ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E COMPUTER ORGANIZATION

PROJECT 4 REPORT

GROUP NO : 1

GROUP MEMBERS:

150170095	:	Mehmet ALTUNER
150170026	:	Berkay OLGUN
150170103	:	Atahan ÖZER
150170076	:	Ekrem UĞUR

SPRING 2020

Contents

1 Introduction

For the final take home exam of the lecture, we designed a working CPU using microprogrammable control unit and microoperations. We implemented mapping, branching and conditioning logics for control unit and designed the format of the microinstructions. We then implemented circuits for each microoperations and wrote the codes of operations using microinstructions which includes microoperations, brancinh and conditioning values and address to use.

2 Control Unit

The control unit is a microprogrammable control unit. It has a ROM as the memory in which codes for operations reside, two registers named as CAR and SBR. CAR is responsible for controlling and addressing ROM and SBR is used for sub-routines. Also branching and mapping logics are implemented in this unit. Control unit simply decides which microoperations to run for the given OPCODE and branches accordingly to the next microinstruction.

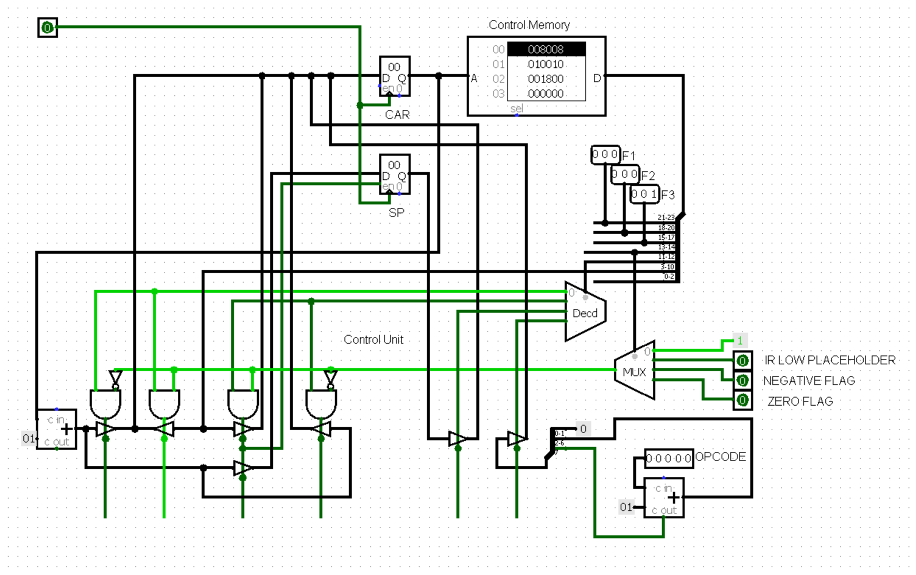

Figure 1: Microprogrammable Control Unit

2.1 Format

We used the format below to achieve a microinstruction structure that is useful for us. Reserving 9 bits to control microoperations, 2 bits each for con-

3-bits	3-bits	3-bits	2-bits	2-bits	8-bits
F1	F2	F3	CD	BR	ADDRESS

ditioning and branching and finally 8 bits for the address value.

2.2 Branching

Conditioning table for branching is as follows:

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional Branch
01	Addressing Mode	I	Addressing mode is one or not
10	Negative Flag	S	It is negative or not
11	Zero Flag	Z	It is zero or not

Note that CD value comes from ROM.

According to the value of CD (whether 1 or 0), following branching happens:

BR	Symbol	Function
00	JMP	CAR <- AD if condition value is 1 CAR <- CAR + 1 if condition value is 0
01	CALL	CAR <- AD, SBR <- CAR + 1 if condition value is 1 CAR <- CAR + 1 if condition value is 0
10	RET	CAR <- SBR
11	MAP	Mapping logic is implemented on given OPCODE

2.3 Mapping

Mapping logic is as follows: Add one to the OPCODE value and shift it to left by two.

Thanks to mapping logic, we have 4 consecutive lines to implement each operation using microinstructions.

3 Micro Operations

Micro Operations are served as building blocks of microinstrutions. We designed 21 different Micro Operations to implement given operations.

Micro operations under the F1 is as follows:

000	Nope	
001	DST <- SRC	SRCTDST
010	DST <- M[AR]	MTDST
011	DST <- IR[L]	IRTODST
100	DST <- DST + 1	INC
101	RF1 <- RF1 - RF2	SUB
110	DST <- DST -1	DEC
111	M[AR] <- DST	SRCTOM

Micro operations under the F2 is as follows:

000	Nope	
001	RF1 <- RF1 + RF2	ADD
010	RF1 <- RF1 & RF2	AND
011	RF1 <- RF1 — RF2	OR
100	DST <- !SRC	NOT
101	DST <- LS(SRC)	LSSRC
110	DSR <- RS(SRC)	RSSRC
111	ALU <- DST	DSTTALU

Micro operations under the F3 is as follows:

000	Nope	
001	IR[L] <- M[AR]	MARTIRL
010	IR[H] <- M[AR]	MARTIRH
011	M[SP] <- SRC	SRCTSP
100	SRC <- M[SP]	SPTSRC
101	PC <- IR[L]	IRLTPC
110	SP <- SP + 1	INCSP
111	SP <- SP - 1	DECSP

Note that RF refers to ”Register File” and it may be address register file or register file.

3.1 SRCTDST

This micro operation writes the value of SRC register into DST register. Its value is 001 under F1. Both DST register and SRC register may take values pointing to address register file or register file.

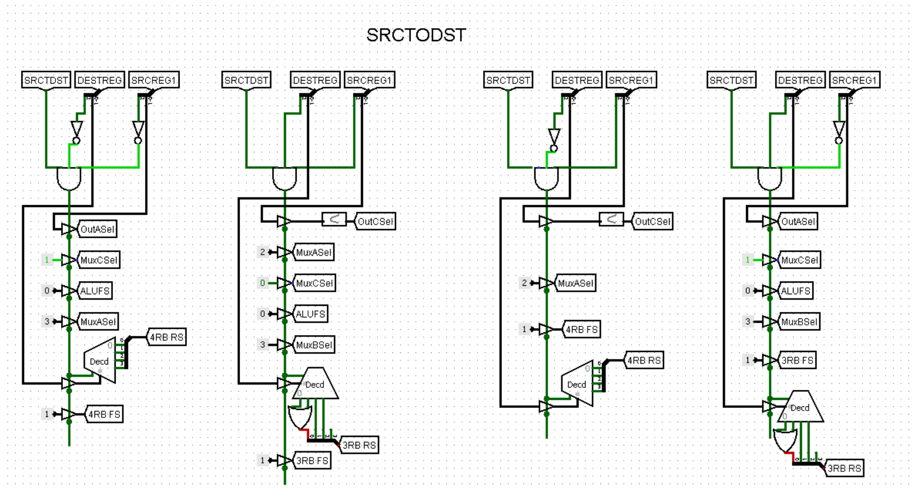

Figure 2: The circuit implementation of micro operation SRCTDST

3.2 MTDST

This micro operation writes the value in the memory cell pointed by the value of AR register into the DST register. Its value is 111 under F1. DST register may only take values pointing to register file.

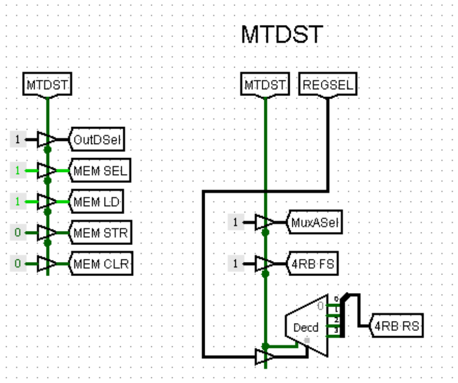

Figure 3: The circuit implementation of micro operation MTDST

3.3 MARTIRL & MARTIRH

These micro operations is used to load the value in the memory cell pointed by AR register to IR register. MARTIRL writes to low part of IR while MARTIRH writes to high part of IR. These micro operations are mainly used in FETCH operation. Their values are 001, 010 respectively under F3.

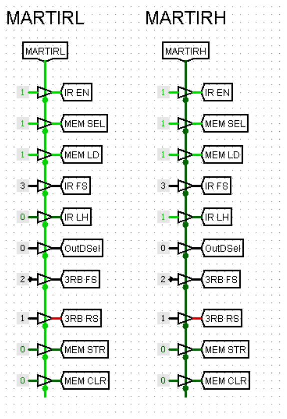

Figure 4: The circuit implementation of micro operations MARTIRL & MARTIRH

3.4 SRCTOM

This micro operation writes the value of SRC register into the memory cell pointed by the AR. SRC may only point to the registers of the register file. Its value is 111 under F1.

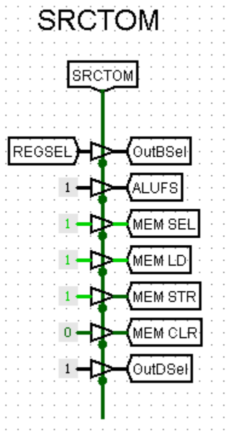

Figure 5: The circuit implementation of micro operation SRCTOM

3.5 Binary ALU Micro Operations

Binary ALU micro operations consist of ADD, SUB, AND , OR. Each of those micro operations have the same circuit except the ALU funsel; therefore, only ADD micro operation is explained in the report. Output of the binary operations depend on the destination register therefore our binary micro operation consist of two part such as destination 3 register bank and destination 4 register bank.

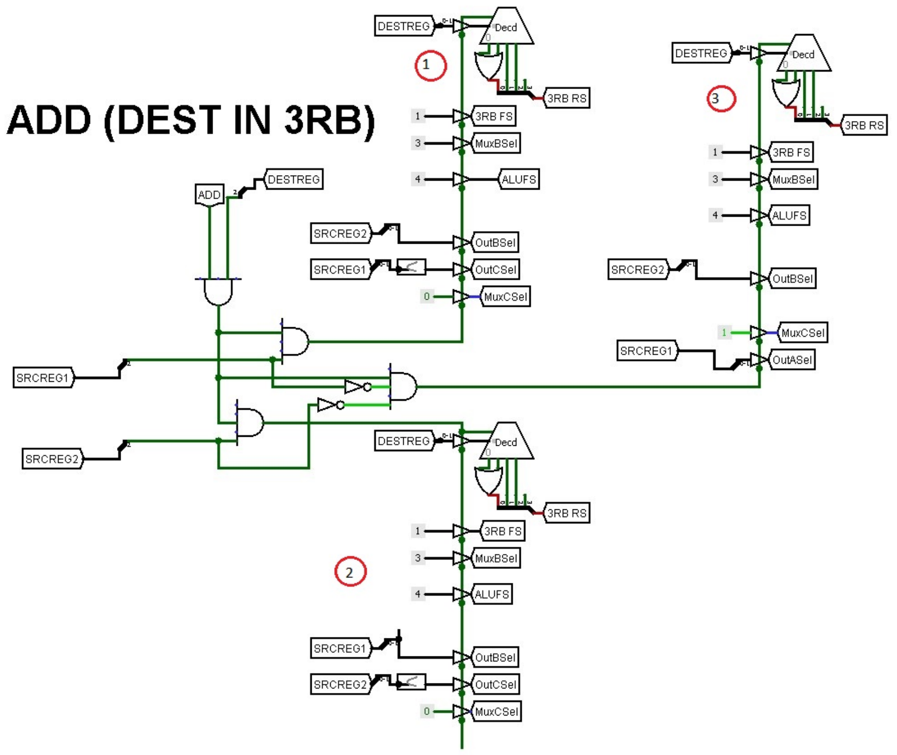

Figure 6: Destination 3 Register Bank

Destination 3RB circuit consist of 3 branches. First branch, which is shown with red numbered circle in the figure, belongs to micro operation 3RB = 3RB + 4RB. Second branch belongs to 3RB = 4RB + 3RB. Final branch belongs to 3RB = 4RB + 4RB.

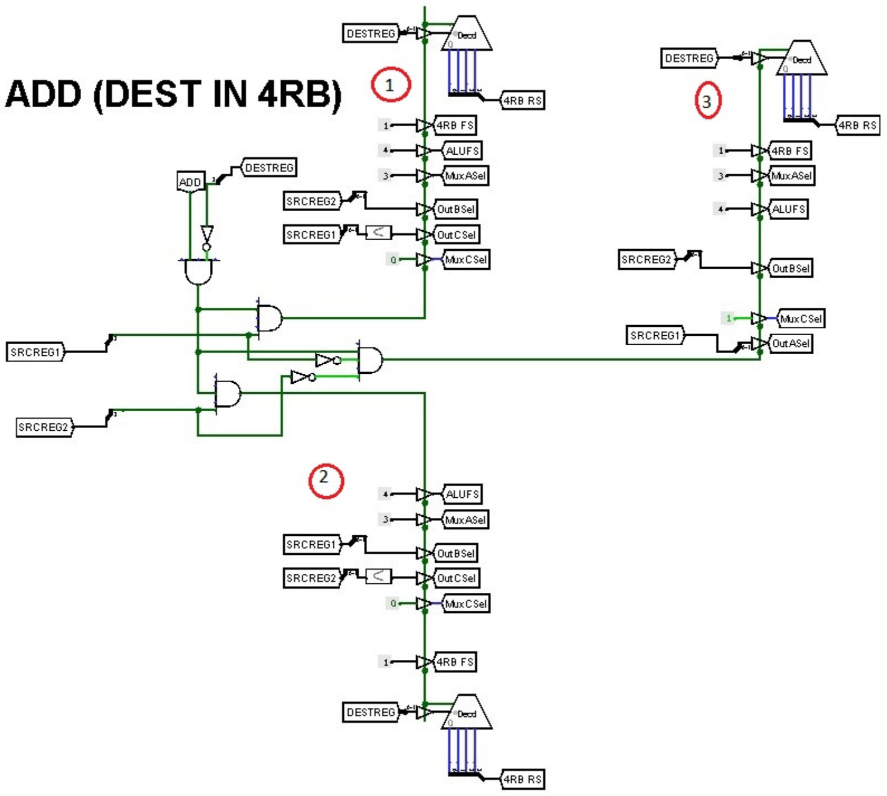

Figure 7: Destination 4 Register Bank

Destination 4RB circuit designed with the same manner as destination 3 RB, only difference is the usage of the multiplexers. First branch belongs to micro operation 4RB = 3RB + 4RB. Second branch belongs to 4RB = 4RB + 3RB. Final branch belongs to 4RB = 4RB + 4RB.

3.6 Unary ALU Micro Operations

Unary ALU micro operations consist of INC, DEC, NOT, LSSR, RSSR. The pair INC-DEC has the same structure, the remaining operations have also the same structure. Between those groups only the function selectors of register banks or Arithmetic Logic Unit changes.

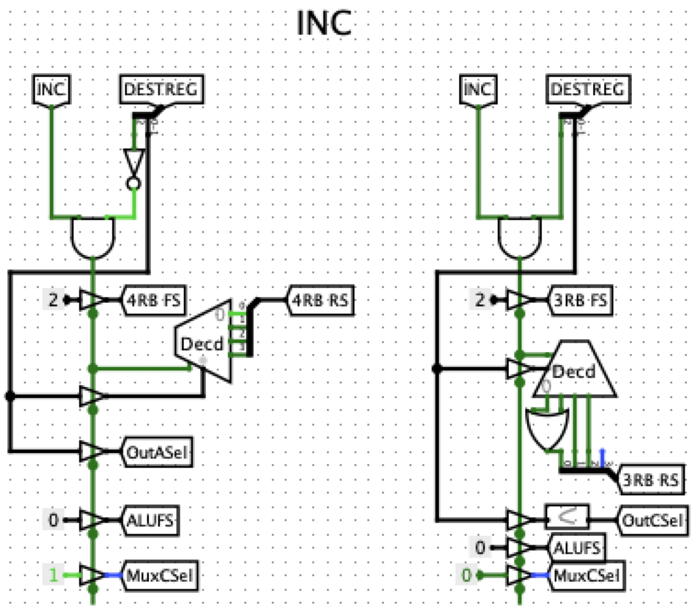

Figure 8: The circuit implementation of micro operation INC

3.6.1 INC

This micro operation increments the selected registers value by 1 and has the same structure with micro operation DEC, except the register function selectors.

3.6.2 DEC

This micro operation decrements the selected registers value by 1 and has the same structure with micro operation INC, except the register function selectors.

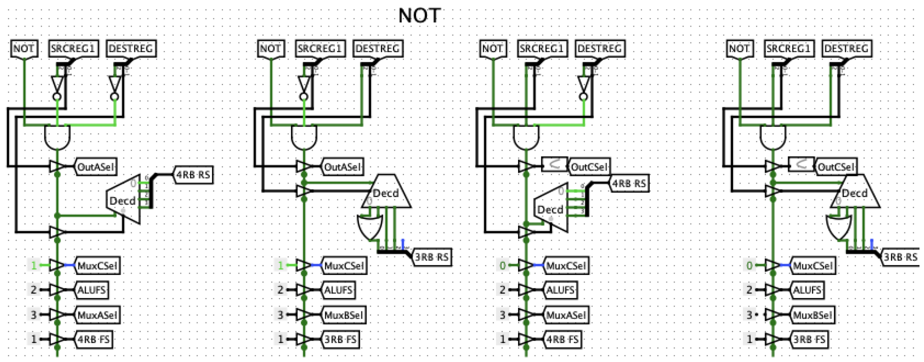

Figure 9: The circuit implementation of micro operation NOT

3.6.3 NOT

This micro operation takes the given register's value, negates and loads it to the destination register. There are 4 different circuits for 4 different cases which are, RF to RF, RF to ARF, ARF to RF, ARF to ARF respectively. According to the input and output register

3.6.4 LSSRC

This micro operation takes the given register's value shifts it logical left and loads it to the destination register. LSSRC has the same structure with NOT micro operation except LSSRC has ALU Function Selector value a, which is logical left shift operation.

3.6.5 RSSRC

This micro operation takes the given register's value shifts it logical right and loads it to the destination register. RSSRC has the same structure with the micro operations above except RSSRC has ALU Function Selector value b, which is logical right shift operation.

3.7 IRLTPC

This micro operation moves low part of the IR to PC.

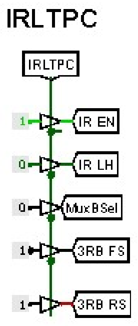

Figure 10: IR low to PC

3.8 INCSP

This micro operation increments SP by 1.

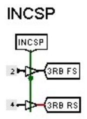

Figure 11: SP Increment

3.9 INCSP

This micro operation decrements SP by 1.

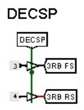

Figure 12: SP Decrement

4 Operation Routines

4.1 FETCH

We used two microoperations to read the instruction value from the memory.

1. MARTIRL
2. MARTIRH

These two microoperations allowed us to fill the instruction register. Then by setting the branching bits to 11 we continue with MAP.

4.2 LD

Using two separate microoperations and considering the indirect address bit of the instruction, we either moved the value in memory address or in instruction register to the register.

1. MTDST
2. IRTODST

4.3 ST

This operation uses only one microoperation. It always executes one microoperation and then fetches the next instruction.

1. SRCTOM

4.4 MOV

Just like store operation, move operation also executes one single microoperation and then branches to the fetch microinstruction routine.

1. SRCTDST

4.5 PSH & PUL

Pushing and pulling from the stack is basically combination of two different operations. We either first store the value and then decrement the stack register's value or we increment the value in stack register and load the direct value to the destination register. Corresponding microoperations are shown below in order.

1. SRCTSP
2. DECSP
3. INCSP
4. SPTSRC

4.6 INC

This operation consists of 3 cycles:

1. SRCTDST
2. INC
3. DSTTALU

First we carry the source register to the destination, then increment it there using micro operation INC. Then we take the output of our operation to the ALU in order to observe the zero flag.

4.7 DEC

This operation consists of 3 cycles:

1. SRCTDST
2. DEC
3. DSTTALU

First we carry the source register to the destination, then decrement it there using micro operation DEC. Then we take the output of our operation to the ALU in order to observe the zero flag.

4.8 NOT

This operation takes only one cycle in our implementation, which is:

1. NOT

In this operation we take the source register's value give it to ALU and take negated output of the ALU to the destination register.

4.9 LSL

This operation takes only one cycle in our implementation, which is:

1. LSL

In this operation we take the source register's value give it to ALU and take shifted output of the ALU to the destination register.

4.10 RSL

This operation takes only one cycle in our implementation, which is:

1. RSL

In this operation we take the source register's value give it to ALU and take shifted output of the ALU to the destination register.

4.11 ADD

This operation takes only one cycle in our implementation, which is:

1. ADD

This operation can perform 6 diffrent addition on register banks,which are :

- ARF = ARF + RF
- ARF = RF + ARF
- ARF = RF + RF
- RF = ARF + RF
- RF = RF + ARF
- RF = RF + RF

4.12 AND

This operation takes only one cycle in our implementation, which is:

1. AND

This operation can perform 6 different ”and” operation on register banks,which are :

- ARF = ARF RF
- ARF = RF ARF
- ARF = RF RF
- RF = ARF RF
- RF = RF ARF
- RF = RF RF

4.13 OR

This operation takes only one cycle in our implementation, which is:

1. OR

This operation can perform 6 different ”or” operation on register banks,which are :

- ARF = ARF — RF
- ARF = RF — ARF
- ARF = RF — RF
- RF = ARF — RF
- RF = RF — ARF
- RF = RF — RF

4.14 SUB

This operation takes only one cycle in our implementation, which is:

1. SUB

This operation can perform 4 different subtraction on register banks,which are :

- ARF = ARF - RF
- ARF = RF - RF
- RF = ARF - RF
- RF = RF - RF

4.15 BRA

Using one microoperation specifically designed for this operation, we load a value to the program counter register.

- IRLTPC

4.16 BEQ & BNE

Depending on the zero flag output coming from the ALU, value of PC is changed with a very straight forward logic. We used the same microoperation as we used in BRA operation. The difference is the conditioning bits.

- IRLTPC

4.17 CALL

This operation's routine consists of three consequent execution. First we basically push the value in PC to stack, using the routine above, then we load the immediate address value to the PC register.

- SRCTSP
- DECSP
- IRLTPC

4.18 RET

In this operation, we basically fill the PC register with the previous value that we pushed to the stack in CALL operation. It is exactly the same as the PUL operation but the destination register is the program counter.

- INCSP
- SPTSRC

5 Results

We have converted the code given to us in the homework file into binary code and run it on our virtual machine. We have also tested numerous cases we have prepared for our operations. We had seen expected results on memory.

5.1 Example Programs

The program codes you see below should be entered to the memory in reversed order, ie. if you see 41 00 you should enter 00 41 to correctly operate the code.

A program that writes 64 into 64th memory cell:

- LD R0 IM 64 – 40 00
- MOV AR R0 – 00 16
- ST R0 D – 00 09

A program that increments the registers and carry it to the others.

- INC R1 R0 – 00 41
- INC AR R1 – 32 46
- INC SP AR – c0 47
- INC R2 SP – e0 42

A program that negates the registers and carry it to the others.

- NOT R1 R0 – 00 59
- NOT AR R1 – 20 5e
- NOT SP AR – c0 5f
- NOT R2 SP – e0 5a

A program that shifts the registers logical left and carry it to the others.

- LSL R1 R0 – 00 61
- LSL AR R1 – 20 66
- LSL SP AR – c0 67
- LSL R2 SP – e0 62

6 Discussion

We learned how to build a basic computer using microprogrammable control unit. Designing the branching logic and format of the microinstructions was the first step we managed to do. After designing those, we specified what microoperations we need to implement given operations.

7 Conclusion

We have come to the conclusion that in software based control unit we designed, we can implement many more operations (even more than we needed to do) compared to the hardware based control units. There will be less need for additional circuit designs and we can directly implement new operations by writing their code's (consists of microinstructions) into their respective places in ROM.