Shayan Pournazari
Syed Khalid
Demitrius Smith

Project 3 Report

Group 8

Introduction:
Our Zero Gravity software aims to provide simplicity and efficiency for our PRPG program. We worked within the specified criteria and created a program to accurately and quickly decode our PRPG instructions.

Instruction Table: The following is our instruction table complete with the format, opcodes, and examples:

PC	Coding	Instruction	Functionality	Example
-	11111111	exit	exit	-------
PC++	0000	STORE	MEM[Ry] = Rx	store R2,R3 00001011
PC++	0001 Rx Ry	LOAD	Rx = MEM[Ry]	load R2,R3 00011011
PC = R(i)	0010 iiii	JUMP	PC = R(imm)	Jump 6 00100110
PC++	0011	MARK	Rx = PC + 4	Mark 1 00110001
PC++	0100 iiii	INIT	R0 = imm [imm = (-8,7)]	init 6 01000110
PC++	0101 Rx Ry	ADD	Rx=Rx + Ry	add R2,R2 01011111
PC++	0110 Rx Ry	SPLIT	Rx = R0 & 0xFF Ry = R0 >> 8	split R1,R2 01100110
PC++	0111 Rx ii	ADDI	Rx = Rx +imm	addi R3, 2 01111101
PC++	1000 Rx ii	SRL	Rx = Rx >>imm	Srl R2, 3 10001011
PC++	1001 Rx Ry	SAND	Rx =(Ry&1)-1	sand R3,R2 10011110
PC++	1010 Rx Ry	SUB	Rx = Rx-Ry	sub R1,R3 10100111
PC++	1011 Rx ii	BEQ	If Rx=0, Branch imm else , PC +4	Beq R1, 2 10110110

PC++	1100 Rx Ry	SLT	If Rx<Ry than R0 = 1 Else R0 = 0	slt R1,R0 11000100
PC++	1101 Rx Ry	XOR	Rx = Rx XOR Ry	xor R0,R3 11010011
PC++	1110 Rx Ry	DC	Drop and Combine	dc R2,R3 11101011

As you see we've used every 4-bit combination for our Opcodes(except 1111, which is used for terminating the program).

Registers: We have support for Registers 1-7. R1-R3 are standard registers, both represented by two bits. We included support for register 4 in four different instructions, Store, SLT, ADDi, and Sub. We examined these instructions and determined that we were able to include a fourth register(for only these instructions) if were were to replace a register that we weren't using in our code. R5 was a constant register, in which it would always equal 23(just as in Mips R0 is always 0). This served a purpose that we no longer needed, but was good to include. R6 and R7 are exclusive to Jump and Mark. These are used to jump to where we need to go.

Branching: Our program supports "Jump" and "Branch If Equal" instructions. Our beq instruction can only function for up to 3(4x3 PC) spaces. We were able to work around this limitation however. The combination of "Mark" and "Jump" allows us to go virtually anywhere we're looking to go in our code though.

Data Memory: Load and Store are both fully supported. We can support range of addresses equal to the max amount a register could hold. We stop at 48 though.

Overall, our code works fairly well. If we were allowed an extra bit we'd be able to clean up the program quite a bit. We could support a lot more registers and our instructions that include immediates would increase, allowing us to cut down on our instructions. If we were down 1 bit we'd be in quite a bit of trouble. We would need to be very conservative with our use of registers, otherwise we would need to get rid of half of our opcodes. It would still be possible to perform though, it would just take a bit more planning.

The advantages of our ISA is the amount of opcodes we were able to extract. We use 15 opcodes, which allows us a plethora of operations. This allows us to efficiently calculate anything that we need to calculate. This puts us at a little disadvantage however, because compromising all of our opcodes into 4 bits means that some of our immediate values are condensed to a maximum of 2. However, for our code, it works well. If we were to restart fresh, we would definitely begin work on the design much earlier. We built our ISA but ended up changing just about everything not too long before the due date. We would have been able to foresee this if we were able to design much earlier than we did.

Activity Log

Date : Time	Location	Progress : Activity	Members
3/19/19 : 10AM-12PM	Library	Going through project : Set objectives	Shayan/Syed/Demitrius
3/21/19 : 8PM-10PM	Discord	Talking about who does what : 10%	Shayan/Syed/Demitrius
3/23/19 : 7PM-9PM	Discord	Trying to figure a design out : 20%	Shayan/Syed/Demitrius
3/25/19 : 4PM-9PM	Library	Thinking of a different design : 30%	Shayan/Syed/Demitrius
3/27/19 : 8PM-10PM	Discord	Working on the project : 50%	Shayan/Syed/Demitrius
3/28/19 : 7PM-10PM	Discord	Working on the project : still trying to figure out everything perfectly	Shayan/Syed/Demitrius
3/29/19 : 11AM-6PM	Apartment	Fixing bugs :65%	Shayan/Syed/Demitrius
3/30/19 : 10AM-3PM	Library	Fixing bugs : 90%	Shayan/Syed/Demitrius
4/1/19 : 12PM-4PM	Library	Final touches	Shayan/Syed/Demitrius

NOTE BY: SYED KHALID

TO THE GRADER.

I HAD TO ALL THE ARCHITECTURE BY MYSELF, THEY DID NOT KNOW EXACTLY WHAT WAS GOING ON. THE FIRST ISA I MADE WAS NOT VERY ACCORDING TO THE GUIDELINES. I ASKED THEM TO HELP BUT THEY HARD CODED EVERYTHING I HAD A LOT OF HARD TIME WORKING ALONG WITH THEM.

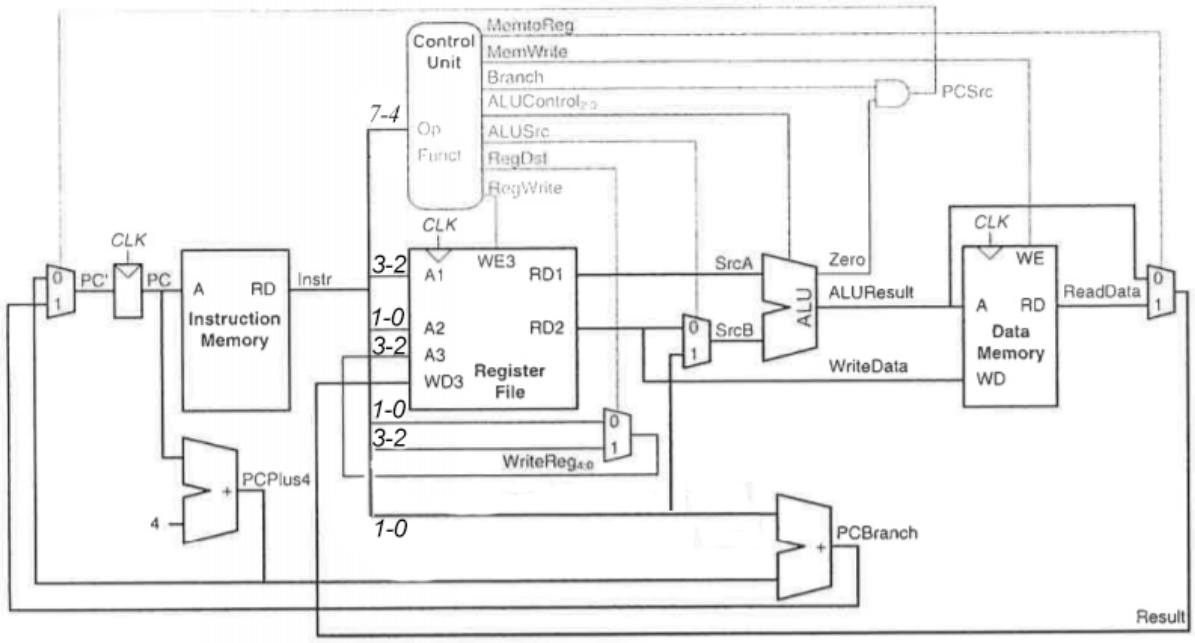
THEY BOTH DID THE REPORT AND THE HARDWARE WHICH I DON'T KNOW HOW GOOD IS. AS FAR AS SOFTWARE AND WHOLE ARCHITECTURE CONCERN I DID THAT BY MESELF. THEY BOTH DID NOT EVEN KNOW HOW TO USE GITHUB.

THIS ACTIVITY LOG ALSO MADE BY THEM. WE ONLY MET THREE TIMES. IN WHICH WE ONLY DISCUSSED. I HAD TO SPENT HOURS WORKING ON THE

SOFTWARE PART. PLEASE GRADE ME ON MY EFFORT.

Hardware Implementation

Datapath Design:

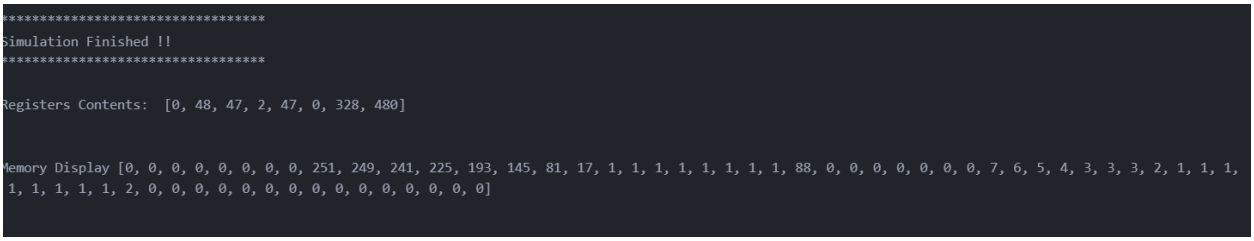


Control Logic Design:

Instructions	REgWrite	RegDst	ALUSrc	Branch	MmWrite	MemtoReg	ALUOp	jump	jalctrl
STORE	0	x	1	0	1	x	00	0	0
LOAD	1	0	1	0	0	1	00	0	0
JUMP	0	x	x	x	0	x	x	1	0
MARK	1	0	1	0	x	0	x	0	0
INIT	1	x	1	1	1	1	1	0	0
ADD	1	1	0	0	0	0	10	0	0
SPLIT	1	1	1	x	0	1	1	0	0
ADDI	1	0	1	0	x	0	x	x	x
SRL	1	1	0	0	0	0	11	x	0

SAND	1	1	0	0	0	0	10	0	0
SUB	1	1	0	0	0	0	10	0	0
BEQ	0	x	0	1	0	x	01	0	0
SLT	1	1	0	0	0	0	10	0	0
XOR	1	1	0	0	0	0	10	0	0
DC	1	1	0	0	0	0	10	0	0

SCREENSHOTS:



ASSEMBLY FILE FOR ALL OF THE SEEDS:

.asm for s3

```
Init 3
add R1,R0
add R3,R0
addi R4,3
addi R4,3
addi R4,2
store R1,R4
init 1
add R2,R0
mark R6
slt R2,R1
beq R0,3
add R3,R1
addi R2,1
Jump R6
 add R0,R3
addi R4,1
Split R1,R3
dc R1,R3
xor R1,R1
add R1,R0
store R1,R4
init 7
xor R2,R2
add R2,R0
add R2,R2
add R2,R0
addi R2,2
slt R4,R2
beq R0,5
xor R2,R2
```

```
addi R2,1
xor R3,R3
add R3,R1
Jump R6


xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R1,R0
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
slt R1,R2
beq R0,1
 Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
addi R4,3
addi R4,3
```

```
addi R4,2
xor R2,R2
addi R2,3
addi R2,3
addi R2,2
load R1,R2
xor R2,R2
Mark R6
PC 272 sand R0,R1
beq R0,3
srl R1,1
beq R1,3
Jump R6
addi R2,1
srl R1,1
beq R1,1
Jump R6
xor R3,R3
addi R4,1
store R2,R4
xor R1,R1
init 7
add R0,R0
add R1,R0
addi R1,3
addi R1,3
addi R1,3
addi R1,1
xor R0,R0
add R0,R1
Sub R1,R4
```

```
addi R1,1
Sub R0,R1
beq R0,3
load R1,R1
xor R2,R2
Jump R6
xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R0,R0
add R0,R0
add R1,R0
addi R1,3
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,1
slt R1,R2
```

```
beq R0,1
Jump R7
 addi R1,1
srl R3,3
srl R3,1
store R3,R1
```

.ASM for 79

```
 init 7
 add R0,R0
 add R0,R0
 add R0,R0
 add R1,R0
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,3
 addi R1,2
 add R3,R1
 addi R4,3
 addi R4,3
 addi R4,2
 store R1,R4
 init 1
 add R2,R0
 mark R6
```

```
 slt R2,R1
 beq R0,3
 add R3,R1
 addi R2,1
Jump R6
add R0,R3
addi R4,1
Split R1,R3
dc R1,R3
xor R1,R1
add R1,R0
store R1,R4
init 7
xor R2,R2
add R2,R0
add R2,R2
add R2,R0
addi R2,2
slt R4,R2
beq R0,5
xor R2,R2
addi R2,1
xor R3,R3
add R3,R1
Jump R6
xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R1,R0
Mark R7
```

```
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
slt R1,R2
beq R0,1
Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
addi R4,3
addi R4,3
addi R4,2
xor R2,R2
addi R2,3
addi R2,3
addi R2,2
load R1,R2
xor R2,R2
Mark R6
sand R0,R1
beq R0,3
srl R1,1
beq R1,3
```

```
Jump R6



addi R2,1
srl R1,1
beq R1,1
Jump R6
xor R3,R3
addi R4,1
store R2,R4
xor R1,R1
init 7
add R0,R0
add R1,R0
addi R1,3
addi R1,3
addi R1,3
addi R1,1
xor R0,R0
add R0,R1
Sub R1,R4
addi R1,1
Sub R0,R1
beq R0,3
load R1,R1
xor R2,R2
Jump R6
xor R1,R1
xor R2,R2
```

```
xor R3,R3
init 7
add R0,R0
add R0,R0
add R1,R0
addi R1,3
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,1
slt R1,R2
beq R0,1
Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
```

.ASM for 118

```
init 7
add R0,R0
add R0,R0
add R0,R0
add R0,R0
add R1,R0
addi R1,3
addi R1,3
add R3,R1
addi R4,3
addi R4,3
addi R4,2
store R1,R4
init 1
add R2,R0
mark R6
slt R2,R1
beq R0,3
add R3,R1
addi R2,1
Jump R6
add R0,R3
addi R4,1
Split R1,R3
dc R1,R3
xor R1,R1
add R1,R0
```

```
store R1,R4
init 7
xor R2,R2
add R2,R0
add R2,R2
add R2,R0
addi R2,2
slt R4,R2
beq R0,5
xor R2,R2
addi R2,1
xor R3,R3
add R3,R1
Jump R6
xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R1,R0
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
slt R1,R2
```

```
beq R0,1
Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
addi R4,3
addi R4,3
addi R4,2
xor R2,R2
addi R2,3
addi R2,3
addi R2,2
load R1,R2
xor R2,R2
Mark R6
sand R0,R1
beq R0,3
srl R1,1
beq R1,3
Jump R6
addi R2,1
srl R1,1
beq R1,1
Jump R6
xor R3,R3
addi R4,1
store R2,R4
xor R1,R1
init 7
add R0,R0
```

```
add R1,R0
addi R1,3
addi R1,3
addi R1,3
addi R1,1
xor R0,R0
add R0,R1
Sub R1,R4
addi R1,1
Sub R0,R1
beq R0,3
load R1,R1
xor R2,R2
Jump R6
xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R0,R0
add R0,R0
add R1,R0
addi R1,3
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
add R0,R0
xor R2,R2
add R2,R0
```

addi R2,3

addi R2,3

addi R2,3

addi R2,3

addi R2,3

addi R2,3

addi R2,1

slt R1,R2

beq R0,1

Jump R7

addi R1,1

srl R3,3

srl R3,1

store R3,R1

.ASM for 251

init 7

add R0,R0

add R0,R0

add R0,R0

add R0,R0

add R0,R0

add R1,R0

addi R1,3

addi R1,3

addi R1,3

addi R1,3

addi R1,3

```
addi R1,3
addi R1,3
addi R1,3
addi R1,3
add R3,R1
addi R4,3
addi R4,3
addi R4,2
store R1,R4
init 1
add R2,R0
mark R6
slt R2,R1
beq R0,3
add R3,R1
addi R2,1
Jump R6
add R0,R3
addi R4,1
Split R1,R3
dc R1,R3
xor R1,R1
add R1,R0
store R1,R4
init 7
xor R2,R2
add R2,R0
add R2,R2
add R2,R0
addi R2,2
slt R4,R2
```

```
beq R0,5
xor R2,R2
addi R2,1
xor R3,R3
add R3,R1
Jump R6


xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R1,R0
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
slt R1,R2
beq R0,1
Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
```

```
addi R4,3
addi R4,3
addi R4,2
xor R2,R2
addi R2,3
addi R2,3
addi R2,2
load R1,R2
xor R2,R2
Mark R6
sand R0,R1
beq R0,3
srl R1,1
beq R1,3
Jump R6
addi R2,1
srl R1,1
beq R1,1
Jump R6
xor R3,R3
addi R4,1
store R2,R4
xor R1,R1
init 7
add R0,R0
add R1,R0
addi R1,3
addi R1,3
addi R1,3
addi R1,1
xor R0,R0
```

```
add R0,R1
Sub R1,R4
addi R1,1
Sub R0,R1
beq R0,3
load R1,R1
xor R2,R2
Jump R6
xor R1,R1
xor R2,R2
xor R3,R3
init 7
add R0,R0
add R0,R0
add R1,R0
addi R1,3
Mark R7
addi R1,1
load R2,R1
add R3,R2
init 7
add R0,R0
add R0,R0
xor R2,R2
add R2,R0
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
addi R2,3
```

```
addi R2,1
slt R1,R2
beq R0,1
Jump R7
addi R1,1
srl R3,3
srl R3,1
store R3,R1
```