The BAW Instruction Set Manual

Bryant Herren, Wesley Ring, Austin Waddell

Contents

1 Introduction

BAW is the instruction set for a floating point co-processor designed in ECGR 3183 at the University of North Carolina at Charlotte. The co-processor was implemented with a single-cycle architecture, and a pipelined architecture. For the pipelined architecture, two branch prediction algorithms were implemented (1 static and 1 dynamic) as well as no branch prediction.

Provided in this document:

- The complete ISA
- Architecture and controller design (units, diagrams, etc)
- VHDL simulation
- Performance results and discussion for pipelined vs. unpipelined approaches

1.1 System Parameters

- 32 bits per instruction/register
- Data is stored using IEEE single-precision floating point numbers.
- Register file has 16 registers
- Timings:
  - Clock cycle (pipelined): 100ns
  - Register Read/Write: 100ns
  - Memory Read/Write: 300ns
  - Single ALU Op: 200ns

1.2 Memory

The system includes a data memory addressed 0-1023 and 16 Floating Point registers (Referenced as X0-X15). Each memory location and register uses a 32-bit value. The simulation can read an input file containing the operational parameters, code, and memory contents (there is an assembler).

1.3 Additional Features

- FP Multiply by -1, 1, or 0 takes 1 cycle
- FP Multiply by power of 2 takes 2 cycles
- Condition Codes:
  - Z - Zero
  - N - Negative
  - V - Overflow
  - C - Carry
  - E - Error (domain errors, etc.)
- All condition codes are set as needed on arithmetic operations
- The round, ceiling, and floor functions would round up to the nearest integer, expressing the result in floating point format.

2 ISA

2.1 Introduction

The ISA is based off of the ARMv8 / LEGv8 ISA. As a result, there will likely be similarities.

2.2 Instruction Format

There are four instruction formats: Register (R), Data (D), Immediate (I), and Branch (B). All processor instructions are 32 bits wide. Table 1 Contains information about the specific instruction formats.

Notes:

Table 1: Instruction Format Descriptions

Format	Description	Example
R (Register)	An instruction whose inputs and outputs are both registers	ADD X9, X21, X9
D (Data)	An instruction used when fetching or placing data in memory	LOAD X9, [X22, #64]
I (Immediate)	An instruction that carries specific additional data	POW X9, #15
B (Branch)	An instruction that deals with changing the location of the PC directly	PC $\leftarrow$ M[Ri]

1. The Opcode is 8 bits long, allowing it to be easily read in hex format. This also allows additional instructions to be added in the future.
2. The remaining three bits of each instruction format are not used. This allows for the addition of specialized registers (zero register, etc.) if needed.

Please see Table 2 for specific information on the Instruction Formats.

Table 2: Instruction Formats

Instruction Format	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
R (Register)	Opcode (8 bits)								Rm (5 bit)					Empty (7 bit)							Rn (5 bit)					Rd (5 bit)					Empty		
D (Data)	Opcode (8 bits)								Address (9 bit)										op2 (2 bit)		Rn (5 bit)					Rd (5 bit)							
I (Immediate)	Opcode (8 bits)								Immediate Data (11 bit)												Rn (5 bit)					Rd (5 bit)					(3 bit)		
B (Branch)	Opcode (8 bits)								Address (16 bits)																	Rd (5 bit)							

2.3 Instructions

2.3.1 Set

SET

ASM	Opcode	Format	Description	Operation	ALU Cycles
SET	00000001	I	Sets Ri to given floating point value	Ri ← FPvalue	1

ASM Example: SET Ri, #FPvalue

Flags

- Zero
- Negitive

2.3.2 Load

LOD

ASM	Opcode	Format	Description	Operation	ALU Cycles
LOD	00000010	D	Copies Rj from memory and into Ri	Ri ← M[Rj]	1

ASM Example: LOD Ri, Rj

Flags

- Zero
- Negitive

2.3.3 Store

STR

ASM	Opcode	Format	Description	Operation	ALU Cycles
STR	00000011	D	Copies data from register Rj into memory	M[Ri] ← Rj	1

ASM Example: STR Ri, Rj

Flags

- None

2.3.4 Move

MOV

ASM	Opcode	Format	Description	Operation	ALU Cycles
MOV	00000100	R	Moves the value of Rj to Ri, deleting the original	Ri ← Rj	1

ASM Example: MOV Ri, Rj

Flags

- Zero
- Negitive

2.3.5 Add

ADD

ASM	Opcode	Format	Description	Operation	ALU Cycles
ADD	00000101	R	Adds Rj and Rk into Ri	Ri ← Rj + Rk	3

ASM Example: ADD Ri, Rj, Rk

Flags

- Zero
- Negitive
- Overflow
- Carry

2.3.6 Subtract

SUB

ASM	Opcode	Format	Description	Operation	ALU Cycles
SUB	00000110	R	Subtrcts Rk from Rj into Ri	Ri ← Rj – Rk	3

ASM Example: SUB Ri, Rj, Rk

Flags

- Zero
- Negitive
- Overflow
- Carry

2.3.7 Negate

NEG

ASM	Opcode	Format	Description	Operation	ALU Cycles
NEG	00000111	R	Sets Ri to the Opposite of Rj	Ri ← -Rj	1

ASM Example: NEG Ri, Rj

Flags
- Zero
- Negitive

2.3.8 Multiply

MUL

ASM	Opcode	Format	Description	Operation	ALU Cycles
MUL	00001000	R	Multiplies Rj and Rk into Ri	Ri ← Rj * Rk	5

ASM Example: MUL Ri, Rj, Rk

Flags
- Zero
- Negitive
- Overflow
- Carry

2.3.9 Divide

DIV

ASM	Opcode	Format	Description	Operation	ALU Cycles
DIV	00001001	R	Divides Rj by Rk into Ri	Ri ← Rj / Rk	8

ASM Example: DIV Ri, Rj, Rk

Flags
- Zero
- Negitive
- Overflow
- Carry
- Error - divide by zero

2.3.10 Floor

FLR

ASM	Opcode	Format	Description	Operation	ALU Cycles
FLR	00001010	R	Sets Ri to the floor of Rj	Ri ← ⌊ Rj⌋	1

ASM Example: FLR Ri, Rj

Flags

- Zero
- Negitive

2.3.11 Ceiling

CEL

ASM	Opcode	Format	Description	Operation	ALU Cycles
CEL	00001011	R	Sets Ri to the ceil of Rj	Ri ← ⌈ Rj⌉	1

ASM Example: CEL Ri, Rj

Flags

- Zero
- Negitive

2.3.12 Round

RND

ASM	Opcode	Format	Description	Operation	ALU Cycles
RND	00001100	R	Sets Ri to Rj rounded to the nearest integer	Ri ← round(Rj)	1

ASM Example: RND Ri, Rj

Flags

- Zero
- Negitive

2.3.13 Absolute Value

ABS

ASM	Opcode	Format	Description	Operation	ALU Cycles
ABS	00001101	R	Sets Ri to the absolute value of Rj	Ri ← — Rj —	1

ASM Example: ABS Ri, Rj

Flags
- Zero

2.3.14 Minimum

MIN

ASM	Opcode	Format	Description	Operation	ALU Cycles
MIN	00001110	R	Sets Ri to the minimum value between Rj and Rk	Ri ← min(Rj, Rk)	1

ASM Example: MIN Ri, Rj, Rk

Flags
- Zero
- Negitive

2.3.15 Maximum

MAX

ASM	Opcode	Format	Description	Operation	ALU Cycles
MAX	00001111	R	Sets Ri to the maximum value between Rj and Rk	Ri ← max(Rj, Rk)	1

ASM Example: MAX Ri, Rj, Rk

Flags
- Zero
- Negitive

2.3.16 Power

POW

ASM	Opcode	Format	Description	Operation	ALU Cycles
POW	00010000	I	Sets Ri to Rj raised to some given integer power	Ri ← Rjˆinteger_value	6

ASM Example: POW Ri, Rj, #integer_value

Flags
- Zero
- Negitive
- Overflow
- Carry

2.3.17 Exponent

EXP

ASM	Opcode	Format	Description	Operation	ALU Cycles
EXP	00010001	R	Sets Ri to Rj exponentiated	Ri ← eˆRj	8

ASM Example: EXP Ri, Rj

Flags
- Overflow
- Carry

2.3.18 Square Root

SQR

ASM	Opcode	Format	Description	Operation	ALU Cycles
SQR	00010010	R	Sets Ri to the square root of Rj	Ri ← $\sqrt{}$ Rj	8

ASM Example: SQR Ri, Rj

Flags
- Zero
- Overflow
- Carry
- Error - complex domain

2.3.19 Branch (Uncond.)

BRU

ASM	Opcode	Format	Description	Operation	ALU Cycles
BRU	00010011	B	Loads Ri from memory into PC	PC ← M[Ri]	1

ASM Example: BRU Ri

Flags

- None

2.3.20 Branch Zero

BRZ

ASM	Opcode	Format	Description	Operation	ALU Cycles
BRZ	00010100	B	Sends the PC to a specific labeled line if Ri is zero	If (Ri == 0) PC ← LABEL (line)	3

ASM Example: BRZ Ri, LABEL

Flags

- Zero

2.3.21 Branch Negative

BRN

ASM	Opcode	Format	Description	Operation	ALU Cycles
BRN	00010101	B	Sends the PC to a specific labeled line if Ri is negative	If (Ri ¡ 0) PC ← LABEL (line)	3

ASM Example: BRN Ri, LABEL

Flags

- Negitive

2.3.22 No-op

NOP

ASM	Opcode	Format	Description	Operation	ALU Cycles
NOP	00010110	-	No operation	No operation	1

ASM Example: NOP

Flags

- None

2.3.23 Halt

HLT

ASM	Opcode	Format	Description	Operation	ALU Cycles
HLT	00010111	-	Stop program	Stop Program	-

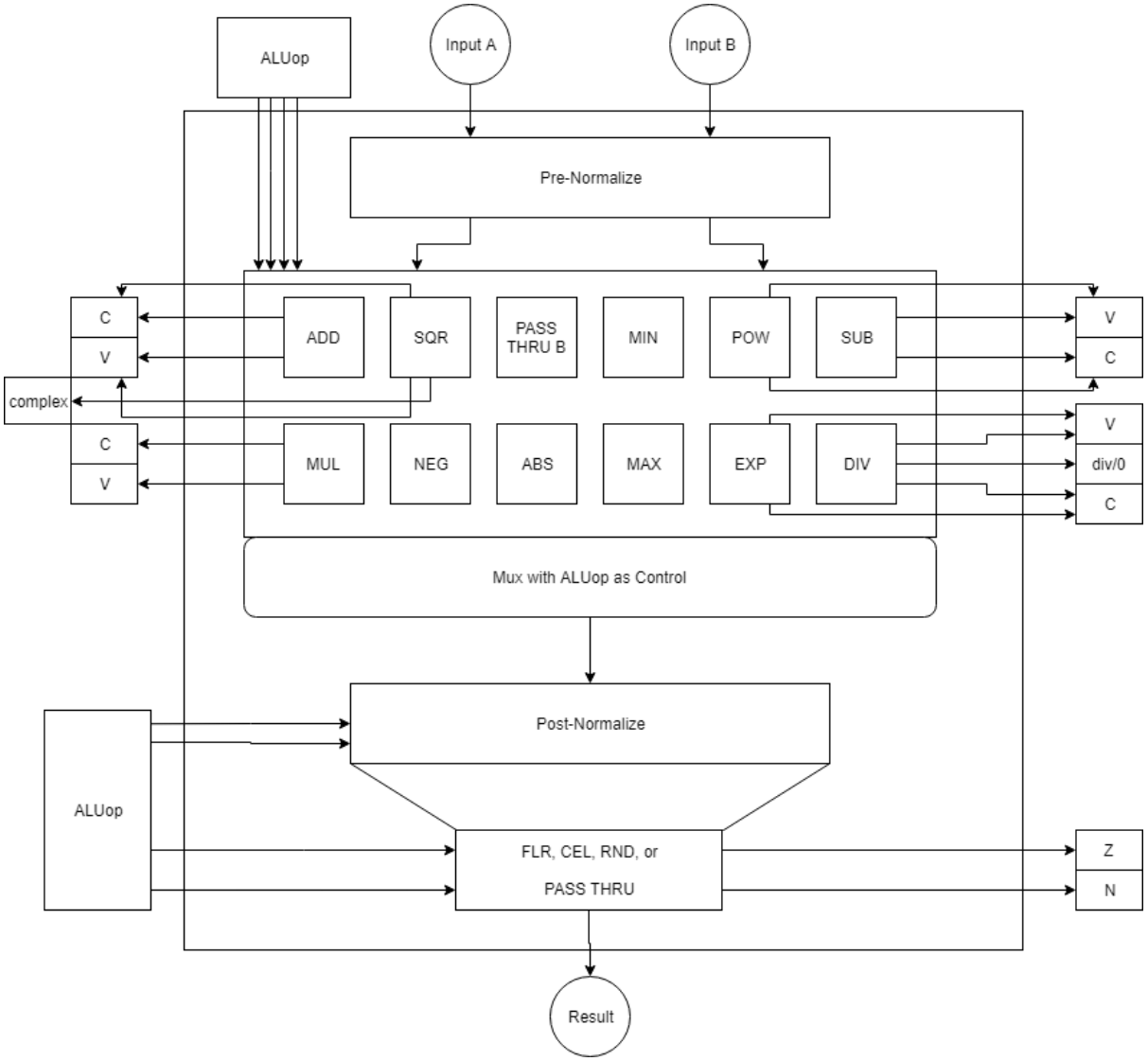
ASM Example: HLT

Flags

- None

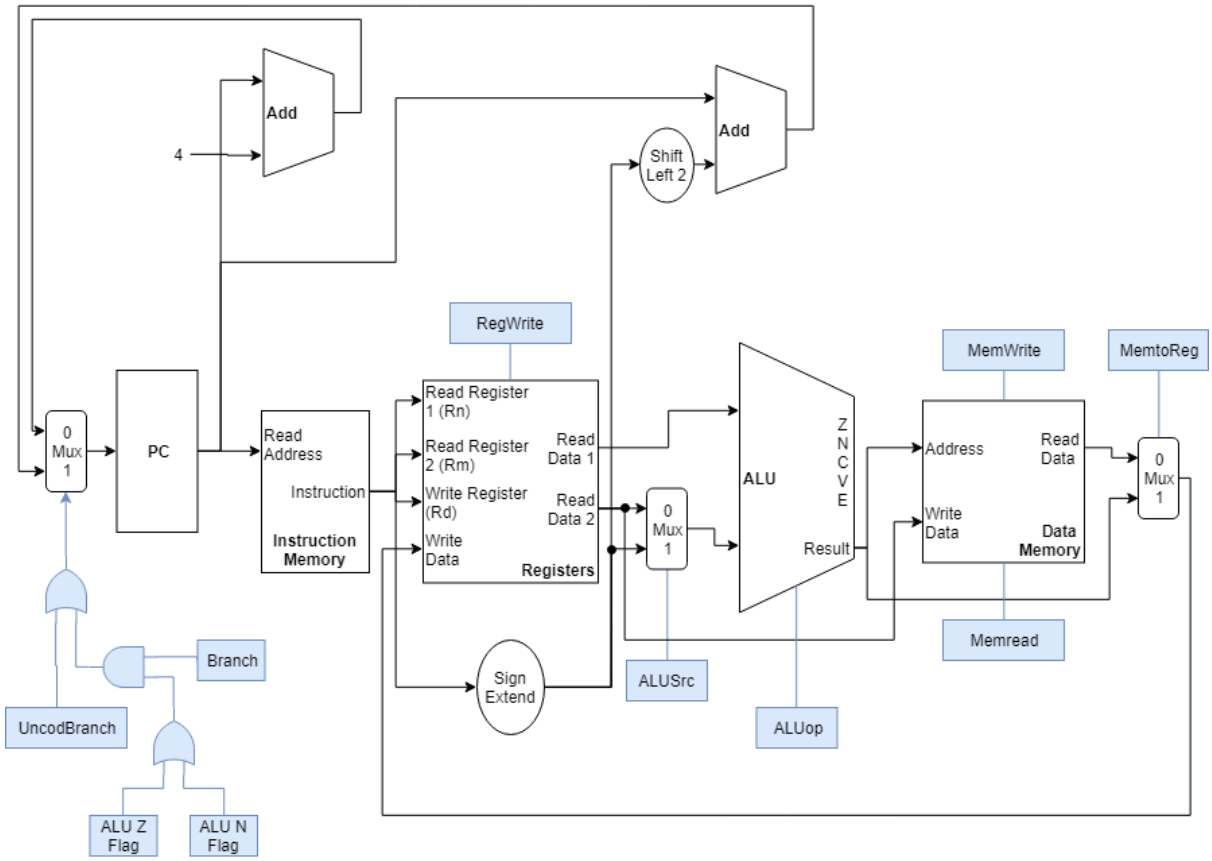
3 Architecture

3.1 ALU



3.2 Datapath

3.2.1 Single Cycle



3.2.2 Pipelined

3.3 controller

4 VHDL Description

5 Testing

6 Conclusion

Datapath Control (We'll document what each of the signals means later, but its the same as the book)

ASM	Opcode	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	UncodBranch
SET	00000001	1	0	1	0	0	0	0
LOD	00000010	1	1	1	1	0	0	0
STR	00000011	1	X	0	0	1	0	0
MOV	00000100	0	0	1	0	0	0	0
ADD	00000101	0	0	1	0	0	0	0
SUB	00000110	0	0	1	0	0	0	0
NEG	00000111	0	0	1	0	0	0	0
MUL	00001000	0	0	1	0	0	0	0
DIV	00001001	0	0	1	0	0	0	0
FLR	00001010	0	0	1	0	0	0	0
CEL	00001011	0	0	1	0	0	0	0
RND	00001100	0	0	1	0	0	0	0
ABS	00001101	0	0	1	0	0	0	0
MIN	00001110	0	0	1	0	0	0	0
MAX	00001111	0	0	1	0	0	0	0
POW	00010000	0	0	1	0	0	0	0
EXP	00010001	0	0	1	0	0	0	0
SQR	00010010	0	0	1	0	0	0	0
BRU	00010011	1	X	X	X	X	X	1
BRZ	00010100	1	X	0	0	0	1	0
BRN	00010101	1	X	0	0	0	1	0
NOP	00010110	0	0	0	0	0	0	0
HLT	00010111	0	0	0	0	0	0	0

ALU Control (ALUop)

ASM	Opcode	ALUop	Description
SET	00000001	0000	Pass Through
LOD	00000010	0000	Pass Through
STR	00000011	0000	Pass Through
MOV	00000100	0000	Pass Through
ADD	00000101	0001	Use Adder
SUB	00000110	0010	Use Subtracter
NEG	00000111	0011	Negate
MUL	00001000	0100	Use Multiplier
DIV	00001001	0101	Use Divider
FLR	00001010	0110	Floor Result
CEL	00001011	0111	Ceil Result
RND	00001100	1000	Round Result
ABS	00001101	1001	Take Absolute Value
MIN	00001110	1010	Take Minimum Input
MAX	00001111	1011	Take Maximum Input
POW	00010000	1100	Take Power
EXP	00010001	1101	Exponentiate
SQR	00010010	1110	Take Square Root
BRU	00010011	0000	Pass Through
BRZ	00010100	0000	Pass Through
BRN	00010101	0000	Pass Through
NOP	00010110	0000	Pass Through
HLT	00010111	0000	Pass Through