16-bit Softcore CPU Design

General Architecture Specifications

- Word addressable system
- 16-bit data
- 16-bit instructions
    - Opcodes are 5 bits, allowing up to 32 instructions.
    - Addresses are 11 bits, allowing up to 2K addressable locations (4KB memory).
- 8 Internal general-purpose registers, addressed with 3 bits.

Internal Registers

R0	General Purpose
R1	General Purpose
R2	General Purpose
R3	General Purpose
R4	Frame Pointer
R5	Stack Pointer
R6	Link Register
R7	Memory Buffer Register

Special Purpose Registers:

PC	Program Counter
IR	Instruction Register
MAR	Memory Address Register
ACC	Accumulator
FLAGS	Flags Register

Arithmetic and Logic Unit I/O

The ALU is a combinational module, it is always working, hence the output must be stopped so that it does not always interfere with the Data Bus. For this reason, the **Accumulator** register acts as a buffer between the ALU output and the Data Bus. When a value needs to be saved, the Accumulator can be enabled into the bus.

The inputs to the ALU do not interfere with trying to overwrite the Bus. However, there must be two unique inputs. For this reason, this architecture uses the Data Bus directly as one of the inputs, and another register (the **Memory Buffer Register MBR**) as a temporary register to hold the second operand.

Figure 1 illustrates the general input and output connections in the ALU:

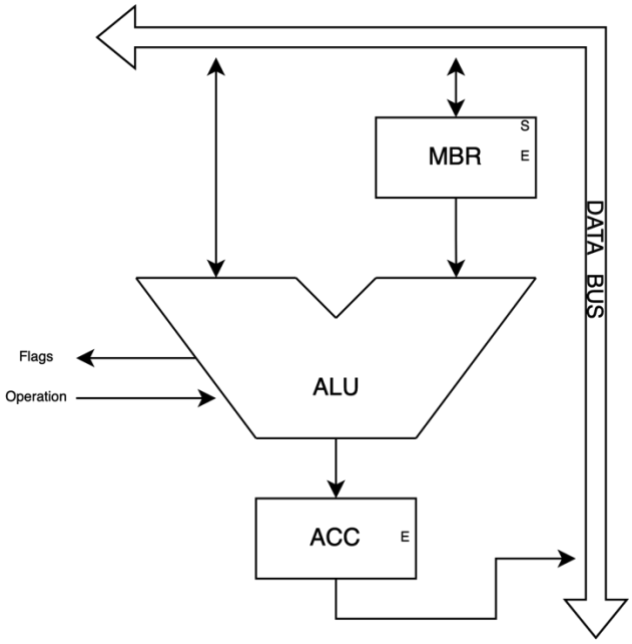

Figure 1: General ALU Connections

Instruction Set Architecture Draft

Each one of the following instructions has a unique Opcode, even though some may have the same name and just different operands.

The instructions are classified into five different categories, and each category determines how the bits in the instruction are used. Figure 2 explains this further.

Instruction Type	Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T1	Opcode					Reg Pos 0			Constant							
T2	Opcode					Reg Pos 0			Reg Pos 1			f1				
T3	Opcode					f2										
T4	Opcode					Reg Pos 0			Reg Pos 1			Reg Pos 2			Unused	
T5	Opcode					Reg Pos 0			Reg Pos 1			Constant				Unused

f1 can be one of:	f2 can be one of:
Unused	Unused
Offset	Label
Constant	

Figure 2: Instruction Types

The following is a summary of the current instructions in the architecture.

1. MOV {reg_dst} #{constant}

Move operation: Moves an immediate constant 8-bit number (**constant**) into a destination register **reg_dst**

Opcode	Destination Register	Constant
5 bits	3 bits	8 bits

Opcode Formats:

Name	Decimal	Hex	Binary
MOV	0	0x00	0b00000

2. MOV {reg_dst} {reg_src}

Move operation: Moves a source register (**reg_src**) into a destination register **reg_dst**

Opcode	Destination Register	Source Register	Unused
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
MOV	1	0x01	0b00001

Memory Operations:

3. LDA {label}

Load Address operation: Loads memory address **label** into destination register **MBR**.
This LDA load operation can be seen as Direct Addressing, whereas LDR can be seen as Indirect.

Opcode	Memory Address (label)
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
LOAD	2	0x02	0b00010

4. LDR {reg_dst} &{reg_adr} #{offset}

Load Register operation: Loads the contents at the memory location specified by the address at register **reg_adr** into destination register **reg_dst**. A 5-bit positive or negative offset specified by **offset** may be applied to the address at **reg_adr**. Being 5-bit and signed, the offset may go from -16 to 15, where each unit represents a word (An offset of -3 means three memory locations lower than the address).

Opcode	Destination Register	Address Register	Offset
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
LOAD	3	0x03	0b00011

5. STRA {label}

Store Address operation: Stores the contents of register **MBR** into the memory location at the address specified by **label**.

Opcode	Memory Address (label)
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
STR	4	0x04	0b00100

6. STRR {reg_src} &{reg_adr} #{offset}

Store Register operation: Stores the contents of register **reg_src** into the memory location specified by the address at register **reg_adr**. A 5-bit positive or negative offset specified by **offset** may be applied to the address at **reg_adr**. Being 5-bit and signed, the offset may go from -16 to 15, where each unit represents a word (An offset of -3 means three memory locations lower than the address).

Opcode	Source Register	Address Register	Offset
5 bits	3 bits	3 bits	5 bits

Example usage:

```
// Store contents of R1 in address specified by R3 offseted by 2 locations.
STR R1, &R3, #2
```

Opcode Formats:

Name	Decimal	Hex	Binary
STR	5	0x05	0b00101

7. PUSH {reg_a} {reg_b} {reg_c}

Push operation: Pushes up to 3 different registers into the stack. Only the registers passed as operands will be pushed.

The 2 least-significant bits are used at know how many registers to push. This is determined by syntax at assembly time.

Opcode	Register A	Register B (Optional)	Register C (Optional)	Number of Registers
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
PUSH	6	0x06	0b00110

8. POP {reg_a} {reg_b} {reg_c}

Push operation: Pops up to 3 different registers from the stack and into the register operands.
The 2 least-significant bits are used at know how many registers to pop. This is determined by syntax at assembly time.

Opcode	Register A	Register B (Optional)	Register C (Optional)	Number of Registers
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
POP	7	0x07	0b00111

Arithmetic and Logic Operations

9. ADD {reg_dst} {reg_a} #{constant} - Instruction Type 2 (f1: Constant)

Add operation: Adds the contents of **reg_a** to a 5-bit constant **constant** and places the result into **reg_dst**

Opcode	Destination Register	Register A	Constant
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
ADD	8	0x08	0b01000

10. ADD {reg_dst} {reg_a} {reg_b} - Instruction Type 4

Add operation: Adds the contents of **reg_a** and **reg_b** and places the result into **reg_dst**

Opcode	Destination Register	Register A	Register B	Unused
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
ADD	9	0x09	0b01001

11. SUB {reg_dst} {reg_a} #{constant}

Subtract operation: Subtracts a 5-bit constant **constant** from the contents of **reg_a** and places the result into **reg_dst**

Opcode	Destination Register	Register A	Constant
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
SUB	10	0x0A	0b01010

12. SUB {reg_dst} {reg_a} {reg_b} - Instruction Type 4

Subtract operation: Subtracts the contents of **reg_**b from **reg_a** and places the result into **reg_dst**

Opcode	Destination Register	Register A	Register B	Unused
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
SUB	11	0x0B	0b01011

13. SHL {reg_dst} {reg_src} #{constant}

Logical Shift Left operation: Shifts **constant** number of bits left to the contents of **reg_src** and places the result into **reg_dst**.

Opcode	Destination Register	Source Register	Constant	Unused
5 bits	3 bits	3 bits	4 bits	1 bit

Opcode Formats:

Name	Decimal	Hex	Binary
SHL	12	0x0C	0b01100

14. SHR {reg_dst} {reg_src} #{constant}

Logical Shift Right operation: Shifts **constant** number of bits right to the contents of **reg_src** and places the result into **reg_dst**.

Opcode	Destination Register	Source Register	Constant	Unused
5 bits	3 bits	3 bits	4 bits	1 bit

Opcode Formats:

Name	Decimal	Hex	Binary
SHR	13	0x0D	0b01101

15. AND {reg_dst} {reg_a} {reg_b} - Instruction Type 4

Bitwise And Operation: Bitwise ANDs the contents of **reg_a** and **reg_b** and stores the result in **reg_dst**.

Opcode	Destination Register	Register A	Register B	Unused
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
AND	14	0x0E	0b01110

16. OR {reg_dst} {reg_a} {reg_b} - Instruction Type 4

Bitwise Or Operation: Bitwise ORs the contents of **reg_a** and **reg_b** and stores the result in **reg_dst**.

Opcode	Destination Register	Register A	Register B	Unused
5 bits	3 bits	3 bits	3 bits	2 bits

Opcode Formats:

Name	Decimal	Hex	Binary
OR	15	0x0F	0b01111

17. NOT {reg_dst} {reg_src}

Bitwise Not Operation: Bitwise Negates the contents of **reg_src** and stores the result in **reg_dst**.

Opcode	Destination Register	Source Register	Unused
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
NOT	16	0x10	0b10000

Control Flow:

18. JMP {label} - Instruction Type 3

Branch Always Operation: Unconditionally jumps to **label**.

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
JMP	17	0x11	0b10001

19. BLN {label} - Instruction Type 3

Branch with Link Operation: Jumps to **label** and stores the current value of the PC into the Link Register **LR**.

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BLN	18	0x12	0b10010

20. RET - Instruction Type 3

Return Operation: Jumps to the address in the Link Register **LR**.

Opcode	Unused
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
RET	19	0x13	0b10011

21. CMP {reg_a} #{constant} - Instruction Type 1

Compare Operation: Compares the contents of **reg_a** to the immediate value **constant** and sets the appropiate flags.

Opcode	Register A	Constant
5 bits	3 bits	8 bits

Opcode Formats:

Name	Decimal	Hex	Binary
CMP	20	0x14	0b10100

22. CMP {reg_a} {reg_b} - Instruction Type 2

Compare Operation: Compares the contents of **reg_a** to the contents of **reg_b** and sets the appropiate flags.

Opcode	Register A	Register B	Unused
5 bits	3 bits	3 bits	5 bits

Opcode Formats:

Name	Decimal	Hex	Binary
CMP	21	0x15	0b10101

23. BEQ {label} - Instruction Type 3

Branch if Equal: Jumps to **label** if the **Zero Flag ZR** from the ALU is set.

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BEQ	22	0x16	0b10110

Example Usage:
```
// Branch to some_label if R0 is equal to zero
CMP R0 #0
BEQ some_label
```

24. BNE {label}
Branch if Not Equal: Jumps to **label** if the **Zero Flag ZR** from the ALU is not set.

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BNE	23	0x17	0b10111

25. BGT {label}
Branch if Greater Than (signed): Jumps to **label** if the **Negative Flag NG** is equal to the **Overflow Flag OV**.

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BGT	24	0x18	0b11000

Example Usage:
```
// Branch if signed number from r0 is greater than r1
MOV R0 #120
MOV R1 #-122
CMP R0 R1
BGT some_label
```

26. BGTU {label}
Branch if Greater Than (unsigned): Jumps to **label** if the **Carry Flag CA** is set the **Zero Flag ZR** is reset (CA == 1 && ZR == 0).

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BGTU	25	0x19	0b11001

Example Usage:
```
// Branch if unsigned number from r0 is greater than r1
MOV R0 #133
MOV R1 #4
```

```
CMP R0 R1
BGTU some_label
```

27. BLT {label}

Branch if Less Than (signed): Jumps to **label** if the **Negative Flag NG** is not equal to the **Overflow Flag OV** (NG != OV).

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BLT	26	0x1A	0b11010

Example Usage:

```
// Branch if signed number from r0 is less than r1
MOV R0 #-123
MOV R1 #4
CMP R0 R1
BLT some_label
```

28. BLTU {label}

Branch if Less Than (unsigned): Jumps to **label** if the **Carry Flag CA** is reset (CA == 0).

Opcode	label
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
BLTU	27	0x1B	0b11011

Example Usage:

```
// Branch if unsigned number from r0 is less than r1
MOV R0 #120
MOV R1 #134
CMP R0 R1
BLTU some_label
```

29. HALT

Halts program execution. Cannot exit this state until a system restart through the physical reset button.

Opcode	Unused
5 bits	11 bits

Opcode Formats:

Name	Decimal	Hex	Binary
RET	28	0x1C	0b11100