00 00
    00 0 00 0 0000000 0 0 1
```

Listing 4: INC/DEC Instructions Microcode

2.2.4 PUL Instruction

PUL instruction does PULL operation.PULL operation is an operation where firstly the SP register's content increased. After, the content which stored in the address value held in the SP written on chosen register.

PULL operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set ARegSel in order to select SP.
  - Set AFunSel in order to increment SP.
- Second Cycle
  - Set OutDSel in order to send SP into memory address input.
  - Set RegSel according to REGSEL.
  - Set FunSel to load.
- Third Cycle
  - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
    OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
    (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
    Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
==========================================================================
```

```
Address: 00100 000 Content: 100 00110 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 0
Address: 00100 001 Content: 000 00000 101 00101 000 000 000 010 01 00 00
    0 00 0 0000000 0 0 0
Address: 00100 010 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1
```

Listing 5: PUL Operation Microcode

2.2.5 CALL Instruction

CALL instruction does PULL operation. CALL operation is an operation where the content in PC register is written to the address which stored in the SP. After, value in SP register decreased. Finally, the value written on ADDRESS field of the instruction written into PC register.

CALL operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set ARegSel in order to select SP.
  - Set AFunSel in order to decrement SP.
  - Enable write for RAM.
  - Set FunSelAlu to 0 in order to buffer.
  - Set OutDSel in order to send SP into memory address input.
  - Set OutCSel in order to send PC as an input to MuxA.
  - Set MuxASel to forward PC into MuxC.
  - Set MuxCSel in oder to send PC as an ALU input.
- Second Cycle
  - Set IR_selH as 0
  - Set MuxBSel in order to forward IR out to address register file input.
  - Set ARegSel to select PC.
  - Set AFunSel in order to load.
- Third Cycle
  - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
=====================================================================
Address: 01011 000 Content: 100 00111 000 00000 000 000 000 010 10 00 00
    0 00 0 0010000 1 0 0
Address: 01011 001 Content: 001 00101 000 00000 000 000 000 000 00 01 00
    0 00 0 0000000 0 0 0
Address: 01011 010 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1
```

Listing 6: CALL Operation Microcode

2.2.6 RET Instruction

RET instruction does RETURN operation. RETURN operation is an operation where firstly the SP register's content increased. After, the content which stored in the address value held in the SP written on PC register.

RETURN operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set ARegSel in order to select SP.
  - Set AFunSel in order to increment SP.
- Second Cycle
  - Set MuxBSel in order to send memory output into address register file input.
  - Set ARegSel in order to select PC.
  - Set AFunSel load.
  - Set OutDSel to send SP into memory address input.
- Third Cycle
  - Reset sequence counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
```

```
3 ======================================================================
4 Address: 01100 000 Content: 100 00110 000 00000 000 000 000 000 00 00 00
      0 00 0 0000000 0 0 0
5 Address: 01100 001 Content: 001 00101 000 00000 000 000 000 010 00 10 00
      0 00 0 0000000 0 0 0
6 Address: 01100 010 Content: 000 00000 000 00000 000 000 000 000 00 00 00
      0 00 0 0000000 0 0 1
```

Listing 7: RETURN Operation Microcode

2.2.7 PSH Instruction

Push operation allows us to write the value in any Rx register where the value in the SP register shows in memory. After doing this operation, the value in the SP register decreases by 1.

PUSH operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set With the appropriate OutBsel, the value of the Rx register is output. The OutBsel selection here is given as 011 as microcode. Because in the system we designed, assignment of REGSEL data in the case of 011 can be made to OutBSel.
  - For ALU to load B input, FunSel value must be 0001. In this case, the last 4 bits of the part allocated to ALU in microcode become 0001. Since the first 3 bits represent Enable, 001 is given to pass the value of 0001 directly.
  - Set RamWriteEnable to 1.
  - 10 must be selected to select the OutDSel value to give the value of the SP register. Therefore, we enter 010 in the value allocated for OutDsel in microcode. The leading 0 allows the last 2 bits to pass directly. The resulting SP value directly reaches the address of the Memory.
- Second Cycle
  - Secondly, we need to decrease the SP value by 1. For this, RegSel selection should show SP. In order for the ARegSel value to show SP, the value must be 100, so we give 100 in microcode.
  - For the selection of FunSel that affects SP, we need to consider the decrement process. So we give 11 to the last 2 bits of the 5-bit AFunSel selection in microcode. Since the first 3 bits represent Enable, we provide 001 to directly pass the value of 11, ie the count.

- Third Cycle
  - The part reserved for CounterReset in Microcode is given 1 to reset the counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
==========================================================================
Address: 00011 000 Content: 000 00000 000 00000 000 011 000 010 00  00
   00  0 00  0 0010001 1 0 0
\newline
Address: 00011 001  Content: 100  00111 000 00000 000 000 000 000 00  00
     00  0 00  0 0000000 0 0 0
\newline
Address: 00011 010 Content: 000 00000 000 00000 000 000 000 000 00  00
   00  0 00  0 0000000 0 0 1
```

Listing 8: PSH Operation Microcode

2.2.8 ST Instruction

The store operation allows the value of any Rx register to be written to the location where the value of AR (Address Register) represents in the memory.

STORE operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set With the appropriate OutAsel, the value of the Rx register is output. The OutAsel selection here is given as 011 as microcode. Because in the system we designed, assignment of REGSEL data in the case of 011 can be made to OutASel.
  - Because we will write the value in the Rx register to the location that the Address Register represents in memory, 001 is selected in OutDSel microcode. Since the first bit is 0, the value 01 goes directly to OutDSel.
  - By selecting MuxCSel value 1, the data from part A of the register file is transferred to ALU.
  - In order to load A value from ALU, 4 bit ALUFunSel 0000 must be selected. For this, 3 bits 001 is selected, which represents Enable in microcode.
- Seccond Cycle

– The part reserved for CounterReset in Microcode is given 1 to reset the counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
==========================================================================
Address: 00001 000 Content: 000 00000 000 00000 011 000 000 001 000 000
   01  0 00  0 0010000 1 0 0
Address: 00001 001 Content: 000 00000 000 00000 000 000 000 000 000 000
   00  0 00  0 0000000 0 0 1
```

Listing 9: ST Operation Microcode

2.2.9 LD Instruction [Direct]

Load operation is divided into 2 ways according to the value of instruction data in the IM part. No IM value was sent to the Address of the ROM while designing the microcode structure. The reason it was designed like this was to make it take up less space. Therefore, we divided the Load operation into 2 and 2 different instructions were performed. If the IM value is 0, OPCODE value for the Load operation was taken 00000. When the IM value was 1, it was converted to OPCODE 11111. Basically, the Load operation writes the value in memory directly to any Rx register, depending on the IM value, or the value in the location where the address register displays in memory.

LOAD operation for IM = 0 cycles are listed below (cycles are relative to fetch):

- First Cycle
  - The selection of the RegSel value is selected based on the REGSEL data in the instruction. Accordingly, 101 was selected in microcode. This selection means that RegSel data is selected according to the REGSEL data sent in the instruction.
  - To load, FunSel value 01 must be selected because registers are designed in this way. We have 5 bits for FunSel in Microcode, the first 3 bits represent Enable, while the last 2 bits represent content. By selecting Enable 001, direct content is selected as FunSel data.
  - By selecting MuxASel value 00, we can direct the value written to IR-L directly to the input of G.P.R. For this reason, MuxASel value 00 is selected in microcode..

- Second Cycle
  - The part reserved for CounterReset in Microcode is given 1 to reset the counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
==========================================================================
Address: 00000 000 Content: 000 00000 101 00101 000 000 000 000 00  00
   00  0 00  0 0000000 0 0 0
Address: 00000 001 Content: 000 00000 000 00000 000 000 000 000 00  00
   00  0 00  0 0000000 0 0 1
```

Listing 10: LD (Direct) Operation Microcode

2.2.10 LD Instruction [Immediate]

LOAD operation for IM = 1 cycles are listed below (cycles are relative to fetch):

- First Cycle
  - The selection of the RegSel value is selected based on the REGSEL data in the instruction. Accordingly, 101 was selected in microcode. This selection means that RegSel data is selected according to the REGSEL data sent in the instruction.
  - To load, FunSel value 01 must be selected because registers are designed in this way. We have 5 bits for FunSel in Microcode, the first 3 bits represent Enable, while the last 2 bits represent content. By selecting Enable 001, direct content is selected as FunSel data.
  - The OutDSel value allows 001 to be selected for OutDSel data by selecting 001 in microcode. In this way, we reach the location where the value in the Address Register points in memory.
  - By selecting the MuxASel value 01 in microcode, we can move the value from the address register in memory to the input of G.P.R.
- Seccond Cycle
  - The part reserved for CounterReset in Microcode is given 1 to reset the counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
========================================================================
Address: 11111 000 Content: 000 00000 101 00101 000 000 000 001 01  00
   00  0 00  0 0000000 0 0 0
Address: 11111 001 Content: 000 00000 000 00000 000 000 000 000 000 000
   00  0 00  0 0000000 0 0 1
```

Listing 11: LD (Immediate) Operation Microcode

2.2.11 BRA Instruction

BRA instruction handles BRA operation. BRA writes address field to PC. It is used for branching. Along with BNE and BEQ, basic if statements can be implemented.

BRA operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set IRSelH to 0.
  - Set MuxBSel in order to reach Addres Register.
  - Set ARegSel in order to set PC.
  - Set AFunSel in order to load value to PC.
- Second Cycle
  - Reset CounterReset to 1 in order to reset counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
   OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
   (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
   Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
========================================================================
Address: 01110 000 Content: 001 00101 000 00000 000 000 000 000 00 01 00
    0 00 0 0000000 0 0 0
Address: 01110 001 Content: 000 00000 000 00000 000 000 000 000 00 00 00
    0 00 0 0000000 0 0 1
```

Listing 12: BRA Operation Microcode

2.2.12 BEQ Instruction

BEQ instruction handles BEQ operation. BEQ writes address field to PC only if "Z" is "1".

BEQ operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set IRSelH to 0.
  - Set MuxBSel in order to reach Addres Register.
  - Set ARegSel in order to set PC.
  - Set AFunSel in order to load value to PC.
- Second Cycle
  - Reset CounterReset to 1 in order to reset counter.

```
1 Address: OpCode (5) Time (3)
2 Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
     OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
     (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
     Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
3 ========================================================================
4 Address: 01110 000 Content: 001 10001 000 00000 000 000 000 000 00 01 00
      0 00 0 0000000 0 0 0
5 Address: 01110 010 Content: 000 00000 000 00000 000 000 000 000 00 00 00
      0 00 0 0000000 0 0 1
```

Listing 13: BEQ Operation Microcode

2.2.13 BNE Instruction

BNE instruction handles BNE operation. BNE writes address field to PC only if "Z" is "0".

BNE operation cycles are listed below (cycles are relative to fetch):

- First Cycle
  - Set IRSelH to 0.
  - Set MuxBSel in order to reach Address Register.
  - Set ARegSel in order to set PC.
  - Set AFunSel in order to load value to PC.

- Second Cycle
  - Reset CounterReset in order to reset counter.

```
Address: OpCode (5) Time (3)
Content: ARegSel (3) AFunSel (E+2) RegSel (3) FunSel (E+2) OutASel (3)
    OutBSel (3) OutCSel (3) OutDSel (3) MuxASel (2) MuxBSel (2) MuxCSel
    (2) IR_selH (1) IR_FunSel (2) IR_Enable (1) ALUFunSel (E+4)
    Ram_WriteEnable(1) FakeSrc (1) CounterReset (1)
=========================================================================
Address: 01110 000 Content: 001 10001 000 00000 000 000 000 000 00 01 00
     0 00 0 0000000 0 0 0
Address: 01110 010 Content: 000 00000 000 00000 000 000 000 000 00 00 00
     0 00 0 0000000 0 0 1
```

Listing 14: BNE Operation Microcode

2.3 Sequence Counter

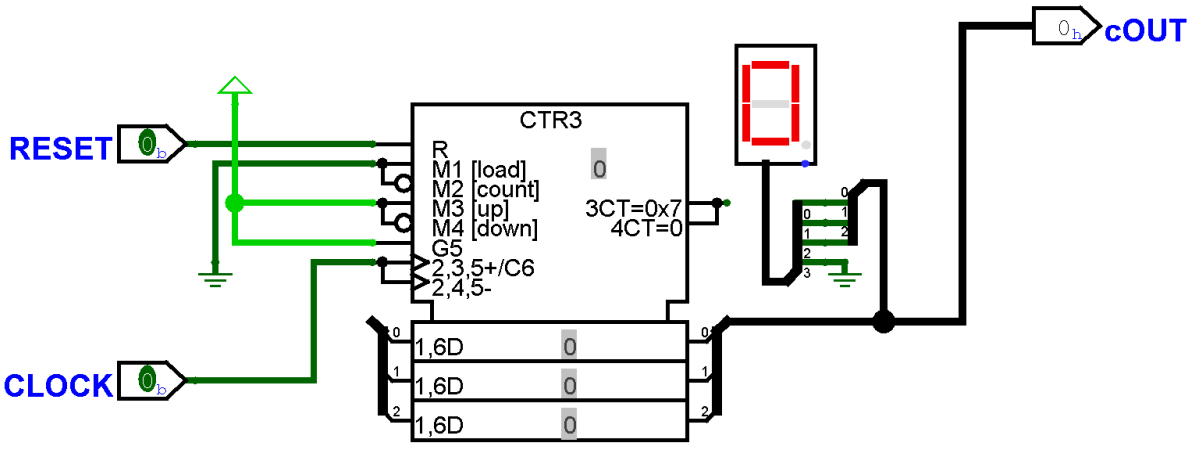

Figure 1: Sequence counter implementation.

Sequence counter is a 3-bit circular counter that counts from decimal 0 to decimal 7. It was initially designed as 4-bit but because no instruction needed more than a few clock cycles, it was reduced to be 3-bit counter.

Output of the sequence counter is decoded with a 3-to-8 bit decoder for instruction implementations to selectively enable a specific cycle.

Output of sequence counter is as such:

```
000
001
010
011
```

```
5 ...
```

Listing 15: Sequence Counter Output

2.4 Fetch Module

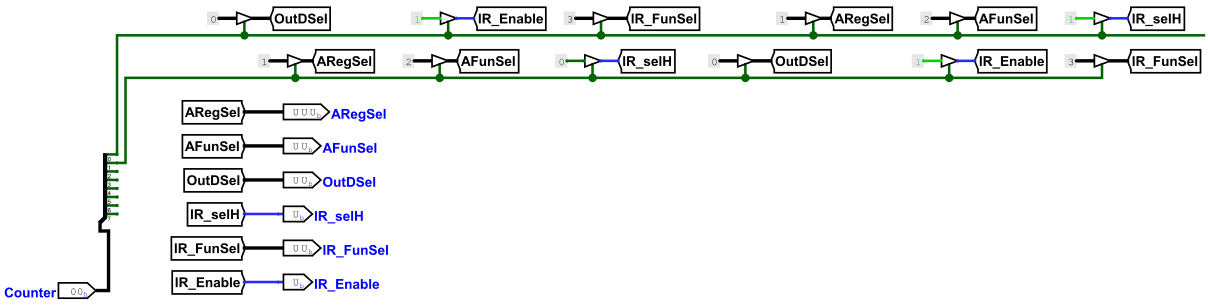

Figure 2: Fetch Module

Fetch module, or fetcher, does the job of retrieving 16-bit instruction to the IR register. Since the memory used in the organization is **asynchronous read**, fetching is done in only 2 cycles.

Fetching prepares a potential operation execution for the basic computer by founding an initial bridge between memory (RAM) and control unit. Without fetching, the computer would not know which instruction to execute. Since the computer is **big-endian**, fetcher first writes into IR_H then in the next cycle it writes into IR_L.

Fetcher increments PC by 2 to point to the next instruction in the memory for the next fetch procedure.

After fetching, OpCode part of the ROM address input is fed with OpCode part of IR_H which is the last 5-bits of data.

Fetch module does the operations listed below:

- First Cycle (Retrieve first 1-byte into IR_H)
  - Set memory address input to PC.
  - Enable IR.
  - Set IR_H to high.
  - Increment PC by 1.
- Second Cycle (Retrieve second 1-byte into IR_L)
  - Set memory address input to PC.
  - Enable IR.

– Set IR_L to high.

– Increment PC by 1.

2.5 The Control Unit

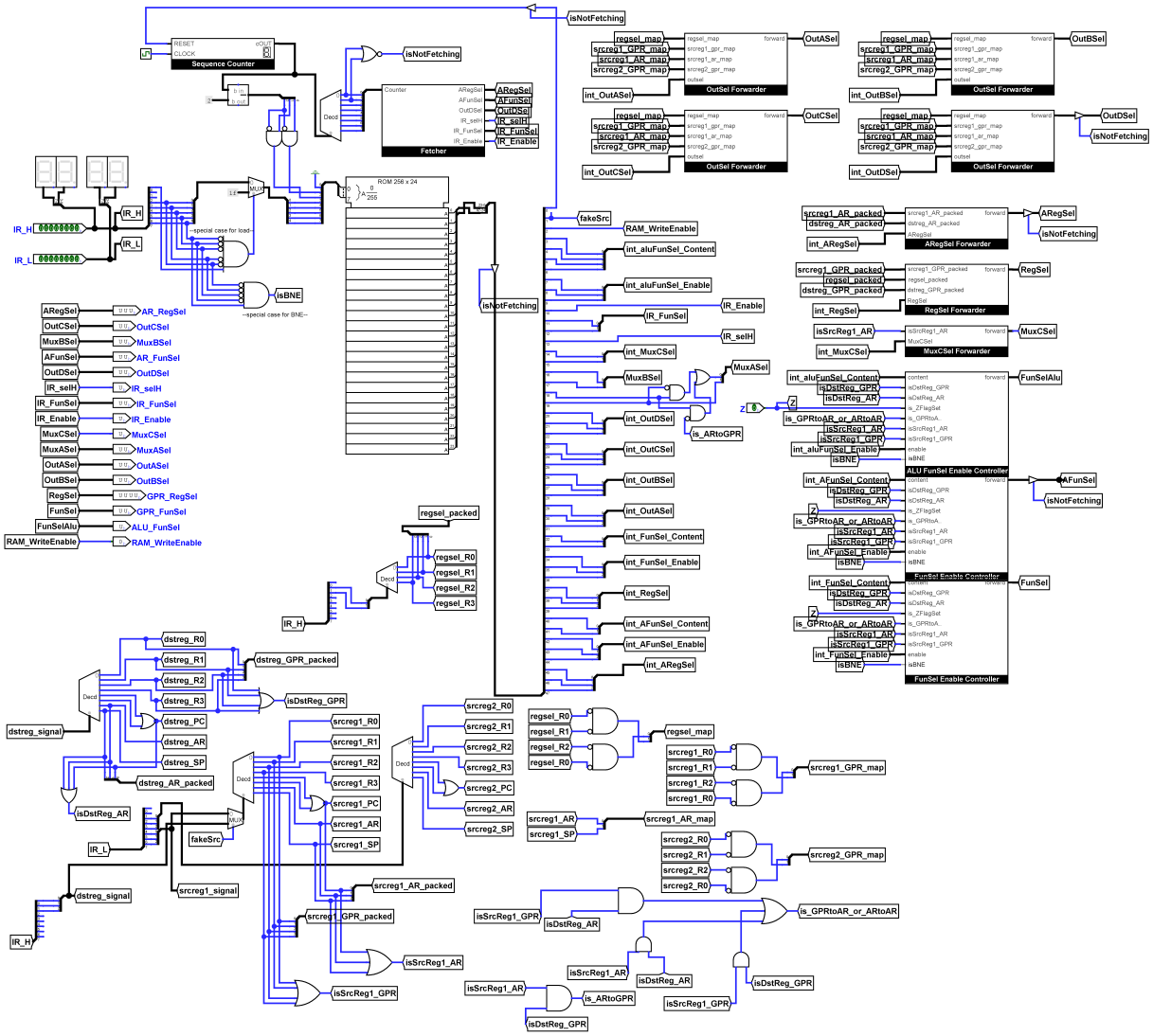

Figure 3: The Control Unit

The control unit consists of the following items:

- 3-bit Sequence Counter
- Fetch Module
- OutSel Forwarder for OutASel
- OutSel Forwarder for OutBSel

- OutSel Forwarder for OutCSel
- OutSel Forwarder for OutDSel
- ARegSel Forwarder
- RegSel Forwarder
- FunSel Forwarder
- AFunSel Forwarder
- MuxCSel Forwarder
- FunSel Enable Controller for GPR FunSel
- FunSel Enable Controller for AR FunSel
- ALUFunSel Enable Controller for ALU FunSel
- 8-Bit Address Width and 24-bit Dual Content Width ROM

At the top of it is a sequence counter and fetcher. First two cycles (0 and 1) are reserved for fetch, so when sequence counter becomes 2, it first gets subtracted by 2 before fed into ROM.

Since ROM has not got 48-bit content bit width but has 24-bit dual content bit width (reason is explained before), there is a trick used for setting time portion of the ROM address properly. The two and gates connected to subtractor output ensure that the time is increased two times at every clock tick. The reason for it is to read 48-bit content from 24-bit dual line at one step. Without doing this, ROM will be misaligned for our microprogram structure.

At the left side, there can be seen a sub-circuit for LOAD and BNE instructions. As Load is the only instruction that accepts Direct and Immediate mode, instead of modifying microprogram structure solely for it, we created an alias instruction with opcode of [0x1f] for Load. When instruction is Load (Immediate), the opcode gets forwarded to ROM normally; but when instruction is Load (Direct), that part converts opcode to 0x1f before feeding into ROM. isBNE tunnel is set when the instruction is BNE and this is required for Enable: 100 configuration which is needed by BNE and BEQ instructions.

At the bottom there are decodings and mappings done. Decoders carefully split the instruction into its building blocks and packs them if necessary. For example, is SRCREG1 is R2, this part makes the SRCREG1_R2 tunnel high. Mappings are generally done for OutSel inputs (For example if SRCREG1 is R1, it allows making OutASel R1). Packed tunnels are needed for RegSel inputs.

At the right side there are modules that are used for forwarding. They are created for this specific microprogram configuration and are used to do the mappings/forwardings described in 'Microprogram Structure' section.

In the middle ROM out signal is used to drive the computer (organization).

2.5.1 OutSel Forwarder

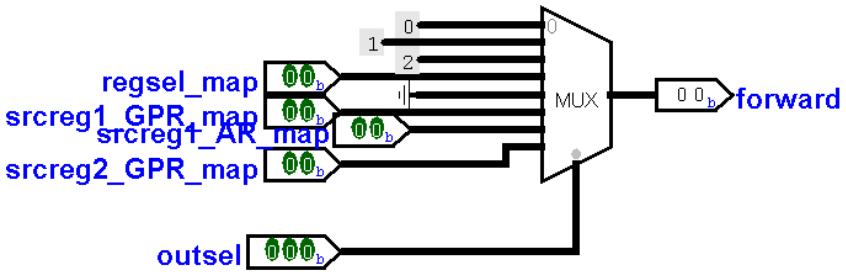

Figure 4: OutSel Forwarder

OutSel Forwarder module does OutSel forwarding according to the specifications laid out in 'OutSel Forwarding' section.

2.5.2 FunSel Enable Controller

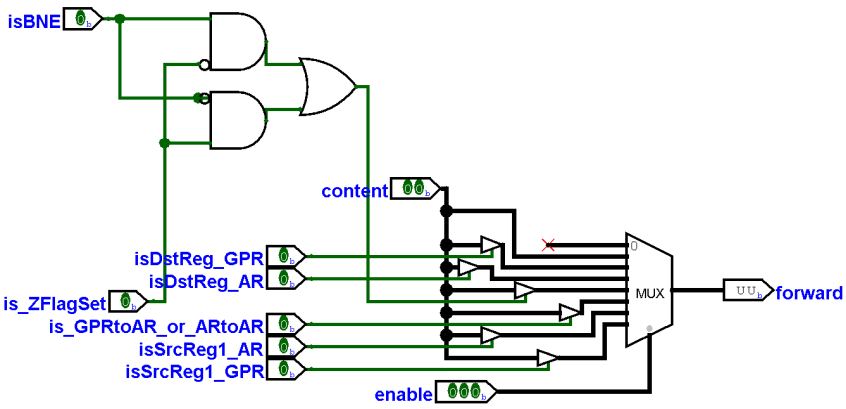

Figure 5: FunSel Enable Controller

FunSel Enable Controller module does enable controlling according to the specifications laid out in 'Enable Controlling' section.

2.5.3 ALU FunSel Enable Controller

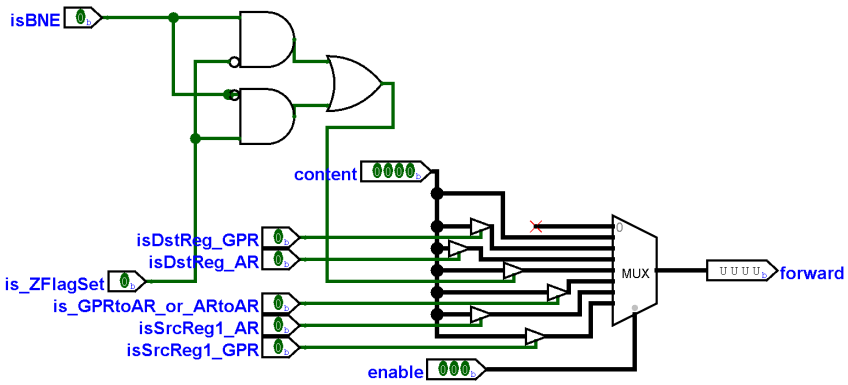

Figure 6: ALU FunSel Enable Controller

ALU FunSel Enable Controller module does enable controlling according to the specifications laid out in 'Enable Controlling' section.

2.5.4 MuxCSel Forwarder

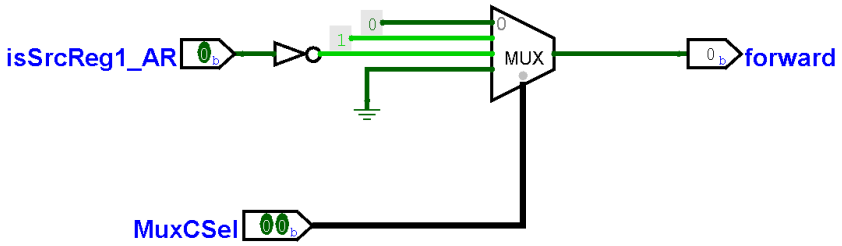

Figure 7: MuxCSel Forwarder

MuxCSel Forwarder module does MuxCSel forwarding according to the specifications laid out in 'MuxCSel Forwarding' section.

2.5.5 ARegSel Forwarder

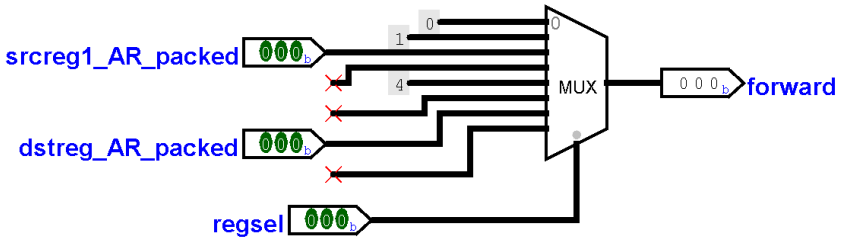

Figure 8: ARegSel Forwarder

ARegSel Forwarder module does ARegSel forwarding according to the specifications laid out in 'Regsel Forwarding' section.

2.5.6 RegSel Forwarder

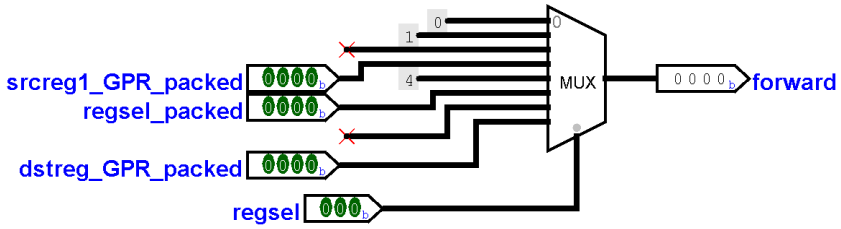

Figure 9: RegSel Forwarder

RegSel Forwarder module does RegSel forwarding according to the specifications laid out in 'Regsel Forwarding' section.

2.6 Computer Organization

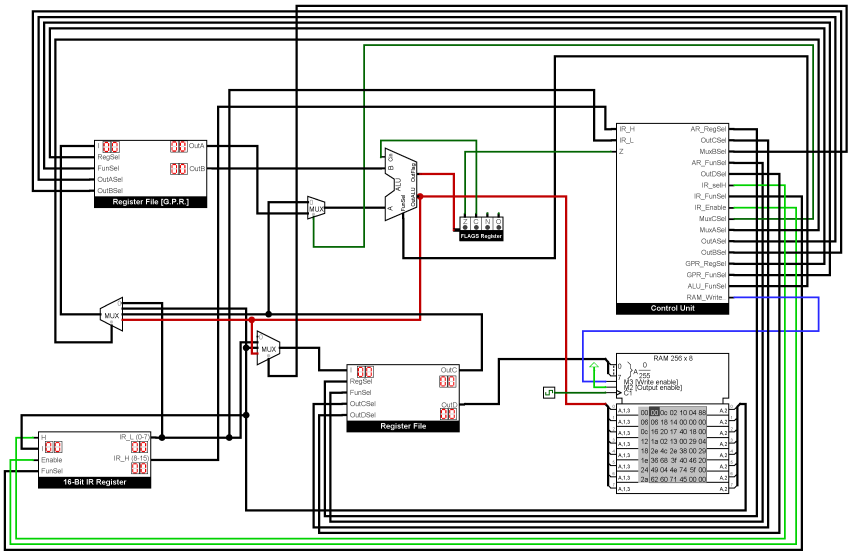

Figure 10: The Computer Organization

The provided Computer Organization schematic was designed and realized, then the control unit was put inside and then connected to the necessary parts.

3 RESULTS

All modules were tested case by case, and all of them worked properly.

3.1 Example Program

An example program which calculates M[A0]+M[A1]+M[A2]+M[A3]+M[A4] and stores the result at M[A6] was given in the project documentation file.

```
    ORG 0x20 # Write the program starting from the address 0x20
    LD R0 IM 0x05 # R0 is used for iteration number
    LD R1 IM 0x00 # R1 is used to store total
    LD R2 IM 0xA0
    MOV AR R2 # AR is used to track data adrress: starts from 0xA0
LABEL:  LD R2 D # R2 <- M[AR] (AR = 0xA0 to 0xA4)
    INC AR AR # AR <- AR + 1 (Next Data)
    ADD R1 R1 R2 # R1 <- R1 + R2 (Total = Total + M[AR])
    DEC R0 R0 # R0 <- R0 - 1 (Decrement Iteration Counter)
    BNE IM LABEL # Go back to LABEL if Z=0 (Itertaion Counter > 0)
    INC AR AR # AR <- AR + 1 (Total will be written to 0xA5)
    ST R1 D # M[AR] <- R1 (Store Total at 0xA5)
```

Listing 16: Example Program Source Code

Example program was assembled manually and the machine code was put into the computer's memory. Note that an extra BRA 0x20 was added into the beginning to jump to the program's first command. We may think of this additional command as the **bootloader**. When a computer starts it starts executing the program in the memory starting by 0 (because PC is initially 0). ORG directive dictates where the program must reside in the memory. Since in this case it was 0x20, BRA 0x20 was put in M[0] to set PC (program counter) to 0x20 at the beginning.

```
v3.0 hex words addressed
00: 71 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 05 02 00 04 a0 16 40 05 00 46 c0 29 28 38 00
30: 80 28 46 c0 0b 00 00 00 00 00 00 00 00 00 00 00
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Listing 17: Assembled Example Program

Then the program was executed with the following input values on the computer:

```
M[A0] = 01
M[A1] = 01
M[A2] = 01
M[A3] = 01
M[A4] = 01
```

Listing 18: Example Program Input

And as expected, the computer successfully computed the sum of these inputs as 5 (1+1+1+1+1=5). Which can be seen here:

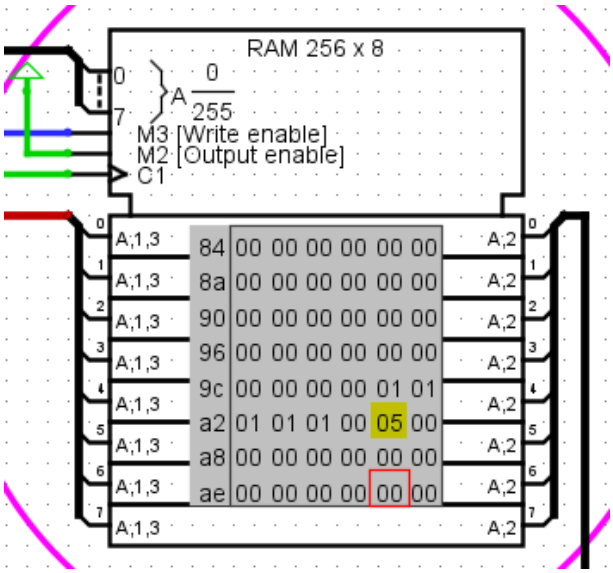

Figure 11: Result of the example program execution (highlighted part is where the result stored).

4 DISCUSSION

Control Unit is the core part of the CPU that founds a bridge between programs and the computer. Control Unit is a sequential circuitry that makes the computer do certain operations on a sequential order. A control unit may have a microprogram in a separate EEPROM to drive output signals to perform operations or it may be hardwired.

In this project, a microprogrammed control unit was designed. It was done so by carefully designing a microprogram structure and then implementing necessity circuit that is capable of operating on the structure designed.

5 CONCLUSION

We learned how to design a microprogrammed Control Unit that is capable of executing 19 different instructions. We learned how to carry out basic operations such as MOV, CALL and BRA as well as addressing modes and how addressing works. We did not encounter with any abnormal result. All of the modules that were designed worked properly. We learned about endianness (big/little endian), microcodes, and CISC / RISC CPUs.

REFERENCES

[1] Computer Organization. Computer Organization Guidelines. *Design of basic computer*.