ICS233-COE301 project document of

Designing a single-cycle processor

Name: Abdullah Alzeid

ID: 201834480

Table of contents

Contents

Design and implementation

Design choices

- We decided to use decoders over multiplexers in the register file because it has better performance.

- We have decided to divide the to build a different 16-bit adder as shown in **figure.8** to get the value of the overflow needed to find the value of SLT instruction in the ALU.

- We have decided to abandon the idea of using the zero flag since it is the complement of the ALU result so as an alternative solution we have designed a branch control component as shown in **figure.4** where it generates a signal to activate a branch when the branch condition is satisfied. Moreover, connecting BGE || BGEU and BNE parts with an AND gate in the branch control will offer the flexibility to integrate BGT and BGTU instructions.

- We have decided to use demultiplexers over multiple 16-bit adders for every new program counter decision in the Datapath as shown in **figure.1** to improve the performance.

- We have solved the issue of branching when the condition of the true and continuing when the condition is false by using an OR gate as shown in **figure.3** with branch signal and the result of ANDing the result bits of the PCCADD to select whether bit-1 should be 1 or 0. So, if it was 1 it means that it is either JALR or a branch instruction while if it was 0 so it is a JUMP, false branch condition or it is not a control flow instruction. Then we have we have fixed bit-0 to be 1 with the ORing result to be ANDed with PCADD result to give the true PCADD signal needed.

- We have decided to build an Opcode checker as shown in **figure.5** to check which branch signal should be activated to avoid false branch signals.

Datapath components diagrams

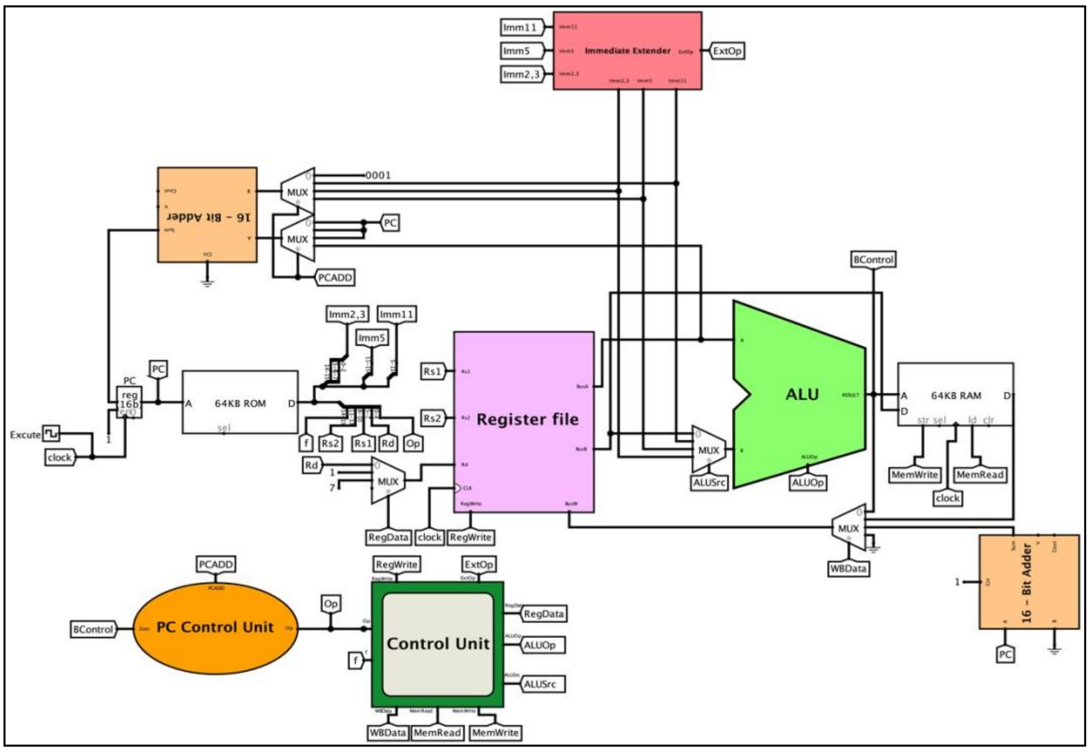


Figure.1 Datapath of the designed single cycle processor.

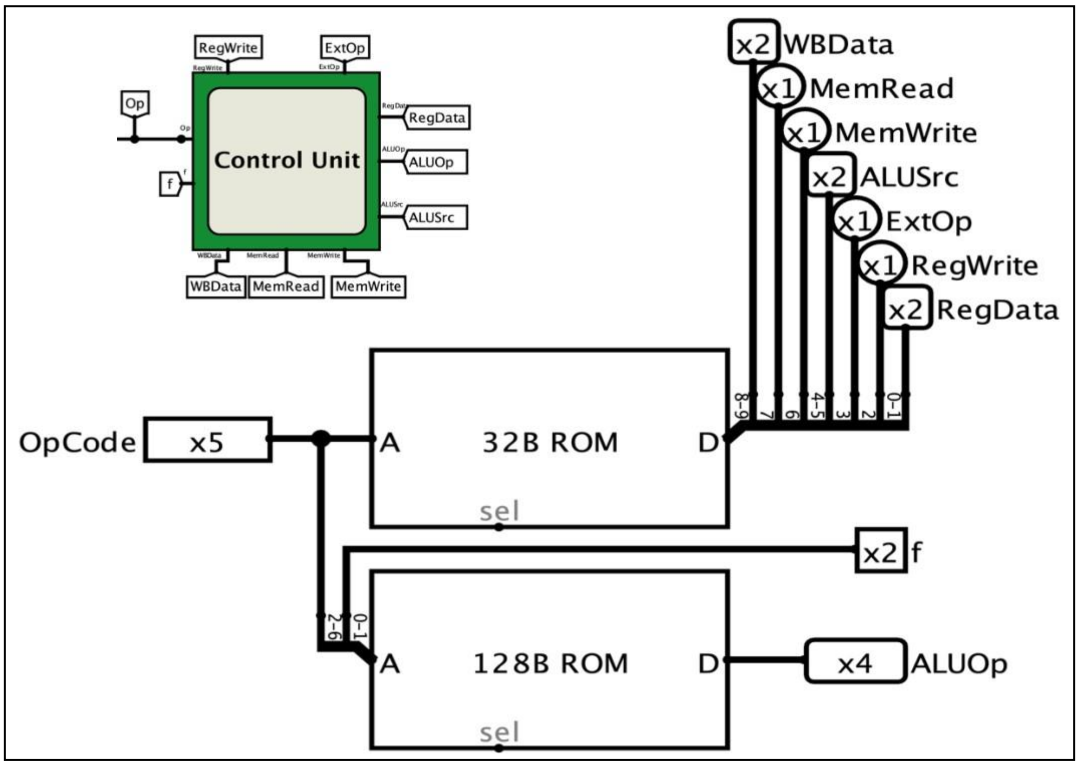


Figure.2 Control unit of the designed single cycle processor.

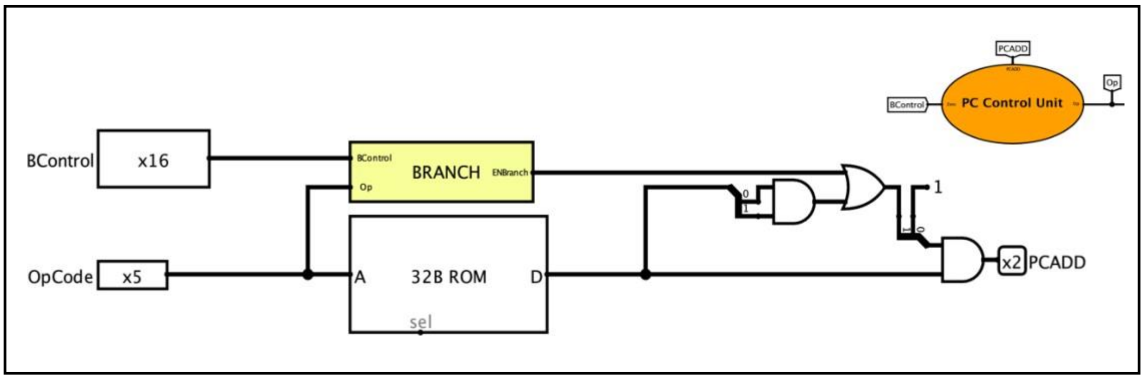


Figure.3 Program counter control unit of the designed single cycle processor.

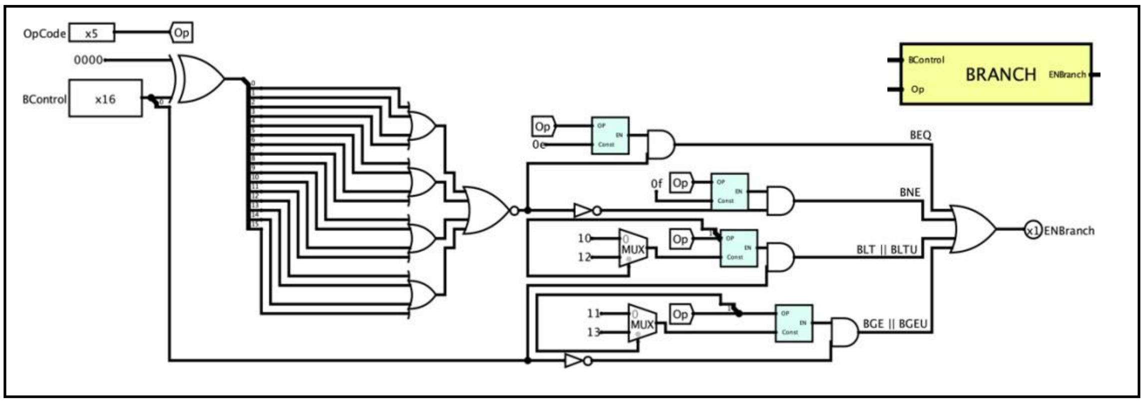


Figure.4 Branch control unit of the designed single cycle processor.

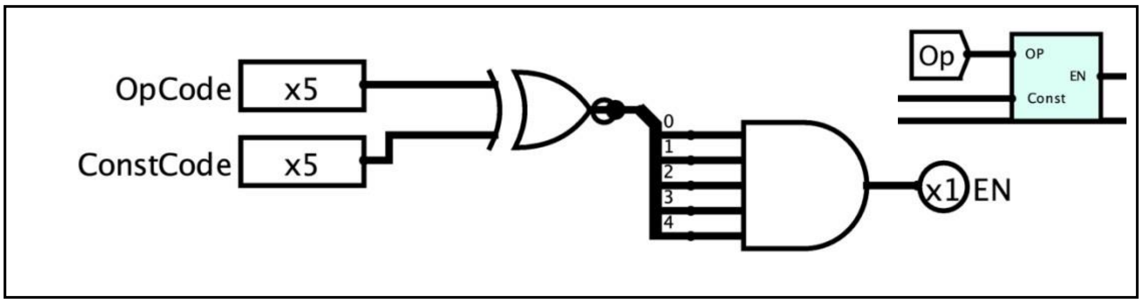


Figure.5 Opcode match checker that is implemented in branch control unit.

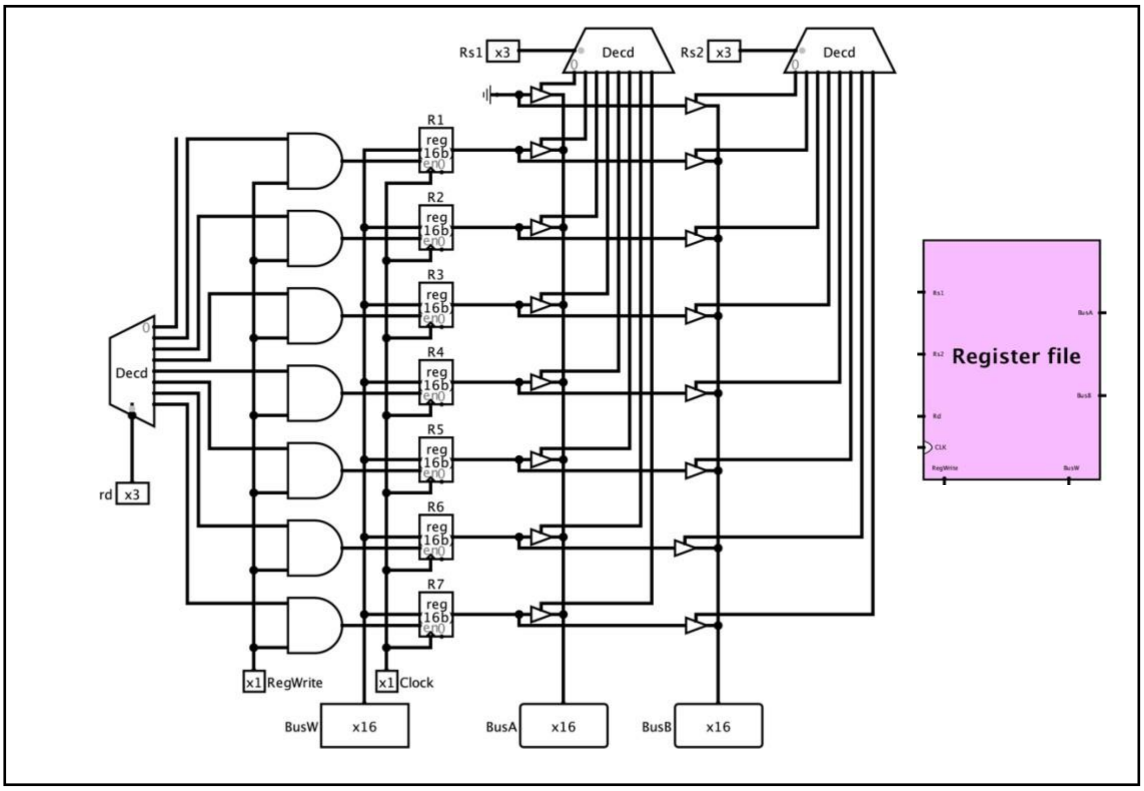


Figure.6 Register file of the designed single cycle processor.

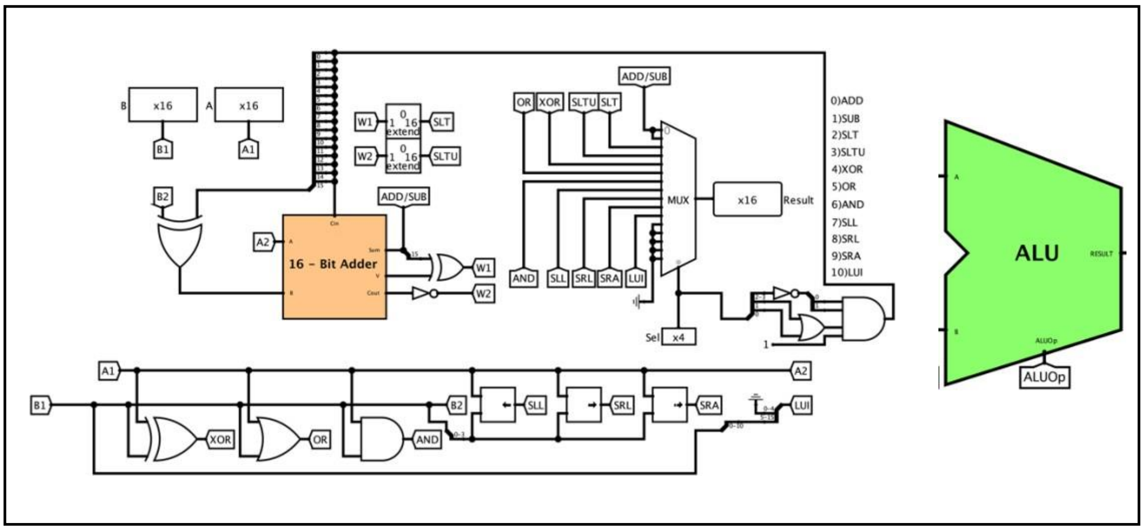


Figure.7 Arithmetic logic unit of the designed single cycle processor.

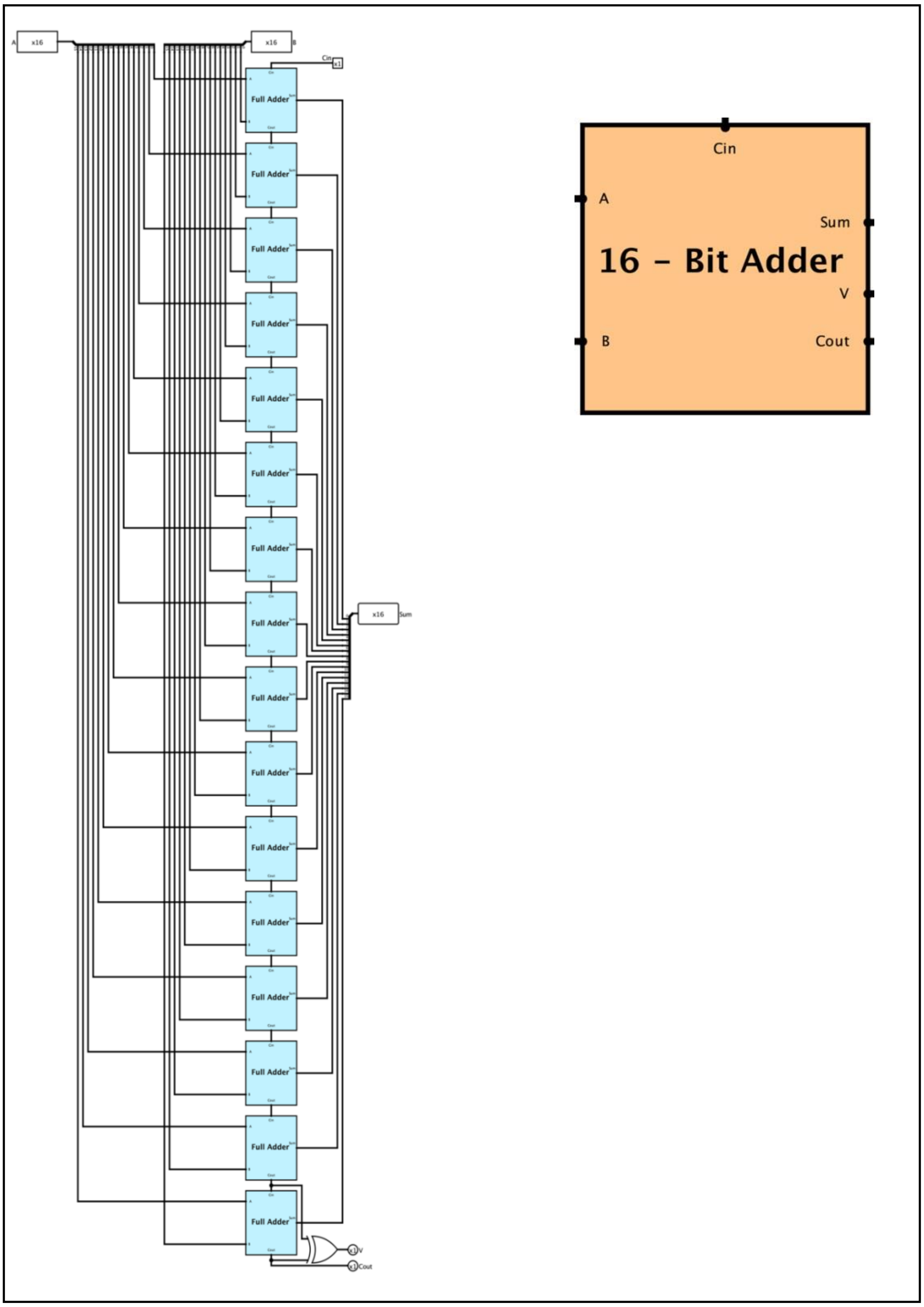


Figure.8 16-bit adder arithmetic logic unit

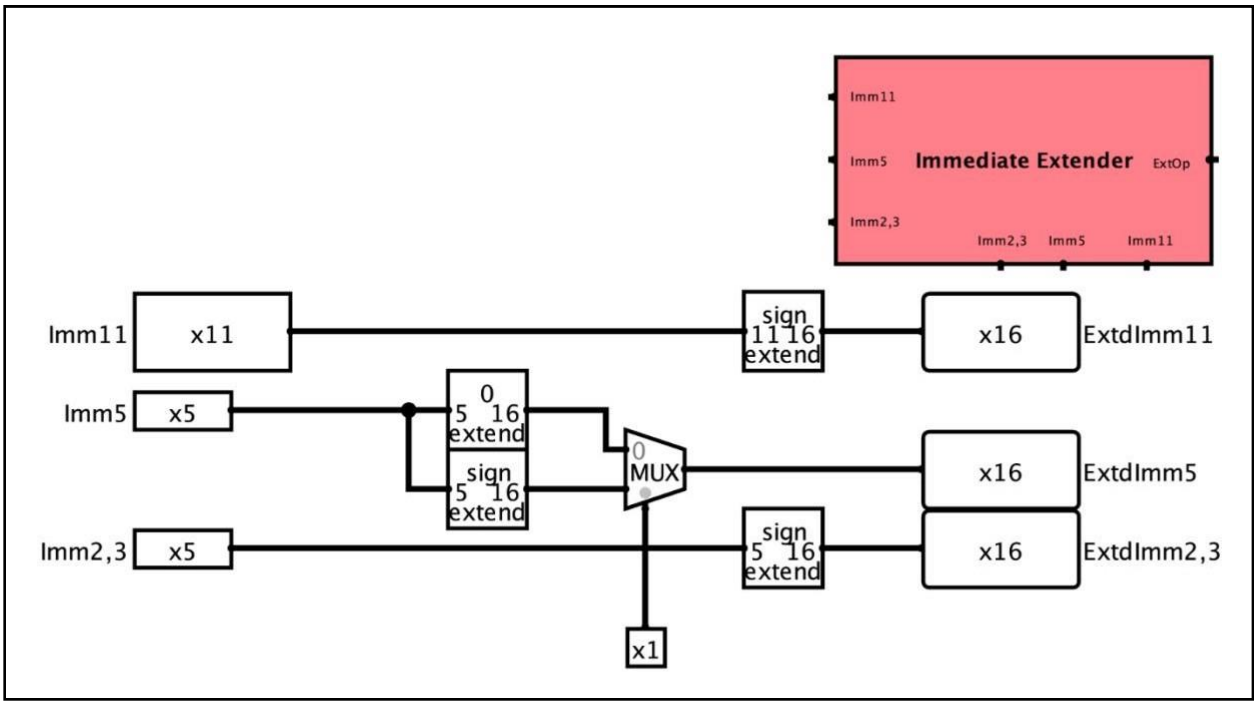


Figure.9 Immediate extender of the designed single cycle processor.

Control signals and their description

Control signals description table	
Datapath control unit signals	
RegData	Controls which register will be written on, whether if it's Rd of choice (00), or R1 for LUI (01), or R7 for JAL (10).
RegWrite	Controls whether if there's a register that will be written on or not. 1 means that there's a register to write on, and 0 means the opposite.
ExtOP	Controls which method of shifting will be applied, 0 means zero-shift is used, 1 means sign-shift is used.
ALUSrc	Controls whether the value of B will be from BusB (00), Imm11(01), Imm5 (10), or Imm2, 3 (11).
MemWrite	Controls whether we write some value on the RAM or not, 1 means that writing on the RAM is activated, 0 means the opposite.
MemRead	Controls whether we read a value from the RAM or not, 1 means that reading from the RAM is activated, 0 means the opposite.
WBData	Controls which value or a source's value will be stored in the written register, 00 means the value will be from the ALU, 01 means the value will be from the memory, 10 means the value will be equal to PC + 1, 11 means the value will be zero.
ALU Control signals	
ALUOp	Controls which arithmetical or logical operation should be performed from A and B to get a certain result. (0000) For addition, (0001) for subtraction, [0010) for SLT, (0011) for SLTU, (0100) for XOR, (0101) for OR, (0110) for AND, (0111) for SLL, (1000) for SRL, (1001) for SRA and (1010) for LUI.
PC control unit signals	
PCCADD	Controls which program counter address should be. (00) for PC + 1, (01) for PC $\pm$ imm11, (10) for PC $\pm$ imm2, 3 and (11) for PC $\pm$ imm5.

Instructions control signals

Instructions control signals table											
Instruction	OP 5 bits	f 2 bits	WBData 2 bits	MemRead 1 bit	MemWrite 1 bit	ALUSrc 2 bits	ExtOp 1 bit	RegWrite 1 bit	RegData 2 bits	ALU Instruction 4 bits	PCADD 2 bits
XOR	0	0	00	0	0	00	0	1	00	0100	00
OR	0	01	00	0	0	00	0	1	00	0101	00
AND	0	10	00	0	0	00	0	1	00	0110	00
SLL	1	00	00	0	0	00	0	1	00	0111	00
SRL	1	01	00	0	0	00	0	1	00	1000	00
SRA	1	10	00	0	0	00	0	1	00	1001	00
ADD	2	00	00	0	0	00	0	1	00	0000	00
SUB	2	01	00	0	0	00	0	1	00	0001	00
SLT	2	10	00	0	0	00	0	1	00	0010	00
SLTU	2	11	00	0	0	00	0	1	00	0011	00
ADDI	3	XX	00	0	0	10	1	1	00	0000	00
SLTI	4	XX	00	1	1	10	0	1	00	0010	00
SLTIU	5	XX	00	1	1	10	0	1	00	0011	00
XORI	6	XX	00	0	0	10	0	1	00	0100	00
ORI	7	XX	00	0	0	10	0	1	00	0101	00
ANDI	8	XX	00	0	0	10	0	1	00	0110	00
SLLI	9	XX	00	0	0	10	0	1	00	0111	00
SRLI	10	XX	00	0	0	10	0	1	00	1000	00
SRAI	11	XX	00	0	0	10	0	1	00	1001	00
LW	12	XX	01	1	0	10	1	1	00	XXXX	00
SW	13	XX	00	0	1	11	1	0	00	XXXX	00
BEQ	14	XX	XX	X	X	XX	X	X	XX	0001	10
BNE	15	XX	XX	X	X	XX	X	X	XX	0001	10
BLT	16	XX	XX	X	X	XX	X	X	XX	0010	10
BGE	17	XX	XX	X	X	XX	X	X	XX	0010	10
BLTU	18	XX	XX	X	X	XX	X	X	XX	0011	10
BGEU	19	XX	XX	X	X	XX	X	X	XX	0011	10
LUI	20	XX	00	0	0	01	0	1	01	1010	00
J	21	XX	XX	X	X	XX	X	X	XX	XXXX	01
JAL	22	XX	10	0	0	01	0	1	10	XXXX	01
JALR	23	XX	10	1	1	10	1	1	00	XXXX	11

Simulation and Testing

Program 1: Test all instructions

Brief description
This program is briefly about testing all instructions to find if they work or not.

Instructions table

Assembly Instruction	HEX	Comment/ Remark
`addi $1,$0,1`	0823	R1 will have the value = 1
`addi $2,$0,1`	0843	R2 will have the value = 1
`xor $3,$1,$2`	1160	R3 will have the value = 0
`or $3,$1,$2`	5160	R3 will have the value = 1
`and $3,$1,$2`	9160	R3 will have the value = 1
`sll $3,$1,$2`	1161	R3 will have the value = 2
`srl $3,$1,$2`	5161	R3 will have the value = 0
`sra $3,$1,$2`	9161	R3 will have the value = 0
`add $3,$1,$2`	1162	R3 will have the value = 2
`sub $3,$1,$2`	5162	R3 will have the value = 0
`addi $2,$0,0`	0043	R3 will have the value = 0
`slt $3,$2,$1`	8a62	R3 will have the value = 1
`sltu $3,$2,$1`	ca62	R3 will have the value = 1
`slti $3,$2,1`	0a64	R3 will have the value = 1
`sltiu $3,$2,1`	0a65	R3 will have the value = 1
`xori $3,$1,1`	0966	R3 will have the value = 0
`ori $3,$1,1`	0967	R3 will have the value = 1
`andi $3,$1,1`	0968	R3 will have the value = 1
`slli $3,$1,1`	0969	R3 will have the value = 2
`srli $3,$1,1`	096a	R3 will have the value = 0
`srai $3,$1,1`	096b	R3 will have the value = 0
`sw $1,0($4)`	0c0d	Going to store the value of R1 in R4 memory
`lw $5,0($4)`	04ac	Going to load the value from R4 to R5
`addi $2, $0,1`	0843	R2 will have the value = 1
`beq $1, $2, bne`	114e	If (R1==R2) go to bne label else jump to label end
`j end`	0215	Jump to label end
`bne:`	-	Label
`addi $2,$0,3`	1843	R2 will have the value = 3
`bne $1,$2,blt`	114f	If (R1!=R2) go to blt label else jump to label end
`j end`	01b5	Jump to label end

Assembly Instruction	HEX	Comment/ Remark
`blt:`	-	Label
`addi $2,$0,3`	1843	R2 will have the value = 3
`blt $1,$2,bge`	1150	If (R1<R2) go to bge label else jump to label end
`j end`	0155	Jump to label end
`bge:`	-	Label
`addi $2,$0,3`	1843	R2 will have the value = 3
`bge $2,$1,bltu`	0a51	If (R1>=R2) go to bltu label else jump to label end
`j end`	00f5	Jump to label end
`bltu:`	-	Label
`addi $2,$0,3`	1843	R2 will have the value = 3
`bltu $1,$2,bgeu`	1152	If (R1<R2) go to bgeu label else jump to label end
`j end`	0095	Jump to label end
`bgeu:`	-	Label
`addi $2,$0,3`	1843	R2 will have the value = 3
`bgeu $2,$1,end`	0a53	If (R1>=R2) go to end label else jump to label end
`j end`	0035	Jump to label end
`end:`	-	Label
`lui 5`	00b4	R1 will have the value = 5
`j testj`	0035	Jump to label testj
`testj:`	-	Label
`addi $1,$0,1`	0823	R1 will have the value = 1
`addi $2,$0,1`	0843	R2 will have the value = 1
`jal testjal`	0036	Jump to testjal label and save the address in R7
`testjal:`	-	Label
`jalr $0,$7,0`	0717	The PC value will be updated to R7 + 0. **Remark:** because R0 is used in the instruction which cannot be written into, Rd will not be updated to PC + 1. So, it will work as well as jr R7 instruction.

Program 2: Counting number of ones

Brief description
The program is briefly about counting the number of ones (high bits) in a specific 16-bit register by reading every digit and ANDing it with a one. If the result is one it will one counter.

Instructions table

Assembly Instruction	HEX	Comment/Remark
`lui 1`	`0034`	R1 will have the value of 32 (100000).
`srli $2, $1, 1`	`094a`	R2 will have the value of 16 (010000).
`lui 7`	`00f4`	R1 will have the value of 112 (1110000).
`ori $3, $1, 7`	`3967`	R2 will have the value of 119 (1110111).
`addi $1, $0, 1`	`0823`	R1 will have the value of 1 (0001).
`addi $5, $0, 1`	`08a3`	R5 will have the value of 1 (0001).
`loop:`	-	Label for the start of the counting 1s loop.
`bge $1, $2, end`	`11f1`	If R1 is greater than or equal to R2 (16), go to end label.
`andi $6, $3, 1`	`0bc8`	R6 will have the value of 1 if $3's first digit is equal to 1.
`srli $3, $3, 1`	`0b6a`	R3 will be zero-shifted by 1 bit to the right.
`addi $1, $1, 1`	`0923`	R1 will be incremented by 1.
`bne $5, $6, loop`	`f58f`	If R6 is not equal to R5 (1) then go back to label loop.
`addi $4, $4, 1`	`0c83`	R4 will be incremented by 1.
`j loop`	`ff55`	Jump to label loop.
`end:`	-	Label for the end of counting 1's loop.

Snapshots

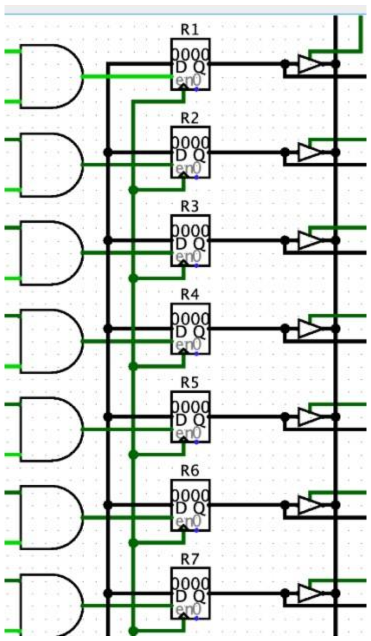


Figure.10 The values of the registers before simulation.

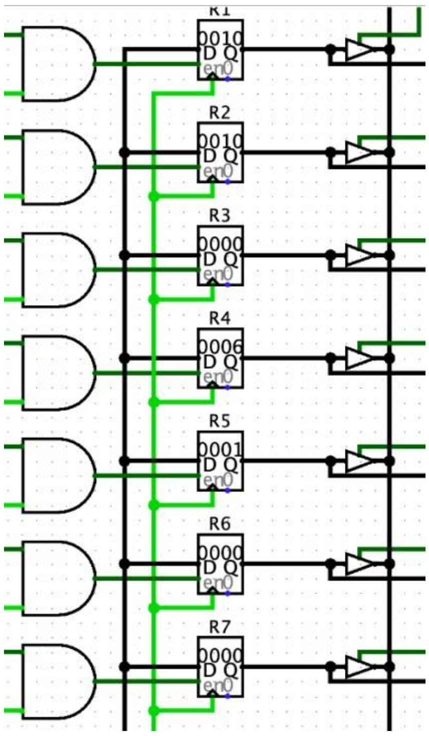


Figure.11 The values of the registers after simulation.

Program 3: Bubble sorting

Brief description

The program will insert n values in the RAM then it will sort it by comparing each index inserted with the one next to it. if the current index is greater, it will swap it with the next one. The program will repeat the operation n^2 times.

Instructions table

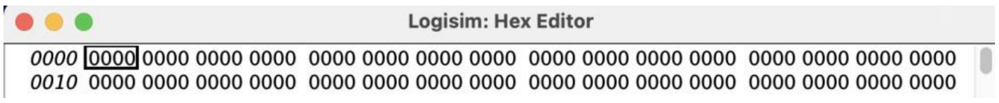
Assembly Instruction	HEX	Comment/ Remark
`addi $3, $0, 0`	0063	R3 will have the value of R0 + 0. R3 = 0 + 0 = 0 **Remark:** R3 is used to specify the index of the loop.
`addi $2, $0, 8`	4043	R2 will have the value of R0 + 8. R2 = 0 + 8 = 8 **Remark:** R2 is used to specify the maximum number of iterations of the loop.
`lui 440`	3714	R1 will have the value of $(110111000\_00000)_2$ = $(3700)_{16}$.
`ori $4, $1, 10`	5187	R4 will have the value of $(110111000\_01010)_2$ = $(370A)_{16}$.
`entryLoop:`	-	A label that indicates that the entry loop starts at this line.
`bge $3, $2, endEntry`	13b1	If (R3 ≥ R2) the loop will end. In other words, the program counter will branch to *endEntry* label.
`addi $4, $4, -8`	c483	R4 will have the value of (R4 – 8) in every iteration in the loop. R4 = R4 – 8. **Remark:** the first value of R4 was $(370A)_{16}$ before subtraction, the value after subtraction is $(3702)_{16}$ and this value will be updated in every iteration
`sw $4, 0($3)`	230d	The value of R4 will be stored in the RAM in index of R3 which is 0 in the first iteration.
`addi $3, $3, 1`	0b63	R3 will have the value of (R3 + 1) which was 0 in the first iteration. **Remark:** the value of R3 will be updated in every iteration.
`j entryLoop`	ff95	The PC value will be updated to PC ±(immediate11). **Remark:** PC value will be updated until the loop ends.

Assembly Instruction	HEX	Comment/ Remark
`endEntry:`	-	A label that indicates that the entry loop ends at this line.
`jal sort`	0056	The PC value will be updated to PC $\pm$(immediate11) and R7 will have the value of PC + 1. In other words, the program counter will jump to the *sort* label and will save the next instruction address in R7. **Remark:** the value of R7 should not be modified until the sorting method ends.
`j end`	0235	The PC value will be updated to PC $\pm$(immediate11). **Remark:** when this instruction is executed, the PC will jump to *end* label which will end the program
`sort:`	-	A label that indicates that the sorting method starts at this line.
`addi $3, $0, 0`	0063	R3 will have the value of 0. R3 = 0 + 0 = 0 **Remark:** R3 is used to specify the index of the first loop.
`firstLoop:`	-	A label that indicates that the first loop starts at this line.
`bge $3, $2, endFirst`	53d1	If (R3 $\geq$ R2) the loop will end. In other words, the program counter will branch to *endFirst* label.
`addi $4, $0, 0`	0083	R4 will have the value of 0. R4 = 0 + 0 = 0 **Remark:** R4 is used to specify the index of the second loop.
`secondLoop:`	-	A label that indicates that the second loop starts at this line.
`addi $1, $3, 1`	0b23	R1 will have the value of R3 + 1 which is i +1. **Remark:** the value of R1 will be updated in every iteration because the value of R3 will be updated in every iteration.
`sub $6, $2, $1`	4ac2	R6 will have the value of R2 – R1 which is R2 – (i + 1). **Remark:** the value of R6 will be updated in every iteration because the value of R1 will be updated in every iteration.

Assembly Instruction	HEX	Comment/ Remark
`bge $4, $6, endSecond`	7411	If (R4 $\geq$ R6) the loop will end. In other words, the program counter will branch to *endSecond* label.
`lw $1, 0($4)`	042c	The value that is stored in the RAM in index R4 will be loaded into R1. **Remark:** this instruction will be executed until the second loop ends. **Comment:** We have used R1 in this instruction because R1 is used in an instruction above and will be updated in every iteration.
`lw $6, 1($4)`	0ccc	The value that is stored in the RAM in index R4 + 1 will be loaded into R6. **Remark:** this instruction will be executed until the second loop ends. **Comment:** We have used R6 in this instruction because R6 is used in an instruction above and will be updated in every iteration.
`blt $1, $6, noSwap`	3170	If (R1 < R6) the program will not swap. In other words, the program counter will branch to *endSecond* label.
`Swapping:`	-	A label that helps to trace the code better.
`sw $1, 1($4)`	0c2d	The value of R1 will be stored in the RAM in index of R4 + 1. **Remark:** this instruction will be executed until the second loop ends.
`sw $6, 0($4)`	340d	The value of R6 will be stored in the RAM in index of R4. **Remark:** this instruction will be executed until the second loop ends.
`noSwap:`	-	A label that indicates that there is no need for swapping (skip swapping).
`addi $4, $4, 1`	0c83	R4 will have the value of (R4 + 1). **Remark:** the value of R4 will be updated in every iteration of the second loop.
`j secondLoop`	fef5	zThe PC value will be updated to PC $\pm$(immediate11). **Remark:** 1- When this instruction is executed, the PC will jump to *secondLoop*. 2- This instruction will be executed until the second loop ends.

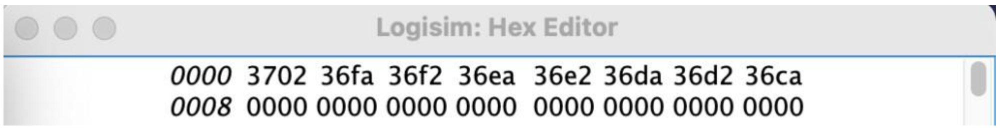
Assembly Instruction	HEX	Comment/ Remark
`endSecond:`	-	A label that indicates that the second loop ends at this line.
`addi $3, $3, 1`	`0b63`	R3 will have the value of (R3 + 1). **Remark:** the value of R3 will be updated in every iteration of the first loop.
`j firstLoop`	`fe75`	The PC value will be updated to PC $\pm$(immediate11). **Remark:** *1-* When this instruction is executed, the PC will jump to *firstLoop.* 2- This instruction will be executed until the first loop ends.
`endFirst:`	-	A label that indicates that the second loop ends at this line.
`jalr $0, $7, 0`	`0717`	The PC value will be updated to R7 + 0. Remark: because R0 is used in the instruction which cannot be written into, Rd will not be updated to PC + 1. So, it will work as well as jr R7 instruction.
`end:`	-	A label that indicates that the program ends at this line.

Snapshots



```
0000 0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000
0010 0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000
```

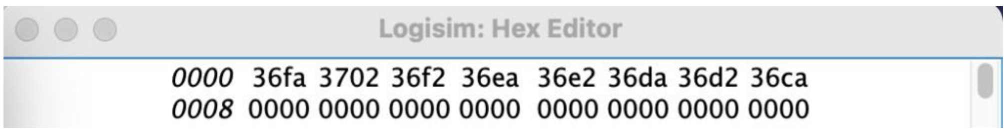
Figure.12 RAM before values entry



```
0000 3702 36fa 36f2 36ea  36e2 36da 36d2 36ca
0008 0000 0000 0000 0000  0000 0000 0000 0000
```

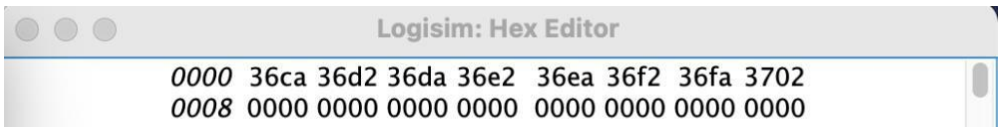
Figure.13 RAM after values entry

Snapshots


Logisim: Hex Editor

```
0000  36fa 3702 36f2 36ea  36e2 36da 36d2 36ca
0008  0000 0000 0000 0000  0000 0000 0000 0000
```

Figure.14 RAM after first sorting operation


Logisim: Hex Editor

```
0000  36ca 36d2 36da 36e2  36ea 36f2 36fa 3702
0008  0000 0000 0000 0000  0000 0000 0000 0000
```

Figure.15 RAM after sorting has been completed

Teamwork

Teamwork distribution table. (%)						
Part \ Member	Omar		Mohammed		Abdulrahman	
CPU design	Datapath	33.3%	Datapath	33.3%	Datapath	33.3%
	Control Unit	33.3%	Control Unit	33.3%	Control Unit	33.3%
	PC control unit	45%	PC control unit	45%	PC control unit	10%
	Branch control	33.3%	Branch control	33.3%	Branch control	33.3%
	Opcode match	100%	Opcode match	-	Opcode match	-
	ALU	33.3%	ALU	33.3%	ALU	33.3%
	Immediate extender	50%	Immediate extender	25%	Immediate extender	25%
	Register file	-	Register file	-	Register file	100%
	Full Adder	100%	Full Adder	-	Full Adder	-
	16-bit adder	100%	16-bit adder	-	16-bit adder	-
Control signals	R- type	-	R- type	-	R- type	100%
	I - type	-	I - type	100%	I - type	-
	SB - type	10%	SB - type	75%	SB - type	15%
	J - type	100%	J - type	-	J - type	-
Instructions testing	Code writing	-	Code writing	-	Code writing	100%
	Code documentation	-	Code documentation	-	Code documentation	100%
	Testing	-	Testing	-	Testing	100%
Counting 1's	Code writing	15%	Code writing	70%	Code writing	15%
	Code documentation	-	Code documentation	100%	Code documentation	-

Teamwork distribution table. (%)						
Part \ Member	Omar		Mohammed		Abdulrahman	
Counting 1's	Testing	15%	Testing	70%	Testing	15%
Bubble sorting	Code writing	100%	Code writing	-	Code writing	-
	Code documentation	100%	Code documentation	-	Code documentation	-
	Testing	100%	Testing	-	Testing	-
Project document construction	40%		30%		30%	