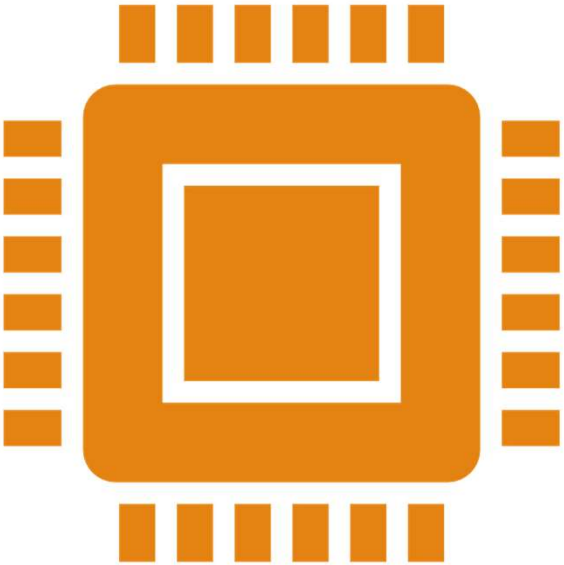
PikaChip Processor

TEAM YELLOW 2324A (DINO NUGGIES)

SPENCER HALSEY, BLAISE SWARTWOOD,

REILLY MOONEY, JACOB SCHEIBE



Overview

- Load Store
- Multicycle
- 16-bit instructions
- Focused on a balance between programmability and processor speed

Instruction Set

Base Integer Instructions

Inst	Name	FMT	Opcode	Description	Note
add	Add	R	0000	R[rd] = R[rs1] + R[rs2]	
sub	Subtract	R	0001	R[rd] = R[rs1] - R[rs2]	
and	And	R	0010	R[rd] = R[rs1] & R[rs2]	
or	Inclusive Or	R	0011	R[rd] = R[rs1] | R[rs2]	
addi	Add Immediate	I	0100	R[rd] = R[rs1] + SE(imm)	
lw	Load Word	I	0111	R[rd] = M[R[rs1] + SE(imm)]	
sw	Store Word	I	1000	M[R[rs1] + SE(imm)] = R[rd]	
beq	Branch ==	U	1001	if (rs1 == BR) PC += SE(imm) << 1	
jal	Jump And Link	U	1100	R[rd] = PC + 2 PC += SE(imm) << 1	PC relative
jalr	Jump And Link Register	I	1101	R[rd] = PC + 2 PC = R[rs1]	register relative
lui	Load Upper Immediate	U	1110	R[rd] = SE(imm) << 8	
lbi	Load Bottom Immediate	U	1111	R[rd] = R[rd] + imm	
si	Shift Immediate	U	0101	if (imm[4] == 0) R[rd] = R[rd] << imm[3:0] else if (imm[4] == 1) R[rd] = R[rd] >> imm[3:0]	
lin	Load Input	U	0110	R[rd] = INPUT	
lout	Load Output	I	1010	OUTPUT = R[rd]	

R = Register file access, SE = Sign extend

Core Instruction Formats

15 12	11 8	7 4	3 0	
rd	rs1	rs2	opcode	Ralts-type (R-type)
rd	rs1	imm	opcode	Ivysaur-type (I-type)
rd	imm		opcode	Umbreon-type (U-type)

RTL (Ralts-Type)

add	sub	and	or
IR <= Mem[PC] PC <= PC + 2			
A <= Reg[IR[11:8]] B <= Reg[IR[7:4]]			
ALUOut <= A + B	ALUOut <= A - B	ALUOut <= A & B	ALUOut <= A | B
Reg[IR[15:12]] = ALUOut			

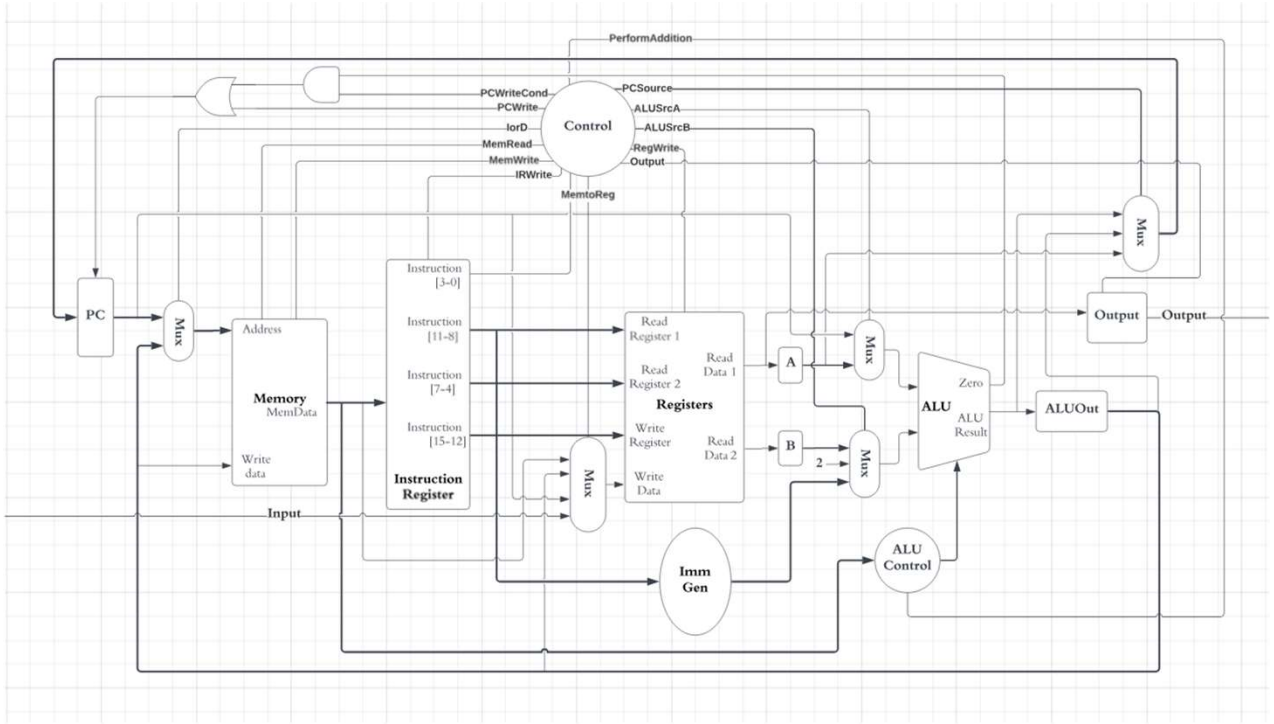
RTL (Ivysaur-Type)

addi	lw	sw	jalr
IR <= Mem[PC] PC <= PC + 2			
A <= Reg[IR[11:8]] B <= Reg[IR[7:4]]			
ALUOut <= A + SE[IR[7:4]]			PC<= A Reg[IR[15:12]] <= PC
Reg[IR[15:12]] <= ALUOut	MDR <= Mem[ALUOut]	Mem[ALUOut] <= B	
	Reg[IR[15:12]] <= MDR		

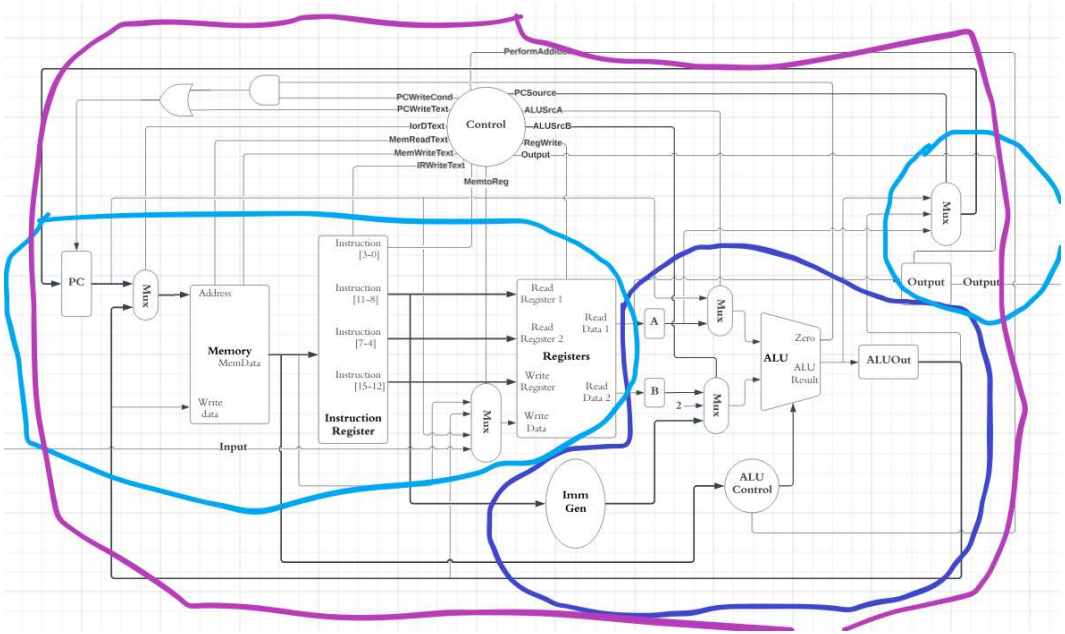
RTL (Umbreon-Type)

beq	lui	lbi	jal	si
IR <= Mem[PC] PC <= PC + 2				
A <= Reg[IR[11:8]] B <= Reg[IR[15:12]]				
A <= Reg[IR[15:12]] B <= Br Imm = SE[IR[11:4]] << 1	Imm = SE[IR[11:4]]	Imm = IR[11:4]	ALUOut <= PC + SE	A <= Reg[IR[15:12]] ALUOut <= A >>|<< Imm[0:3]
Target = PC + Imm	Result = Imm << 8	Result = IR[15:12] + Imm	PC <= ALUOut	Reg[IR[15:12]] <= ALUOut
If (A == B) PC = Target	Reg[IR[15:12]] = Result	Reg[IR[15:12]] = Result	Reg[IR[15:12]] <= PC	

Datapath


PerformAddition
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
PCSource
ALUSrcA
ALUSrcB
RegWrite
Output
PC
Mux
Address
Memory
MemData
Write
data
Input
Instruction
[3-0]
Instruction
[11-8]
Instruction
[7-4]
Instruction
[15-12]
Instruction
Register
Read
Register 1
Read
Register 2
Registers
Write
Register
Write
Data
Read
Data 1
Read
Data 2
A
B
2
Mux
ALU
Zero
ALU
Result
ALUOut
Output
Output
Imm
Gen
ALU
Control

Testing


PerformAddition
PCWriteCond
PCWriteText
IorDText
MemReadText
MemWriteText
IRWriteText
MemtoReg
Control
PCSource
ALUSrcA
ALUSrcB
RegWrite
Output
PC
Mux
Address
Memory
MemData
Write data
Input
Instruction [3-0]
Instruction [11-8]
Instruction [7-4]
Instruction [15-12]
Instruction Register
Read Register 1
Read Register 2
Registers
Write Register
Write Data
Read Data 1
Read Data 2
A
B
2
Zero
ALU
ALU Result
ALUOut
Output
Imm Gen
ALU Control

Unique Design Aspects

Shifting:

- Single shift instruction
- Two extra bits to determine type
- Feed bits from immediate genie to ALU Control

Branching:

- Single branch instruction, beq
- Dedicated register for branch comparison, br
- Templates for other branching

Branching Templates

Bne

Assembly	High-Level

add br, t0, x0
beq t1, Branch
add t0, t0, t1
 	
Branch:
	If (a != b) {
	a = a + b
	}

Beq

Assembly	High-Level

add br, t0, x0
beq t1, Branch
 	
Branch:
add t0, t0, t1
	If (a == b) {
	a = a + b
	}

Bgt

Assembly	High-Level

sub t2, t0, t1
si t2, 31 //imm[4] is 1 to shift right
//imm[3:0] is 1111 to shift 15 bits
add br, x0, t2
addi t2, x0, 1
beq t2, Branch
add br, x0, t0
beq t1, Branch
add t0, t0, t1
 	
Branch:
	If (a > b) {
	a = a + b
	}

Blt

Assembly	High-Level

sub t2, t0, t1
si t2, 31 //imm[4] is 1 to shift right
//imm[3:0] is 1111 to shift 15 bits
add br, x0, t2
addi t2, x0, 0
beq t2, Branch
add br, x0, t0
beq t1, Branch
add t0, t0, t1
 	
Branch:
	If (a < b) {
	a = a + b
	}

Ble

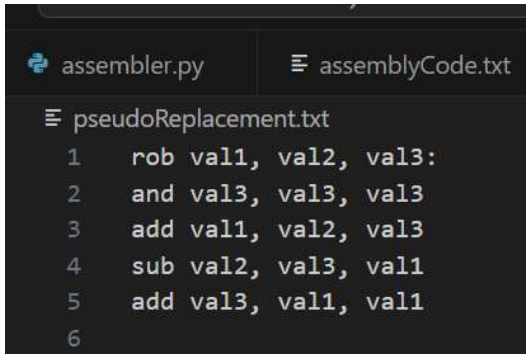
Assembly	High-Level

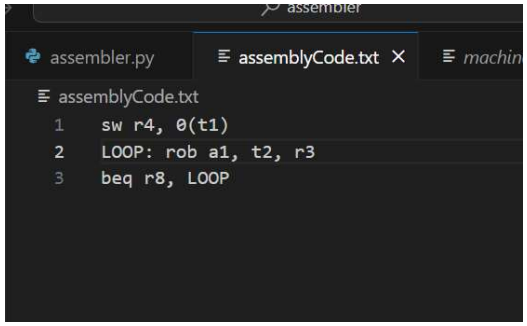
sub t2, t0, t1
si t2, 31 //imm[4] is 1 to shift right
//imm[3:0] is 1111 to shift 15 bits
add br, x0, t2
addi t2, x0, x0
beq t2, Branch
add t0, t0, t1
 	
Branch:
	If (a <= b) {
	a = a + b
	}

Bge

Assembly	High-Level

sub t2, t0, t1
si t2, 31 //imm[4] is 1 to shift right
//imm[3:0] is 1111 to shift 15 bits
add br, x0, t2
addi t2, x0, 1
beq t2, Branch
add t0, t0, t1
 	
Branch:
	If (a >= b) {
	a = a + b
	}





Assembler

- Designed to take regular instructions and pseudo instructions and output the machine code
- Built in Python using Regular Expressions

Assembly Language Code

```
START: Lin a0
add s0, a0, x0
jal ra, relPrime
Lout a0
InfLoop: Lin a1
add br, s0, x0
beq a1, InfLoop
jal x0, START
```

```
relPrime: lui t0, -1
lbi t0, -8
add sp, sp, t0
sw ra, 0(sp)
sw s0, 2 (sp)
sw s1, 4 (sp)
sw s2, 6 (sp)
add s0, a0, x0
addi s1, x0, 2
addi s2, x0, 1
```

```
relLoop: add a0, s0, x0
add a1, s1, x0
jal ra, gcd
add br, s2, x0
beq a0, END
addi s1, s1, 1
jal x0, relLoop
```

Performance Data

Running the default value: 5040

Number of bytes: **110 bytes**

Number of instructions: **112,000 instructions**

Number of cycles: **460,000 cycles**

Average cycles per instruction: **4.09 Average CPI**

Cycle Time: **50.1 ns**

Total Execution Time: **23 Milliseconds**

Logical gates and registers used: **446 Total Registers, 16,384 memory bits**

Design Changes & Challenges

Changes:

- Remove extra slot to allow memory to update after branching/jumping
- Speed up processor

Challenges:

- Clocking
- Lw/Sw
- Few Instruction Types

Questions?