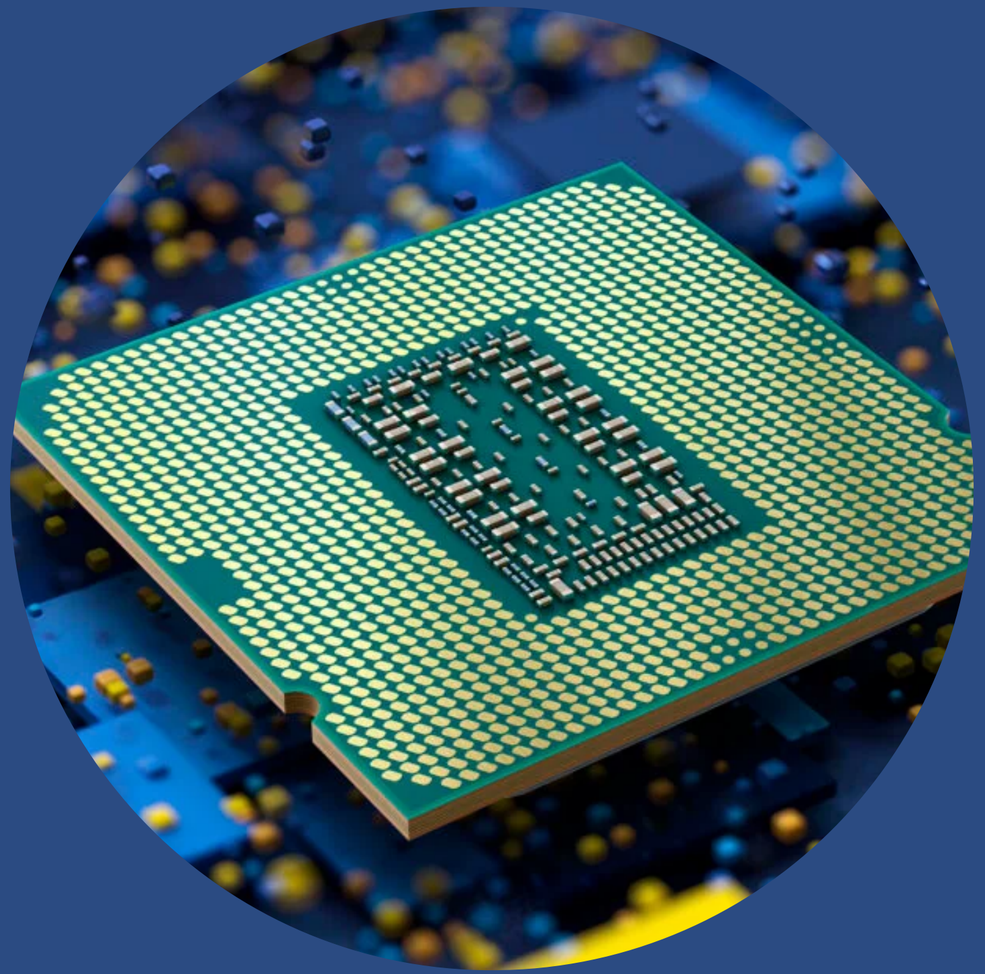
BY RON AND ZHEN

Team Avicenna

Accumulator Processor

Overall Architecture

OUR PROCESSOR IS AN ACCUMULATOR
EVERYTHING THAT IS ACCESSED HAS TO BE PULLED FROM OUR STACK AND GOES THROUGH OUR ACCUMULATOR RATHER THAN BEING SAVED IN OTHER REGISTERS



Our Stack

WITH OUR STACK THE PART IN RED IS WHAT IS EXPECTED TO BE IN THOSE SPOTS.

THE PART IN YELLOW IS FREE TO USE AND UP TO THE USER. THESE CAN BE TREATED AS REGISTERS.

0(sp)	ra - return address
1(sp)	v0 - return value
2(sp)	a0 - argument 0
3(sp)	a1 - argument 1
4(sp)	b0 - branch
5(sp)	whatever you want g0, s0, t1 ...
6(sp)	whatever you want g0, s0, t1 ...
...	...

Types/Formats

N-Type

does not take in any arguments

- jra
- accgetra
- rageacc

I-Type

These types need an immideiate passed through to funciton such as

- stackset 20
- stackget 4

Instruction Set

15 instructions

Mnemonic	Format	Name	OpCode	English Description	Symbolic Description	RTL	Sample Usage	Usage explanation
stackset	I	set stack to	x 00000	set the specified place of stack to the value of acc	{M[imm](11:4)} = acc	PC = PC + 2 inst = Mem[PC] imme = SE(inst[15 : 5]) ALUOut = sp + imme Memory[ALUOut] = acc	stackset 20	set 20(sp) = acc
stackget	I	get value from stack	x 00001	set acc to the value at the specified place on stack	acc = {M[imm](11:4)}	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15 : 5]) ALUOut = sp + imme acc = Memory[ALUOut]	stackget 4	acc = 4(sp)
accseti	I	set acc to imme	x 00010	set acc to a immediate value.	acc = imm	newPC = PC + 2 PC = newPC inst = Mem[PC]	accseti 7	acc = 7
stackadd	I	stackadd	x 00011	acc = acc + ?(sp) acc wil be equal to the sum of acc and the value at the specified place on stack	acc = {M[imm](11:4)} + acc	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15 : 5]) ALUOut = sp + imme ALUOut = Mem[ALUOut] + acc acc = ALUOut last two lines: acc = Mem[ALUOut] +	stackadd 4	acc = acc + 4(sp)
addi	I	add immediate	x 00100	acc = acc + imme acc equals acc plus an immediate	acc = acc +imm	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15:5]) ALUOut = acc + imme acc = ALUOut	addi 3	acc = acc + 3

Instruction Set

stacksub	I	substack	x 00101	acc = acc - ?(sp) acc will be equal to the difference between acc and the value stored at the specified place on the stack	acc = acc - {M[imm](11:4)}	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15 : 5]) ALUOut = sp + imme ALUOut = acc - Mem[ALUOut] acc = ALUOut	stacksub 8	acc = acc - 8(sp)
bne	I	branch if not equal	x 00110	if acc != 8(sp), branch to the label provided (branch will always compare the value of acc and the value stored at 8(sp).)	if(acc != M[8]) PC = PC + {imr	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15:5])	bne while	if(acc != 12(sp)) go to while
beq	I	branch if equal	x 00111	if acc == 8(sp), branch to the label provided (branch will always compare the value of acc and the value stored at 8(sp).)	if(acc == M[8]) PC = PC + {im	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15:5])	beq break	if(acc == 12(sp)) go to break
ble	I	branch if less than or equal	x 01000	if acc <= 8(sp), branch to the label provided (branch will always compare the value of acc and the value stored at 8(sp).)	if(acc < M[8]) PC = PC + {imn	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15:5])		
jal	I	jump and link	x 01001	go to the provided label.	M[0] = PC +4; PC = PC + {imı	newPC = PC + 2 PC = newPC inst = Mem[PC] imme = SE(inst[15:5])	jal while	go to while
spinit	I	initialize stack pointer	x 01010	make space on the stack.	sp = sp - imm	newPC = PC + 2 PC = newPC inst = Mem[PC] ALUOut = sp - imme sp = ALUOut	spinit 24	sp = sp -24

Instruction Set

sprelease	I	release stack	x 01011	reset the stack	sp = sp +imm	newPC = PC + 2 PC = newPC inst = Mem[PC] ALUOut = sp + imme sp = ALUOut	sprelease 12	sp = sp + 12
jra	N	jump ra	x 01100	jump to the address stored in ra	pc = pc + M[0]	newPC = PC + 1 PC = newPC inst = Mem[PC] ALUOut = pc + MEM[0] pc = ALUOut	jra (no argument)	go to the address stored in ra
accgetra	N	acc get ra's value	x 01101	acc equal's ra's value	acc = M[0]	newPC = PC + 1 PC = newPC inst = Mem[PC] acc = ra	accgetra (no argument)	acc = ra
ragetacc	N	ragetacc's value	x01110	ra will be equal to acc's value	M[0] = acc	newPC = PC + 1 PC = newPC inst = Mem[PC] acc = ra	ragetacc (no argument)	ra = acc

Assembler

output.txt - Notepad

File Edit Format View Help

```
01CA
000D
0000
0042
0140
0241
0180
0181
0080
```

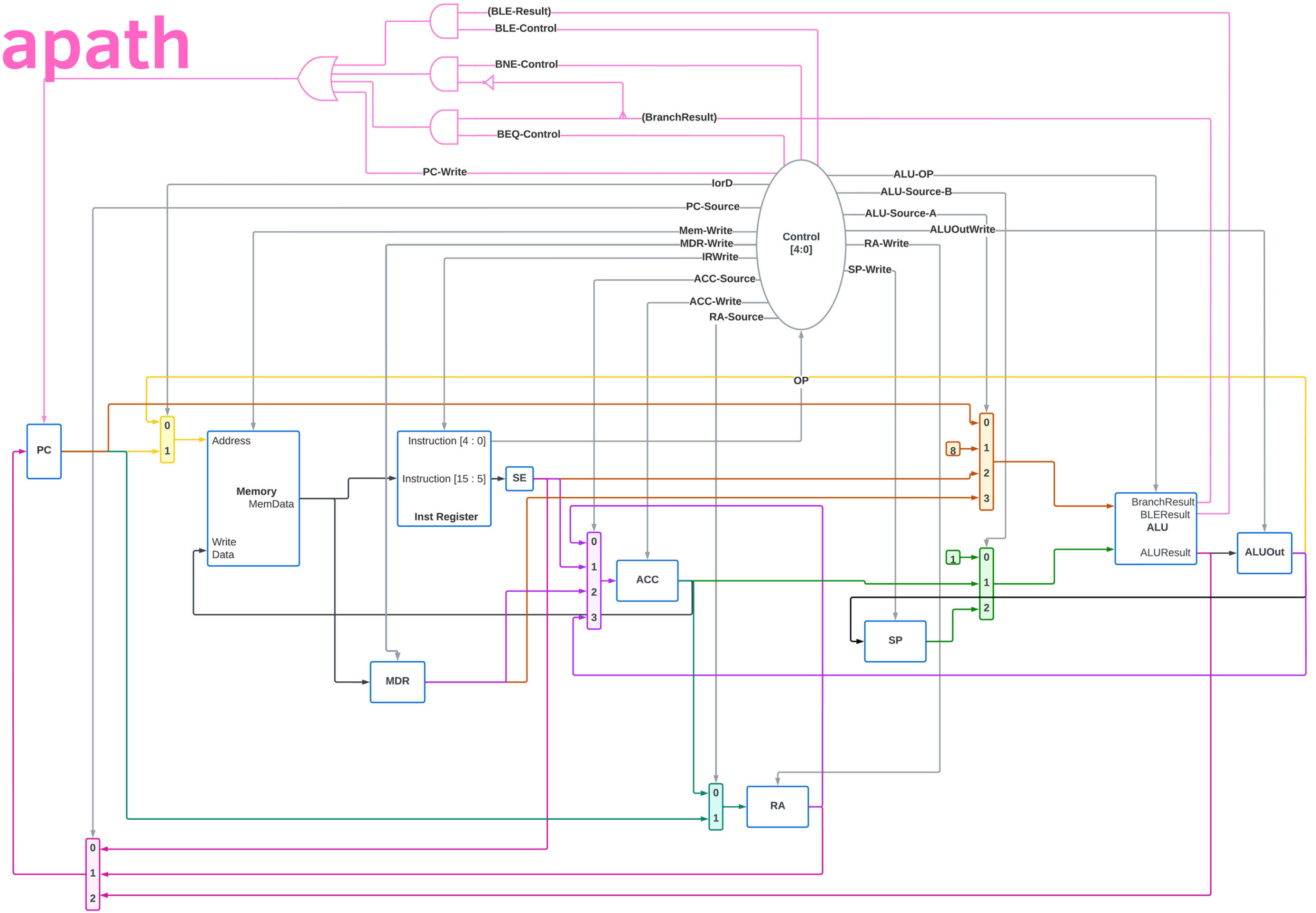
test.txt - Notepad

File Edit Format View H

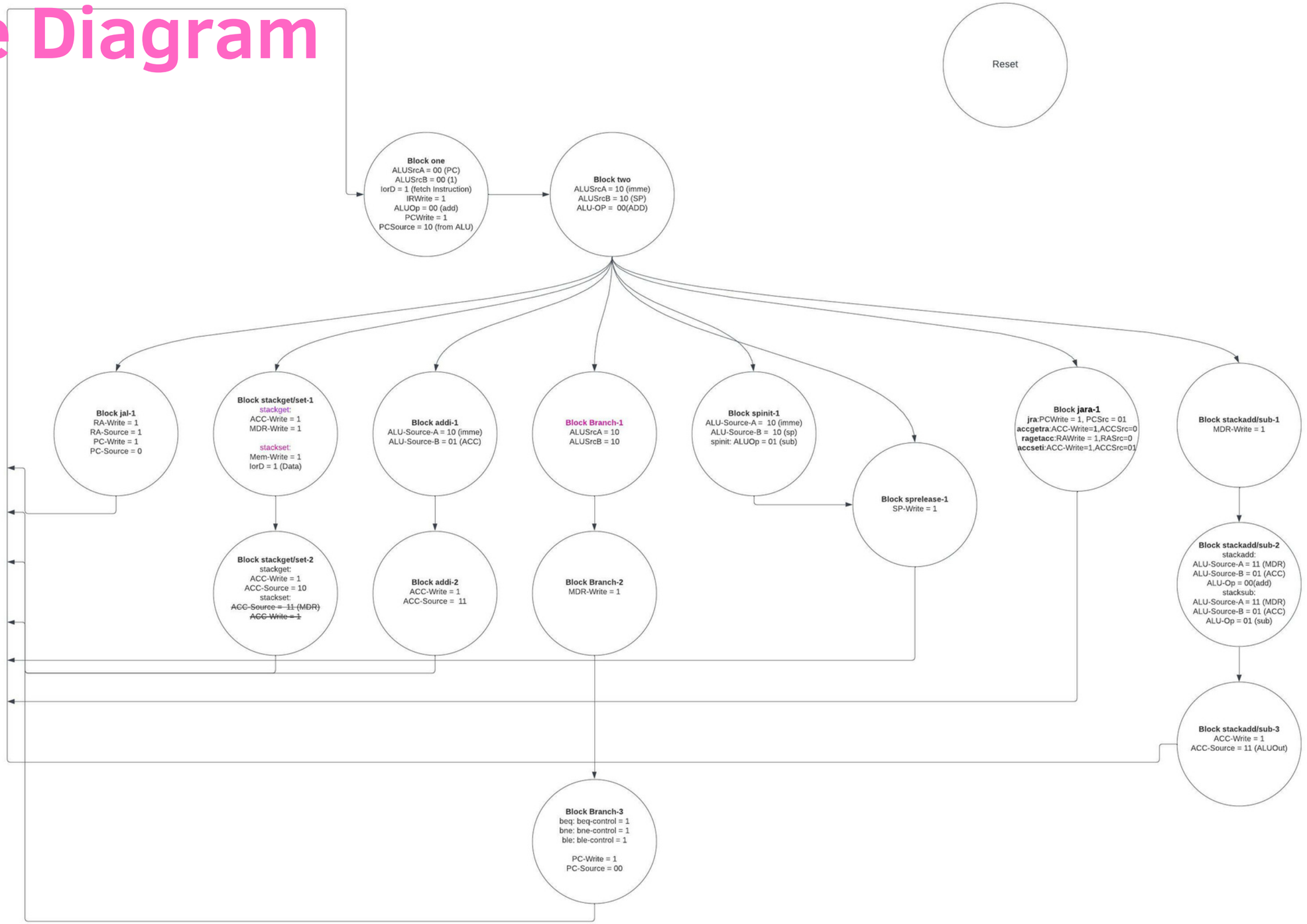
```
spinit 14
accgetra
stackset 0
accseti 2
stackset 10
stackget 18
stackset 12
stackget 12
stackset 4
```

```
def write(self, instructionType, instruction):

    if(instructionType == "I"):

        binarytest = int(instruction["immediate"]+"00000", 2)
        s = str(self.addDecToHex(binarytest, int(instruction["opcodeDec"])))
        print(s)

        s = s.split('x')
        s = s[1]
        s = (s.rjust(4,"0")).upper()


        self.f.write(s +  " \n")

    elif(instructionType == "N"):

        s = hex(instruction["opcodeDec"])
        s = s.split('x')
        s = s[1]
        s = (s.rjust(4,"0")).upper()

        self.f.write(s +  " \n")



def close(self):
    self.f.close()
```

Datapath


(BLE-Result)
BLE-Control
BNE-Control
(BranchResult)
BEQ-Control
PC-Write
IorD
PC-Source
Mem-Write
MDR-Write
IRWrite
ACC-Source
ACC-Write
RA-Source
Control
[4:0]
ALU-OP
ALU-Source-B
ALU-Source-A
ALUOutWrite
RA-Write
SP-Write
OP
PC
Address
Memory
MemData
Write
Data
Instruction [4 : 0]
Instruction [15 : 5]
Inst Register
SE
8
1
ACC
MDR
SP
RA
BranchResult
BLEResult
ALU
ALUResult
ALUOut

State Diagram



RTL

jal	stackset stackget	addi	Branch beq/bne/ble	spinit	sprelease	stackadd stacksub	jra accgetra ragetacc accseti
PC = PC + 1 Inst = Mem[PC]							
Imme = SE(inst[15 : 5]) ALUOut = sp + imme							
ra = PC PC=imme	stackset: Mem[ALUOut]= acc stackget: MDR=Mem[ALU Out]	ALUOut = acc + imme	ALUOut = sp+8	ALUOut=sp - imme	sp = ALUOut	MDR = Mem[ALU Out]	jra: PC=ra accgetra: acc = ra ragetacc: ra = acc accseti: acc=imme
	stackget: Acc = MDR	Acc = ALUOut	MDR = Mem[AL UOut]	sp = ALUOut		stackadd: ALUOut = acc + MDR stacksub: ALUOut = acc - MDR	
			if(acc op MDR) PC = Imme			acc = ALUOut	

Performance

Using 0x13B0 for input in relPrime

Average CPI: 4.012

Total Exe Time: 46,982,950 ns

Instructions Ran: 120, 468

Special Feature

1. Everything on Stack:
    a. Since Unlike a Save/Load type architecture, we only have on register-acc, so we put all the value that need to be stored for later use in stack.
2. Branch Value:
    a. since all our instructions have at most one parameter, we want to make our branch instruction also following this format, so Our branch is like "beq WhileLoop", the two value that will be compared in beq is the value stored in acc and 4(sp)
3. Return Value and Argument:
    a. In order to get the argument from caller or passing the return value to caller, our functions needs to get those two kinds of value "Across Stack"

Approach to Testing

Unit Tests

testing specific units with different inputs to make sure they were outputting what was necessary

For testing multiple units together would only container 3 - 4 units at max that would be given inputs and tested

Thanks!