Instruction Set and ALU Operation Documentation

Section 1: Instruction Set

Table 1 lists the microcontroller's complete instruction set.

Types of operands:

- General-purpose register addresses: 'rA', or 'rB'.
    - Placeholders to represent any address to a general-purpose register. In an actual instruction, these would be replaced by actual general-purpose register addresses, such as 'r1' or 'r7.'
- Number literals: 'NUM'.
    - Placeholders to represent any number literal, or a numerical value included directly in the program itself. For example, if a program needed to initialize a certain register to the value 0x34, then a user would include 0x34 as a number literal in an instruction such as 'MOV r3, 0x34'.

An 'ALIAS' refers to another name for an instruction, often used to increase readability and clarity.

Table 1

List of instructions

Instruction	OPCODE	FLAG	Type Description
ADD	00000	00	ADD rA, rB • Add rA and rB and put result in rA. • If result is zero, set zero bit.
		01	ADD rA, NUM • Add NUM to rA and put result in rA.
SUB	00001	00	SUB rA, rB • Subtract rA and rB and put result in rA.
		01	SUB rA, NUM • Subract NUM from rA and put result in rA.

NEG	00010	00	NEG rA • Place the 2's complement of rA into rA.
INC	00011	00	INC rA • Increment rA by 1 and store the result in rA.
DEC	00100	00	DEC rA • Decrement rA by 1 and store the result in rA.
AND	00101	00	AND rA, rB • Bitwise AND rA and rB and store the result in rA.
		01	AND rA, NUM • Bitwise AND rA and NUM and store the result in rA.
OR	00110	00	OR rA, rB • Bitwise OR rA and rB and store the result in rA.
		01	OR rA, NUM • Bitwise OR rA and NUM and store the result in rA.
XOR	00111	00	XOR rA, rB • Bitwise XOR rA and rB and store the result in rA.
		01	XOR rA, NUM • Bitwise XOR rA and NUM and store the result in rA.
NOT	01000	00	NOT rA • Bitwise NOT (1's complement) rA and store the result in rA.
LSL	01001	00	LSL rA, rB • Logical shift rA to the left rB times and store the result in rA.
		01	LSL rA, NUM • Logical shift rA to the left NUM times and store the result in rA.
LSR	01010	00	LSR rA, rB • Logical shift rA to the right rB times and store the result in rA.
		01	LSR rA, NUM • Logical shift rA to the right NUM times and store the result in rA.
ASL	-------	00	ALIAS to LSL rA, rB
		01	ALIAS to LSL rA, NUM

ASR	01011	00	ASR rA, rB • Arithmetic shift rA to the right rB times and store the result in rA.
		01	ASR rA, NUM • Arithmetic shift rA to the right NUM times and store the result in rA.
CMP	01100	00	CMP rA, rB • Equivalent to SUB but *only* sets the flags and *does not* put result into rA.
		01	CMP rA, NUM • Equivalent to SUB but *only* sets the flags and *does not* put result in to rA
AJMP	01101	00	AJMP rA • Sets the program counter to the value stored in rA.
		01	AJMP NUM • Sets the program counter to the value NUM.
RJMP (*same opcode as AJMP*)	01101	10	RJMP rA • Adds the value of rA to the program counter.
		11	RJMP NUM • Adds NUM to the program counter.
LDI	01110	00	LDI rA, rB • Loads the value located at the memory address stored in rB into rA.
		01	LDI rA, NUM • Loads the value located at the memory address NUM into rA.
STO	01111	00	STO rA, rB • Writes the value of rA at the memory address stored in rB.
		01	STO rA, NUM • Writes the value of rA at the memory address NUM.
RET	11110	00	RET • *TODO: Determine stack structure and behavior*.
HALT	11111	00	HALT • Stops the microcontroller's execution.

Section 2: ALU Selector Codes and Operations

Table 2 contains a list of all the ALU's operations. At any given moment, the ALU sets the zero flag if the output is zero, and the negative flag if the output is negative (i.e., the most significant bit is '1'). The carry and overflow flags are set according to the specification listed in the description of certain operations. A and B are 16-bit word inputs, and Y is a 16-bit word output.

Table 2

ALU Selector Codes and Operations

Instruction	**SELECTOR Value**	**Description**
Add	0000	Sets Y to A + B. Sets the carry flag if a bit was carried past the most significant bit. Sets the overflow flag if A and B were the same sign, but Y and A are not.
Subtract	0001	Sets Y to A + (-B), where -B represents the two's complement of B (see below). Sets the carry flag if a bit was carried past the most significant bit. Sets the overflow flag if A and B were different signs and Y has a different sign from A.
Negate (2's complement)	0010	Sets Y to the two's complement of A, which is equivalent to `(not A) + 1`. Example: 000101 becomes 111011. When A and Y represent signed numbers, where the most significant bit represents the sign of the number, this effectively negates the number.
Increment	0011	Sets Y to to A + 1.
Decrement	0100	Sets Y to A + (-1).
Pass Thru	0101	Sets Y to A. This is useful for setting flags based on the value of A without calculating a new value.
Bitwise AND	0110	Sets Y to bitwise A AND B. Each bit in Y is '1' if *both* A *and* B have a '1' in that position, and '0' if not.
Bitwise OR	0111	Sets Y to bitwise A OR B. Each bit in Y is '1' if either A or B has a '1' in that position, and '0' if not.
Bitwise XOR	1000	Sets Y to bitwise A XOR B. Each bit in Y is '1' if *exactly one* of A and B contain a '1' in that position, and '0' if not.

Bitwise NOT (1's complement)	1001	Sets Y to NOT A. Each bit in Y is '1' if that bit in A is '0', and each bit in Y is '0' if that bit in A is '1'. In essence – switches all zero bits to ones and vice versa.
Logical Shift Left	1010	Sets Y to A shifted left by B places. Fills in the right side with '0's.
Logical Shift Right	1011	Sets Y to A logically shifted right by B places. Fills in the left side with '0's.
Arithmetic Shift Right	1100	Sets Y to A arithmetically shifted right by B places. Fills in the left side with the most significant bit of A (which preserves A's sign if it's a signed integer).