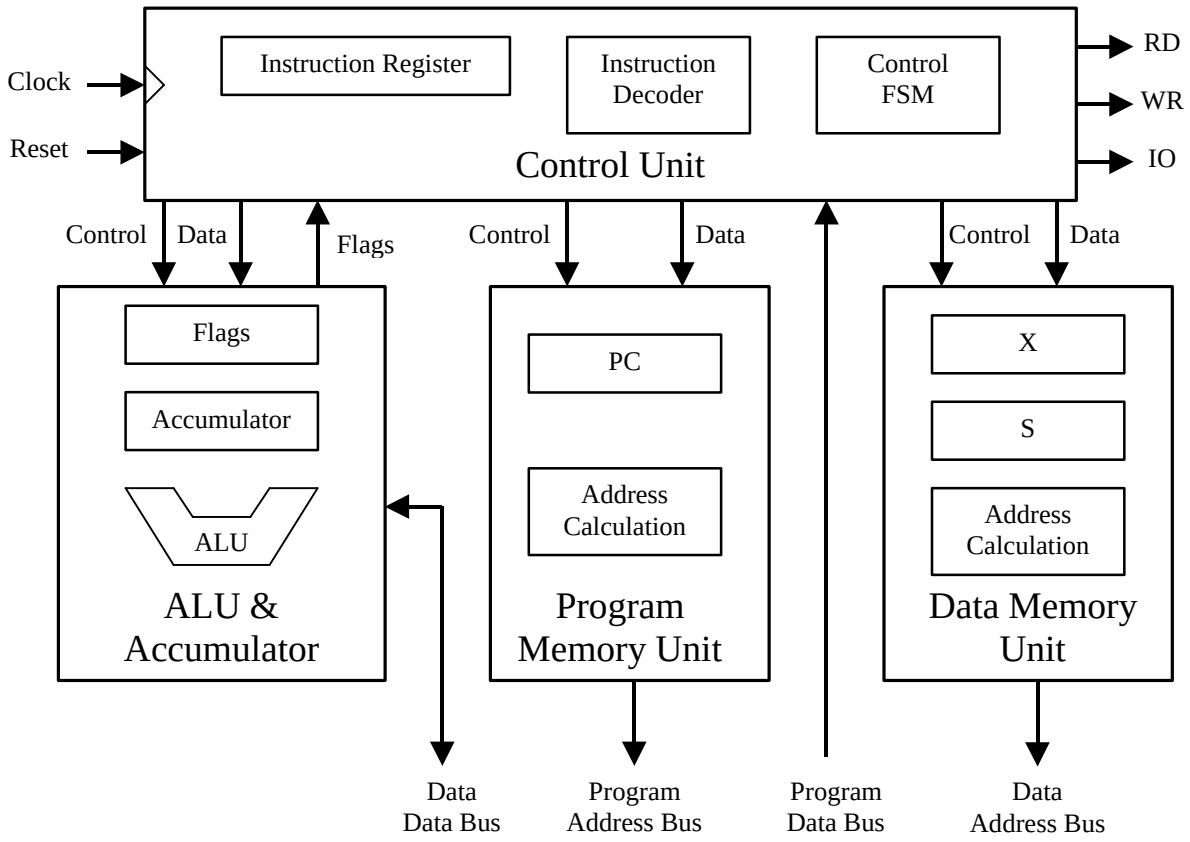
Caltech10 CPU

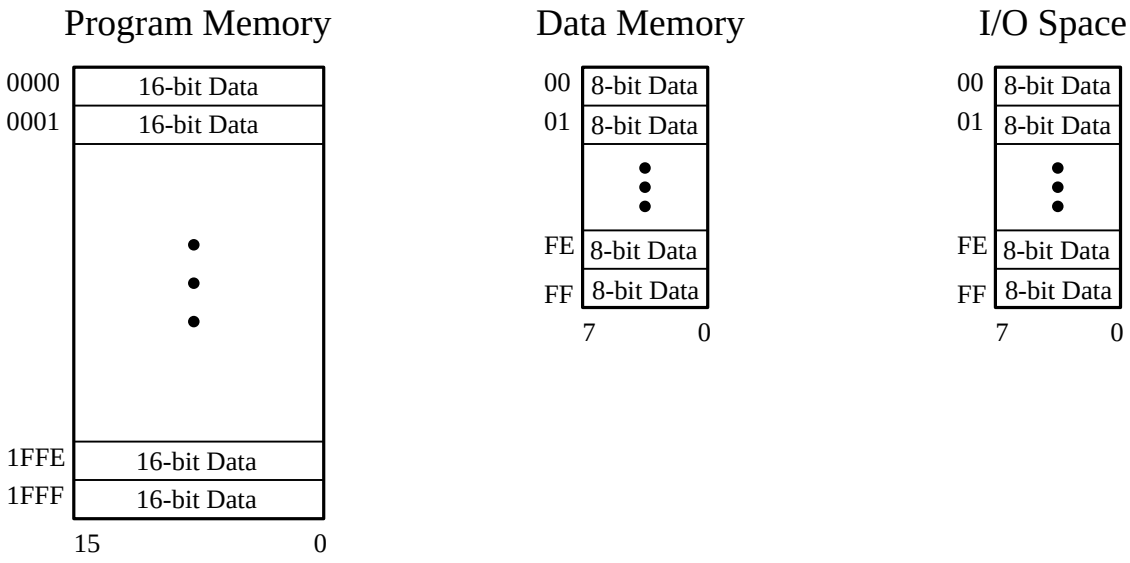
The Caltech10 CPU is an 8-bit Harvard-architecture, accumulator-based CPU. It has an 8K x 16 program memory space, a 256 x 8 data memory space, and a 256 x 8 input/output space. The CPU has an 8-bit accumulator, two 8-bit data addressing registers (X and S) and a 13-bit program counter. All instructions are one 16-bit word and almost all execute in a single clock cycle.

Block Diagram



External Memory Interface

The CPU has three distinct memory spaces and two separate data and address busses. The available memory consists of program memory (8192 16-bit words), data memory (256 8-bit words), and input/output (256 8-bit words). Each is described in more detail below.



Program Memory

The CPU program memory space consists of 8192 16-bit words and has its own address and data lines. For accessing program memory the processor has thirteen (13) address lines (ProgramAB0 to ProgramAB12) and sixteen (16) data lines (ProgramDB0 to ProgramDB15). There is no read or write signal since the program memory is only read by the processor. Note that the program memory address lines represent a word address, not a byte address since only words are read from the memory.

Data Memory

The CPU data memory space consists of 256 8-bit words and has its own address and data lines (shared with the I/O space). For accessing the data memory the processor has eight (8) address lines (DataAB0 to DataAB7) and eight (8) data lines (DataDB0 to DataDB7). Data memory is not accessed on every clock and the RD and WR signals are used to indicate if the memory is being read or written. The address lines are valid the entire time RD or WR is active. When reading data the incoming data is latched on the inactive edge of RD. When writing data the data bus is valid the entire time WR is active.

Input/Output Space

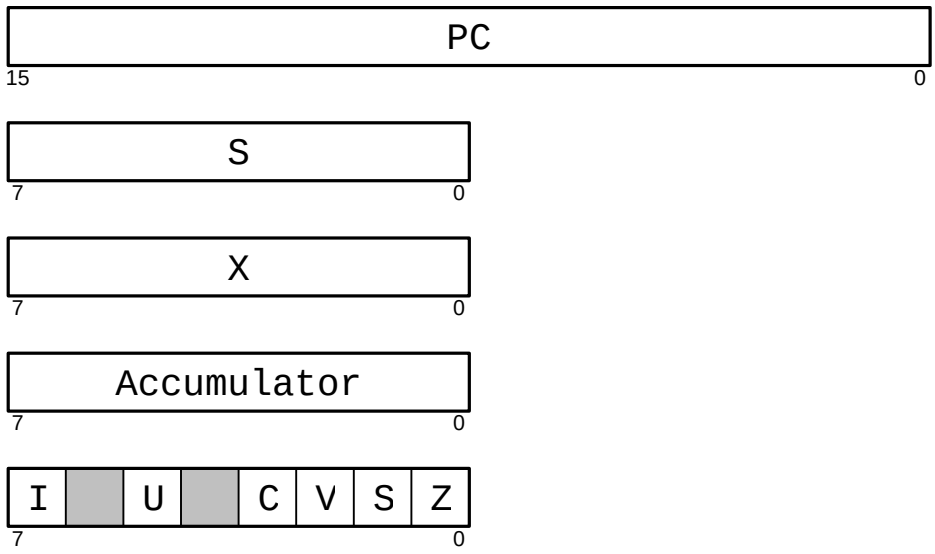
The CPU can access 256 8-bit I/O locations. The input/output space shares data and address lines with the data memory space and the IO signal is active to indicate input/output space is being accessed as opposed to data memory space. This does not cause any access issues since the CPU can only access one data value at a time. The data address lines (DataAB0 to DataAB7) are used to indicate the input or output location being accessed. The I/O data is communicated on the data memory data lines (DataDB0 to DataDB7). To indicate an input location is being accessed the RD and IO signals are both active. To indicate an output location is being accessed the WR and IO signals are both active. The address lines and IO signal are valid the entire time RD or WR is active. When reading data the incoming data is latched on the inactive edge of RD. When writing data the data bus is valid the entire time WR is active.

CPU Architecture

The CPU contains a number of blocks to allow it to execute code. The registers are used for arithmetic and logical results as well as addressing. The processor is accumulator-based so the accumulator is always one operand to the ALU and any ALU results are stored in the accumulator. The status register holds summary information about any ALU operation and is used for conditional branching. The ALU performs the arithmetic and logical operations for the CPU. The program memory interface fetches the next instruction to be executed and contains the program counter. The data memory interface handles reading and writing data and input/output values to external memory and I/O space. It contains the X and S registers which may be used to generate external addresses. Each of these blocks is described in more detail below.

Register Set

The Caltech10 CPU has one 16-bit register (the program counter) and four 8-bit registers as shown below. The program counter (PC) is always used to access the instructions to be executed. While the X and S registers are almost interchangeable in the instruction set, the X register is meant to be used for general indexed addressing and the S register is meant to be used as a stack pointer. Both the X and S registers can be used to access data memory.



Accumulator

The accumulator is an 8-bit register and is part of the ALU. It is always the only operand for one operand instructions (such as `NEG`) and is always one of the operands for two operand instructions (such as `ADD`). The result of an ALU operation can only be written to the accumulator. Additionally all load (`LD`) and input (`IN`) instructions store the value read from data memory or I/O space into the accumulator and all store (`ST`) and output (`OUT`) instructions write the accumulator value to memory or I/O space.

Status Register

The status register is an 8-bit register that holds the status from an ALU operation. Not all status bits are affected by every ALU operation. For details on which bits are affected by the ALU operations see the instruction descriptions below. The status bits are unaffected by any non-ALU operation such as loads and stores. The status bits (from high bit to low bit) are given in the following table.

Bit Name	Bit Number	Description
I	7	Global Interrupt Enable/Disable
–	6	unused
U	5	User bit used by STU and CLU instructions
–	4	unused
C	3	Carry (carry out of bit 7)
V	2	Signed Overflow (resulting sign bit is wrong)
S	1	Sign (high bit of result)
Z	0	Zero (result is zero)

ALU

The ALU can perform all of the standard logic and arithmetic operations. These include Boolean operations, shifts and rotates, addition, subtraction, and comparison. One operand is always the accumulator. The other operand (for two operand instructions) is a value from data memory or an immediate value from the instruction. The result, if stored, is always stored in the accumulator. All ALU operations affect the sign (S) and zero (Z) flags. Arithmetic and shift operations also affect the overflow (V) and carry (C) flags. The actual operation the ALU is to perform is encoded in the instruction (see below).

Data Memory Interface

The data memory interface is used to access the data memory and I/O space. It has eight (8) bits of address and eight (8) bits of data and thus can access 256 bytes in data memory and 256 bytes of I/O. The data memory address can be formed from an absolute address in the instruction or by adding an offset in the instruction to the value of one of the index registers (X and S). The data memory interface includes a RD signal to indicate a read operation, a WR signal to indicate a write operation, and an IO signal to indicate I/O space is being accessed instead of the data memory.

Program Memory Interface

The program memory interface is used to access the code stored in the program memory. It is only used to fetch instructions and thus only reads the program memory. It has thirteen (13) bits of address and sixteen (16) bits of data and thus can access 8192 16-bit words of program memory. The program memory address is always either from the program counter (PC register), possibly modified with a signed value from the instruction, or from the instruction itself for unconditional jumps and subroutine calls.

Control Unit and Instruction Register

The control unit contains the 16-bit Instruction Register that holds the instruction to be executed and the logic for decoding this instruction. The control unit also contains a simple state machine to control instruction execution. Since all instructions except CALL and RTS execute in a single clock, the state machine is only needed for those two instructions.

Instruction Set

All instructions are a single 16-bit word. This word is divided into bit fields to indicate the operation to be performed by the CPU. All instructions execute in 1 cycle except the subroutine instructions, `CALL` and `RTS` which take 4 and 3 cycles respectively.

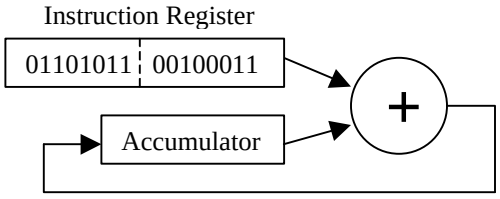
Each of the instructions is described in greater detail below.

Data Addressing Modes

Instructions that have more than one operand have a number of addressing modes available for accessing the second operand. These modes are immediate addressing, direct addressing, indexed addressing, and indexed addressing with pre- or post- increment or decrement. The last mode is only available in load and store instructions. Each of these modes is described in greater detail below

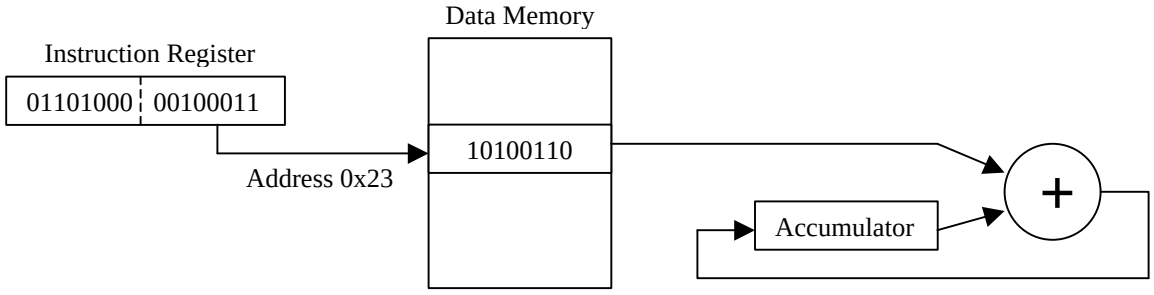
Immediate Addressing

The operand is in the instruction. Its value is part of the instruction. This mode is typically used for constants in comparisons and arithmetic expressions. For example, the instruction ADDI $23 would indicate the following operation:



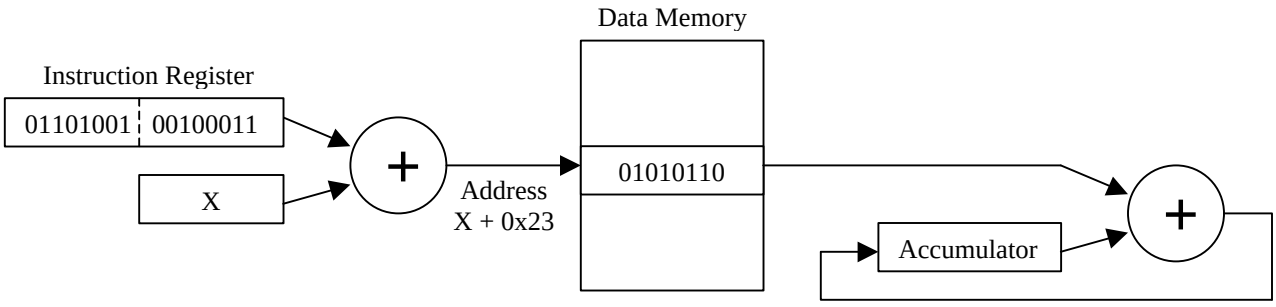
Direct Addressing

The address of the operand is in the instruction. The value operand is located at the specified address in data memory. This mode is typically used for accessing variables stored at fixed locations. For example, the instruction ADD $23 would indicate the following operation:



Indexed Addressing

The address of the operand is the offset (unsigned 8-bit value) given in the instruction plus the value in the index register (X or S) specified in the instruction. This mode is typically used to access elements of a string or array. It can also be used for indirect register addressing when the offset is 0. For example, the instruction ADD X, $23 would indicate the following operation:



Indexed Addressing with Pre- or Post- Increment or Decrement

This mode is only available in load and store instructions and is identical to indexed addressing except that the index register (either S or X) is incremented or decremented either before or after it is used to form the address. This mode is typically used to sequentially access elements of a string or array.

Instruction Addressing Modes

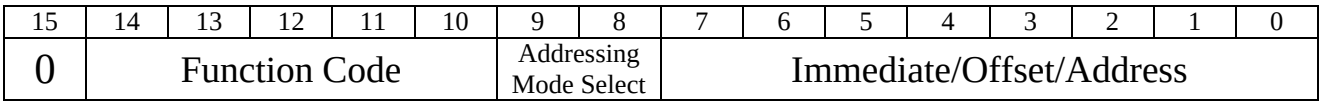
Flow control instructions such as jumps and calls have an additional addressing mode, relative addressing. In relative addressing the address of the location in program memory to jump to is the address of the next instruction plus the signed value in the low eight bits of the instruction. Also, the direct addressing mode has a 13-bit address instead of an 8-bit address.

For the instruction descriptions the following abbreviations are used for the operands:

- a 13-bit absolute program memory address
- r 8-bit signed program memory address offset
- k 8-bit immediate value
- m 8-bit absolute data memory address
- o 8-bit unsigned data memory address offset
- p 8-bit I/O port number

ALU Instructions

ALU instructions implement all of the arithmetic, logical, and shift operations for the CPU. All ALU operations affect the sign (S) and zero (Z) flags. Additionally all instructions other than the logical instructions affect the carry (C) and overflow (V) flags. The format for the ALU instructions is given below.



The Function Code indicates which operation should be performed. The Addressing Mode Select bits determine the addressing mode for the second operand and the interpretation of the low 8-bits. The coding for these bits is 00 for absolute addressing (low 8 bits are the absolute address of the operand), 01 for X register indexed addressing (low 8 bits are the offset to add to the X register to form the address of the operand), 10 for S register indexed addressing (low 8 bits are the offset to add to the S register to form the address of the operand), and 11 for immediate addressing (low 8 bits are the value of the operand).

Below is the encoding for each of the ALU instructions.

Instruction	Opcode	Operands	Flags Affected	Description
01100000mmmmmmmm	ADC	m	S V C Z	Acc ← Acc + (m) + C
01100001oooooooo	ADC	X, o	S V C Z	Acc ← Acc + (X + o) + C
01100010oooooooo	ADC	S, o	S V C Z	Acc ← Acc + (S + o) + C
01100011kkkkkkkk	ADCI	k	S V C Z	Acc ← Acc + k + C
01101000mmmmmmmm	ADD	m	S V C Z	Acc ← Acc + (m)
01101001oooooooo	ADD	X, o	S V C Z	Acc ← Acc + (X + o)
01101010oooooooo	ADD	S, o	S V C Z	Acc ← Acc + (S + o)
01101011kkkkkkkk	ADDI	k	S V C Z	Acc ← Acc + k
01000100mmmmmmmm	AND	m	S Z	Acc ← Acc • (m)
01000101oooooooo	AND	X, o	S Z	Acc ← Acc • (X + o)
01000110oooooooo	AND	S, o	S Z	Acc ← Acc • (S + o)
01000111kkkkkkkk	ANDI	k	S Z	Acc ← Acc • k
0111000100000001	ASR	–	S V C Z	arithmetic shift right Acc
00110000mmmmmmmm	CMP	m	S V C Z	Acc – (m)
00110001oooooooo	CMP	X, o	S V C Z	Acc – (X + o)
00110010oooooooo	CMP	S, o	S V C Z	Acc – (S + o)
00110011kkkkkkkk	CMPI	k	S V C Z	Acc – k
0111101100000000	DEC	–	S V C Z	Acc ← Acc – 1
0000000000000000	INC	–	S V C Z	Acc ← Acc + 1

Instruction	Opcode	Operands	Flags Affected	Description
0101100000000000	LSL	–	S V C Z	logical shift left Acc
0111000100000000	LSR	–	S V C Z	logical shift right Acc
0010011100000000	NEG	–	S V C Z	Acc ← -Acc
0010110100000000	NOT	–	S Z	Acc ← NOT Acc
01110100mmmmmmmm	OR	m	S Z	Acc ← Acc OR (m)
01110101oooooooo	OR	X, o	S Z	Acc ← Acc OR (X + o)
01110110oooooooo	OR	S, o	S Z	Acc ← Acc OR (S + o)
01110111kkkkkkkk	ORI	k	S Z	Acc ← Acc OR k
0101000000000000	RLC	–	S V C Z	rotate left Acc thru C
0101001000000000	ROL	–	S V C Z	rotate left Acc
0111000100000010	ROR	–	S V C Z	rotate right Acc
0111000100000011	RRC	–	S V C Z	rotate right Acc thru C
00011000mmmmmmmm	SBB	m	S V C Z	Acc ← Acc – (m) – C
00011001oooooooo	SBB	X, o	S V C Z	Acc ← Acc – (X + o) – C
00011010oooooooo	SBB	S, o	S V C Z	Acc ← Acc – (S + o) – C
00011011kkkkkkkk	SBBI	k	S V C Z	Acc ← Acc – k – C
00010000mmmmmmmm	SUB	m	S V C Z	Acc ← Acc – (m)
00010001oooooooo	SUB	X, o	S V C Z	Acc ← Acc – (X + o)
00010010oooooooo	SUB	S, o	S V C Z	Acc ← Acc – (S + o)
00010011kkkkkkkk	SUBI	k	S V C Z	Acc ← Acc – k
01001100mmmmmmmm	TST	m	S Z	Acc • (m)
01001101oooooooo	TST	X, o	S Z	Acc • (X + o)
01001110oooooooo	TST	S, o	S Z	Acc • (S + o)
01001111kkkkkkkk	TSTI	k	S Z	Acc • k
00110100mmmmmmmm	XOR	m	S Z	Acc ←Acc ⊕ (m)
00110101oooooooo	XOR	X, o	S Z	Acc ←Acc ⊕ (X + o)
00110110oooooooo	XOR	S, o	S Z	Acc ←Acc ⊕ (S + o)
00110111kkkkkkkk	XORI	k	S Z	Acc ←Acc ⊕ k

Flag Instructions

The flag instructions are used to set and reset the status register bits. The instructions take no operands and only affect the flag indicated. Below is the encoding for each of the flag instructions.

Instruction	Opcode	Operand	Action
0111111110000001	STI	–	set interrupt (I) flag
0000011101101001	CLI	–	clear interrupt (I) flag
0111111100100010	STU	–	set user (U) flag
0000011111001010	CLU	–	clear user (U) flag
0111111100001100	STC	–	set carry (C) flag
0000011111100100	CLC	–	clear carry (C) flag

Index Register Instructions

The index register instructions are used to manipulate the X and S registers. The instructions take no operands and do not affect the flags. Below is the encoding for each of the index register instructions.

Instruction	Opcode	Operand	Action
0000011110000000	TAX	–	X ← Accumulator
0110011100000001	TXA	–	Accumulator ← X
0000010110000000	INX	–	X ← X + 1
0000110110000000	DEX	–	X ← X – 1
0000011101010000	TAS	–	S ← Accumulator
0110011100000000	TSA	–	Accumulator ← S
0000011001000000	INS	–	S ← S + 1
0000111001000000	DES	–	S ← S – 1

Load / Store Instructions

Load and store instructions read and write the accumulator from/to data memory. The flags are not affected by load and store operations. The available addressing modes are immediate (load only), absolute addressing, indexed addressing, and indexed addressing with pre- or post- increment or decrement. Indexed addressing can use the S or X register. The low byte of the instruction is always the immediate value, absolute address, or indexed address offset, depending on the addressing mode. The encoding for the load and store operations is given below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	LD (00) / ST (01)		Pre / Post	Inc / Dec	S / X	Addressing Mode Select		Immediate/Offset/Address							

Bits 13 and 14 indicate whether the instruction is a load or store operation. The Addressing Mode Select bits determine the addressing mode for the operation and the interpretation of the low 8-bits of the instruction. The coding for these bits is given below.

Addressing Mode Select Bits	Addressing Mode	Interpretation of Bits 7 to 0
00	absolute addressing	data address to read or write
01	immediate	value to load
10	indexed addressing with pre- or post- increment or decrement	offset to add to the index register to form the data address to read or write
11	indexed addressing	offset to add to the index register to form the data address to read or write

Bit 12 is 0 for pre-increment or -decrement and 1 for post-increment or -decrement. Bit 11 is 0 for pre- or post-increment and 1 for pre- or post-decrement. Bit 10 selects the S register (0) or X register (1) for indexed addressing.

Below is the encoding for each of the load and store instructions.

Instruction	Opcode	Operand	Result	Description
10001001kkkkkkkk	LDI	k	Acc ← k	load Accumulator with immediate value k
10000000mmmmmmmm	LDD	m	Acc ← (m)	load Accumulator direct from (m)
10010111oooooooo	LD	X + o	Acc ← (X + o)	load Accumulator indirect from (X + o)
10010110oooooooo	LD	X+ + o	Acc ← (X + o) X ← X + 1	load Accumulator indirect from (X + o) and post-increment X
10011110oooooooo	LD	X– + o	Acc ← (X + o) X ← X – 1	load Accumulator indirect from (X + o) and post-decrement X
10000110oooooooo	LD	+X + o	X ← X + 1 Acc ← (X + o)	load Accumulator indirect from (X + o) and pre-increment X

Instruction	Opcode	Operand	Result	Description
10001110oooooooo	LD	–X + o	X ← X – 1 Acc ← (X + o)	load Accumulator indirect from (X + o) and pre-decrement X
10010011oooooooo	LD	S + o	Acc ← (S + o)	load Accumulator indirect from (S + o)
10010010oooooooo	LD	S+ + o	Acc ← (S + o) S ← S + 1	load Accumulator indirect from (S + o) and post-increment S
10011010oooooooo	LD	S– + o	Acc ← (S + o) S ← S – 1	load Accumulator indirect from (S + o) and post-decrement S
10000010oooooooo	LD	+S + o	S ← S + 1 Acc ← (S + o)	load Accumulator indirect from (S + o) and pre-increment S
10001010oooooooo	LD	–S + o	S ← S – 1 Acc ← (S + o)	load Accumulator indirect from (S + o) and pre-decrement S
10100000mmmmmmmm	STD	m	(m) ← Acc	store Accumulator direct at (m)
10110111oooooooo	ST	X + o	(X + o) ← Acc	store Accumulator indirect at (X + o)
10110110oooooooo	ST	X+ + o	(X + o) ← Acc X ← X + 1	store Accumulator indirect at (X + o) and post-increment X
10111110oooooooo	ST	X– + o	(X + o) ← Acc X ← X – 1	store Accumulator indirect at (X + o) and post-decrement X
10100110oooooooo	ST	+X + o	X ← X + 1 (X + o) ← Acc	store Accumulator indirect at (X + o) and pre-increment X
10101110oooooooo	ST	–X + o	X ← X – 1 (X + o) ← Acc	store Accumulator indirect at (X + o) and pre-decrement X
10110011oooooooo	ST	S + o	(S + o) ← Acc	store Accumulator indirect at (S + o)
10110010oooooooo	ST	S+ + o	(S + o) ← Acc S ← S + 1	store Accumulator indirect at (S + o) and post-increment S
10111010oooooooo	ST	S– + o	(S + o) ← Acc S ← S – 1	store Accumulator indirect at (S + o) and post-decrement S
10100010oooooooo	ST	+S + o	S ← S + 1 (S + o) ← Acc	store Accumulator indirect at (S + o) and pre-increment S
10101010oooooooo	ST	–S + o	S ← S – 1 (S + o) ← Acc	store Accumulator indirect at (S + o) and pre-decrement S

Branch Instructions

Branch instructions are used to control the flow of the program. The instruction set includes both conditional and unconditional branches. All branches also execute the instruction immediately following the branch whether the branch is taken or not. Below is the encoding of the unconditional jump instruction.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	Address												

The unconditional jump instruction contains the address to jump to in the low 13-bits. Following is the encoding of the conditional jump instructions.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	Condition		1	Condition	Invert Condition		Relative Address							

The conditional jumps are based on the condition and contain a relative number of instructions (signed value) to jump if the condition is met. The relative address (range -128 to +127) is added to the address of the instruction following the jump. Thus the relative jump range is -127 to +128 instructions from the conditional jump instruction itself. The conditions are encoded according to the following table.

Condition Bits 13, 12, 10	Invert Condition Bits 9, 8	Condition
000	00	(C = 0) AND (Z = 0)
000	11	(C = 1) OR (Z = 1)
001	00	C = 0
001	11	C = 1
010	00	S = 0
010	11	S = 1
011	00	Z = 0
011	11	Z = 1
100	00	V = 0
100	11	V = 1
101	00	(S ≠ V) OR (Z = 1)
101	11	(S = V) AND (Z = 0)
110	00	S ≠ V
110	11	S = V
111	00	U = 0
111	11	U = 1

Below is the encoding for each of the branch instructions.

Instruction	Opcode	Operand	Cycles	Action
110aaaaaaaaaaaaa	JMP	a	1	PC ← a
10001000rrrrrrrr	JA	r	1	branch if above if (C = 0) AND (Z = 0) then PC ← PC + r + 1
10001100rrrrrrrr	JAE / JNC	r	1	branch if carry bit clear if C = 0 then PC ← PC + r + 1
10001111rrrrrrrr	JB / JC	r	1	branch if carry bit set if C = 1 then PC ← PC + r + 1
10001011rrrrrrrr	JBE	r	1	branch if below or equal if (C = 1) OR (Z = 1) then PC ← PC + r + 1
10011111rrrrrrrr	JE / JZ	r	1	branch if zero bit set (equal) if Z = 1 then PC ← PC + r + 1
10101111rrrrrrrr	JG	r	1	branch if greater than if (S = V) AND (Z = 0) then PC ← PC + r + 1
10111011rrrrrrrr	JGE	r	1	branch if greater than or equal if S = V then PC ← PC + r + 1
10111000rrrrrrrr	JL	r	1	branch if less than if S ≠ V then PC ← PC + r + 1
10101100rrrrrrrr	JLE	r	1	branch if less than or equal if (S ≠ V) OR (Z = 1) then PC ← PC + r + 1
10011100rrrrrrrr	JNE / JNZ	r	1	branch if zero bit clear (not equal) if Z = 0 then PC ← PC + r + 1
10011000rrrrrrrr	JNS	r	1	branch if sign bit clear (positive) if S = 0 then PC ← PC + r + 1
10111100rrrrrrrr	JNU	r	1	branch if U-bit clear if U = 0 then PC ← PC + r + 1
10101000rrrrrrrr	JNV	r	1	branch if overflow bit clear if V = 0 then PC ← PC + r + 1
10011011rrrrrrrr	JS	r	1	branch if sign bit set (negative) if S = 1 then PC ← PC + r + 1
10111111rrrrrrrr	JU	r	1	branch if U-bit set if U = 1 then PC ← PC + r + 1
10101011rrrrrrrr	JV	r	1	branch if overflow bit set if V = 1 then PC ← PC + r + 1

Subroutine and Stack Instructions

The subroutine and stack instructions are used to implement subroutines and allow the flags to be pushed and popped on or off the stack. Pushing a value on the stack does a pre-decrement of the S register and popping a value off the stack does a post-increment of the S register. Other stack operations can be done using indexed addressing with the S register. The `CALL` and `RTS` instructions are the only instructions that take more than 1 clock to execute. The `CALL` instruction executes in 4 clocks and the `RTS` instruction executes in 3 clocks. Below is the encoding for each of the subroutine and stack instructions.

Instruction	Opcode	Operand	Cycles	Action
`111aaaaaaaaaaaa`	CALL	a	4	direct subroutine call $S \leftarrow S - 1$; $(S) \leftarrow (PC + 1)_{12:8}$; $S \leftarrow S - 1$; $(S) \leftarrow (PC + 1)_{7:0}$; $PC \leftarrow a$
`0001111100000000`	RTS	–	3	return from subroutine (PC popped off stack) $PC_{7:0} \leftarrow (S)$; $S \leftarrow S + 1$; $PC_{12:8} \leftarrow (S)$; $S \leftarrow S + 1$
`0000001000000000`	POPF	–	1	pop flag register off of the stack Flags $\leftarrow (S)$; $S \leftarrow S + 1$
`0000111000000000`	PUSHF	–	1	push flag register onto the stack $S \leftarrow S - 1$; $(S) \leftarrow$ Flags

I/O Instructions

The I/O instructions are used to read (`IN`) and write (`OUT`) the accumulator value from/to I/O space. The address to read or write in I/O space is given by the port number in the low eight (8) bits of the instruction. Below is the encoding for the `IN` and `OUT` instructions.

Instruction	Opcode	Operand	Action
`10010000pppppppp`	IN	p	read from I/O port Accumulator $\leftarrow$ I/O port p
`10110000pppppppp`	OUT	p	output to I/O port I/O port p $\leftarrow$ Accumulator

Miscellaneous Instructions

The encoding for each of the miscellaneous instruction is given below.

Instruction	Opcode	Operand	Action
`0001111110000000`	NOP	–	no operation

Instruction Encoding Table

Instruction Word Bits 15 to 12	Instruction Word Bits 11 to 8															
	0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
0000	INC		POPF			INX	INS	*1						DEX	PUSHF DES	
0001	SUB m	SUB X, o	SUB S, o	SUBI k					SBB m	SBB X, o	SBB S, o	SBBI k				RTS NOP
0010								NEG						NOT		
0011	CMP m	CMP X, o	CMP S, o	CMPI k	XOR m	XOR X, o	XOR S, o	XORI k								
0100					AND m	AND X, o	AND S, o	ANDI k					TST m	TST X, o	TST S, o	TSTI k
0101	RLC		ROL						LSL							
0110	ADC m	ADC X, o	ADC S, o	ADCI k				TSA TXA	ADD m	ADD X, o	ADD S, o	ADDI k				
0111		*2			OR m	OR X, o	OR S, o	ORI k				DEC				*3
1000	LDD m		LD +S, o				LD +X, o		JA r	LDI k	LD -S, o	JBE r	JAE r JNC r		LD -X, o	JB r JC r
1001	IN p		LD S+, o	LD S, o			LD X+, o	LD X, o	JNS r		LD S-, o	JS r	JNE r JNZ r		LD X-, o	JE r JZ r
1010	STD m		ST +S, o				ST +X, o		JNV r		ST -S, o	JV r	JLE r		ST -X, o	JG r
1011	OUT p		ST S+, o	ST S, o			ST X+, o	ST X, o	JL r		ST S-, o	JGE r	JNU r		ST X-, o	JU r
1100	JMP a															
1101	JMP a															
1110	CALL a															
1111	CALL a															

*1 CLI (0000011101101001)
CLU (0000011111001010)
CLC (0000011111100100)
TAS (0000011101010000)
TAX (0000011110000000)

*2 LSR (0111000100000000)
ASR (0111000100000001)
ROR (0111000100000010)
RRC (0111000100000011)

*3 STC (0111111100001100)
STU (0111111100100010)
STI (0111111110000001)