

# DIGIRULE 2A



# USER MANUAL

# DESCRIPTION

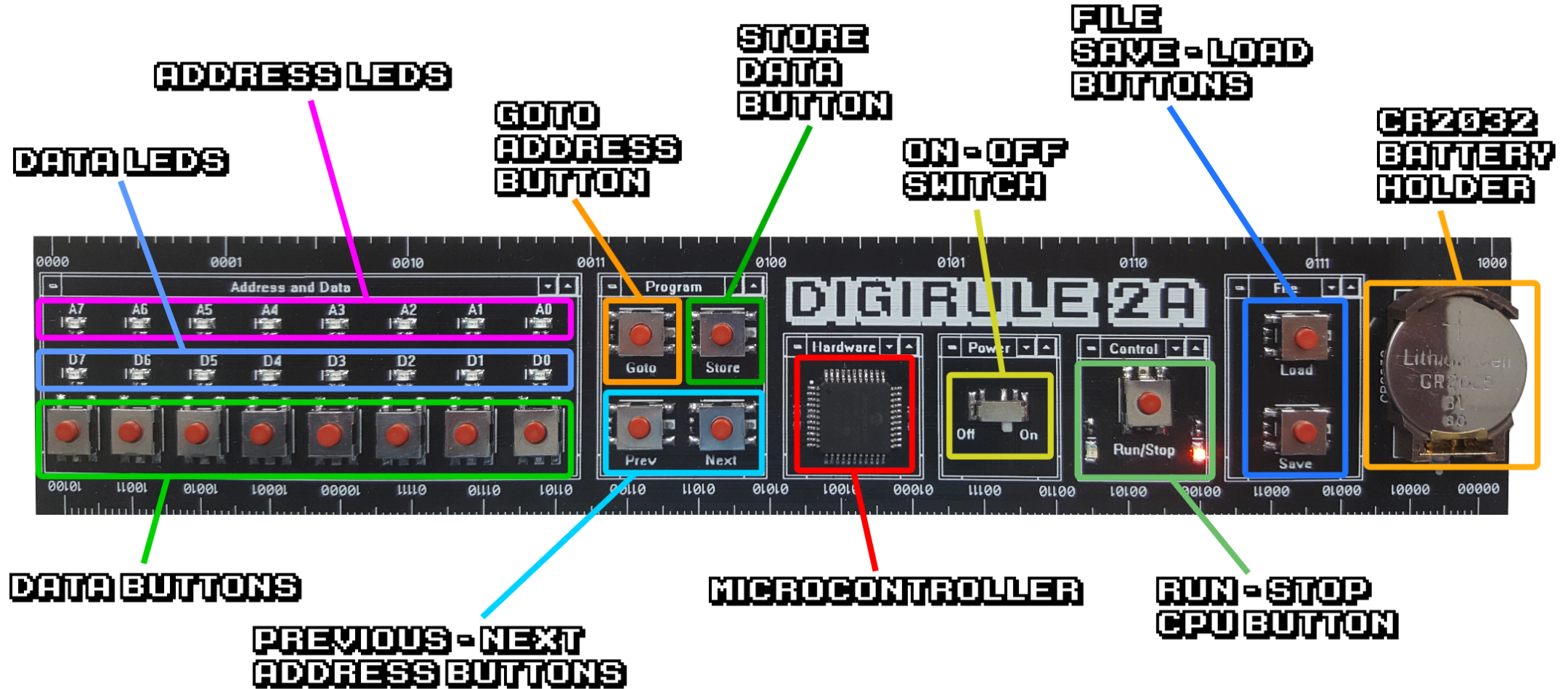
The Digirule 2A is an open source hardware - programmable 8-bit binary computer built into a 20cm (8") PCB ruler and is powered by an 8-bit Microchip PIC18F45K20 microcontroller.

Full details can be found here: [bradsprojects.com/digirule2](https://bradsprojects.com/digirule2)

Note: The Digirule 2A is a slightly modified version of the Digirule 2. It has improved firmware which includes better button response, two new instructions and a new start-up animation. You can find details of the original Digirule 2 from the link above (including instruction manual download).

# OVERVIEW

The top of the Digirule 2A contains the buttons, power switch, battery, microcontroller and LED's:





The Rear of the Digirule 2A contains a summary of the instruction set, logo's, ICSP (programming port) header and some basic help details:

**LOGO'S**

**ICSP HEADER**

Instruction Set					
00000000	HALT	(1 Byte)	00001101	ANDRA	(2 Bytes)
00000001	NOP	(1 Byte)	00001110	ORLA	(2 Bytes)
00000010	SPEED	(2 Bytes)	00001111	ORRA	(2 Bytes)
00000011	COPYLR	(3 Bytes)	00010000	XORLA	(2 Bytes)
00000100	COPYLA	(2 Bytes)	00010001	XORRA	(2 Bytes)
00000101	COPYAR	(2 Bytes)	00010010	DECR	(2 Bytes)
00000110	COPYRA	(2 Bytes)	00010011	INCR	(2 Bytes)
00000111	COPYRR	(3 Bytes)	00010100	DECRJZ	(2 Bytes)
00001000	ADDLA	(2 Bytes)	00010101	INCRJZ	(2 Bytes)
00001001	ADDRA	(2 Bytes)	00010110	SHIFTRL	(2 Bytes)
00001010	SUBLA	(2 Bytes)	00010111	SHIFTRR	(2 Bytes)
00001011	SUBRA	(2 Bytes)	00011000	CBR	(3 Bytes)
00001100	ANDLA	(2 Bytes)	00011001	SBR	(3 Bytes)
			00011010	BCRSC	(3 Bytes)
			00011011	BCRSS	(3 Bytes)
			00011100	JUMP	(2 Bytes)
			00011101	CALL	(2 Bytes)
			00011110	RETURN	(1 Byte)
			00011111	RETLA	(2 Bytes)
			00100000	ADDRPC	(2 Bytes)
			00100001	INITSP	(1 Byte)
			00100010	RANDA	(1 Byte)

**GENERAL HELP**

**ICSP**

# DIGIRULE 2A DIFFERENCES

The Digirule 2A contains the following differences compared to the Digirule 2:

- The 'Save' button in the 'program' window has been renamed to 'Store'
- The Run/Stop button and LED's have been moved slightly to make it clearer as to when a program is running or not
- The rear of the Digirule 2A has been updated to include two new instructions (instructions 33 and 34)
- The firmware has been completely revamped by Brent Hauser and is written in C (whereas the old firmware was written in basic).

The firmware update brings about the following changes:

- The SPEED instruction (instruction number 2) has been improved
- Two new instructions have been added:
  - Instruction number 33 (INITSP) initialises the stack pointer
  - Instruction number 34 (RANDA) generates a pseudo-random number between 1 and 255
- The CPU will halt if it fetches an instruction with an unknown opcode
- The buttons are more responsive regardless of what speed the CPU is running at
- Holding the Load button and then pressing the Run/Stop button will reset the CPU. This can also be used to clear the internal stack pointer
- Holding the Load button and then pressing the Prev button clears all RAM and resets the CPU.
- When entering data, holding the Data 7 button for one second will set all bits.
- When entering data, holding the Data 0 button for one second will clear all bits.
- The power-on animation has been updated
- Unused Microcontroller I/O pins are programmed as outputs, so they don't float
- A progress bar is displayed on the address LED's during a Load or Save operation
- The CPU call/return stack is now four levels deep. The internal stack pointer is automatically initialized whenever a program is loaded. Also, new instruction INITSP (opcode 33) may be used to explicitly initialize the stack pointer, such as at the beginning of a program. This will ensure the stack will not overflow if the program is restarted.

# OPERATION

## Run/Stop Modes

The Digirule 2A has two modes of operation, 'run' mode and 'stop' mode. When in stop mode, the device is able to be programmed. Programs can be stored to permanent memory and restored from permanent memory. When in run mode, the CPU is running through the program which is stored within the 256 bytes of memory. The 'run/stop' button toggles between run mode and stop mode.

## Programming the Digirule 2A

Ensuring the Digirule 2A is in stop mode, a program may be entered by using the 8 data input buttons - the status of each of the 8 bits is shown using the corresponding data LED. If a data LED is off, pressing the corresponding data button will turn it on and vice-versa. The address LED's show the memory address which is currently accessed. To store your 8 bits of data into the current address and advance to the next address in sequence - simply press the 'save' button within the 'program' window.

You can check the contents of each address by using the 'prev' and 'next' buttons, the address will show on the address LED's while the data stored within that address will show on the data LED's. To jump to a specific address, simply enter the address on the data input buttons, then press the 'goto' button.

## Storing a Program in Permanent Memory

You can store up to eight individual 256-byte programs in permanent memory. This will hold your programs even if the battery is removed. To do this, press and hold the 'save' button within the 'File' window, the data LED's will animate left and right - then press any of the eight data input buttons to store your program within that particular memory slot. You may now release the buttons.

## Loading a Program from Permanent Memory

To load one of the eight, 256-byte blocks of memory, simply press and hold the 'load' button within the 'File' window. The data LED's will animate back and forth - then press any of the eight data input buttons to load the program within that particular memory slot. You may now release the buttons.

## Running a Program

To run a program, simply access the address of the start of the program (which would normally be 00000000) and then press the 'run/stop' button. You can start a program from any address however and can even have multiple programs within a 256-byte block. For example, you could have one program which uses addresses 00000000 to 00011111 and then have another program which uses addresses 00100000 to 11111111. To run the first program, simply access address 00000000 then press the 'run/stop' button. To access the second program, simply access address 00100000 then press the 'run/stop' button.

## Stopping a Program

Simply press the 'run/stop' button while a program is running, to stop that program.

## Turning Off the Address LED's While the CPU is Running

When the CPU is in run mode, you can toggle the address LED's on and off by pressing the 'goto' button.

## Preconfigured Registers

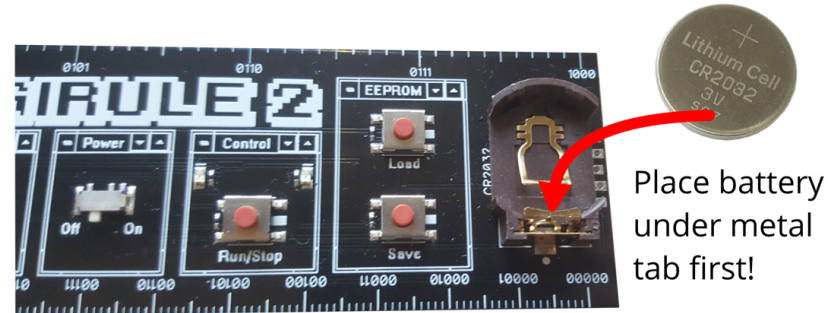
The Digirule 2A has four registers (addresses) which are set aside for specific uses. These are the last four addresses within the 256-byte memory space and are as follows:

- Address 252 (binary 11111100) = Status register. This register can be read from or written to. The discrete bits within this register are as follows:
  - Bit 0 = zero flag (logic 0 means the previous instruction resulted in a non-zero answer while a logic 1 means the previous instruction resulted in a zero answer).
  - Bit 1 = carry flag (logic 0 means the previous instruction gave a carry of 0 while a logic 1 means the previous instruction gave a carry of 1).
  - Bit 2 = address LED function flag (logic 0 means the address LED's function as normal – showing the currently accessed address while a logic 1 means the address LED's will show the data loaded into the addressLEDRegister).
- Address 253 (binary 11111101) = Button Register. This register can be read from in order to obtain the status of each of the eight data buttons. This is useful to allow user input while a program is running.
- Address 254 (binary 11111110) = Address LED Register. This register can be read from or written to. This register is written to in order to show data on the 8 address LED's. NOTE – in order for this to function, you must first set bit 2 to a logic '1' within the status register otherwise the address LED's will show the currently accessed memory address.
- Address 255 (binary 11111111) = Data LED Register. This register can be read from or written to. This register is written to in order to show data on the 8 data LED's.

# INSERTING THE BATTERY

It is important that care be taken when installing the CR2032 battery. Ensure the battery has the positive label facing up (away from the Digirule 2A) and the battery is slotted under the metal tab FIRST and then pressed down to be held by the plastic tabs. Failing to follow this procedure may result in damage to the metal tab.

 **IMPORTANT!**



# FEATURES AND SPECIFICATIONS

- 35 instructions
- Each program can be up to 256 bytes long
- 8 Permanent memory locations to back up your programs
- 8-bit data bus
- 8-bit address bus
- 8 data LED's
- 8 address LED's
- 8 data input buttons
- Address previous and next buttons
- Address go to button
- Data store (and go to next address) button
- CPU run / stop button
- 'File' load / save buttons
- Powered by a 3V CR2032 coin cell battery



# BUILT-IN PROGRAMS

As standard, each Digirule 2A comes with 8 basic built-in programs. To access a program, simply hold down the 'LOAD' button, then press any one of the Data Input buttons to load the program from that location. Once loaded, ensure the address LED's show all zeros (00000000) then press the 'run' button to execute that program.

## List of the eight built-in example programs:

- D0** - Kill the Bit game. (A single LED will shift from right to left, the player needs to press any of the 8 data input buttons in an attempt to turn the corresponding LED off. Pressing a button when the LED is on - will turn it off however pressing a button when the LED is off, will turn it on. The aim of the game is to have all LED's off.
- D1** - Smiley face persistence of vision display. (once run, press the 'GOTO' button to turn off the address LED's, then wave the ruler back and forth quite fast in a dark room to see some smiley faces drawn in the air)
- D2** - 8-bit counter (uses the data LED's to count from 0 to 255 (00000000 to 11111111))
- D3** - 16-bit counter (uses the Data LED's for the lower 8-bits and the Address LED's as the upper 8-bits)
- D4** - Knightrider KIT car scanner (The Data LED's move back and forth like the KIT car in Knightrider)
- D5** - Button test (once run, press a button to have the corresponding Data LED light up)
- D6** - Target practice (this is a game whereby you have to press a data button that corresponds with the currently lit data LED to score a point. Your score is presented on the address LED's. If you 'shoot' and miss, you lose all of your points.)
- D7** - Police flasher (this just flashes groups of 4 LED's back and forth on the Address and Data LED's)

# INSTRUCTION SET SUMMARY

Number	Instruction	Description	Bytes	Flags
0	HALT	Stop executing instructions (performs the same task as pressing the CPU RUN/STOP button)	1	None
1	NOP	No operation	1	None
2	SPEED	Set the speed that the CPU steps through each instruction once the CPU is running	2	None
3	COPYLR	Copy a literal value to the specified RAM location	3	None
4	COPYLA	Copy a literal value to the Accumulator	2	None
5	COPYAR	Copy the contents of the Accumulator to the specified RAM location	2	None
6	COPYRA	Copy the contents of the specified RAM location to the Accumulator	2	Zero
7	COPYRR	Copy the contents of one RAM location to another RAM location	3	Zero
8	ADDLA	Add a literal value to the Accumulator (result stored in Accumulator)	2	Carry, Zero
9	ADDRA	Add the contents of a RAM location to the Accumulator (result stored in Accumulator)	2	Carry, Zero
10	SUBLA	Subtract a literal value from the Accumulator (result stored in the Accumulator)	2	Carry, Zero
11	SUBRA	Subtract the contents of the specified RAM location from the Accumulator (result stored in Accumulator)	2	Carry, Zero
12	ANDLA	AND a literal value with the Accumulator (result stored in Accumulator)	2	Zero
13	ANDRA	AND the contents of the specified RAM location with the Accumulator (result stored in Accumulator)	2	Zero
14	ORLA	OR a literal value with the Accumulator (result stored in Accumulator)	2	Zero
15	ORRA	OR the contents of the specified RAM location with the Accumulator (result stored in Accumulator)	2	Zero
16	XORLA	XOR a literal value with the Accumulator (result stored in Accumulator)	2	Zero
17	XORRA	XOR the contents of the specified RAM location with the Accumulator (result stored in Accumulator)	2	Zero
18	DECR	Decrement the contents of the specified RAM location by one	2	Zero
19	INCR	Increment the contents of the specified RAM location by one	2	Zero
20	DECRJZ	Decrement the contents of the specified RAM location by one, if the result IS zero, skip the next two lines of code	2	Zero
21	INCRJZ	Increment the contents of the specified RAM location by one, if the result IS zero, skip the next two lines of code	2	Zero
22	SHIFTRL	Shift the contents of the specified RAM location left (through the carry flag) by one	2	Carry
23	SHIFTRR	Shift the contents of the specified RAM location right (through the carry flag) by one	2	Carry
24	CBR	Clear the specified bit within the specified RAM location	3	None
25	SBR	Set the specified bit within the specified RAM location	3	None
26	BCRSC	Check the specified bit within the specified RAM location, if it IS a ZERO, skip the next two lines of code	3	None
27	BCRSS	Check the specified bit within the specified RAM location, if it IS a ONE, skip the next two lines of code	3	None
28	JUMP	Jump to a specific RAM location and continue running instructions from there	2	None
29	CALL	Jump to a specific RAM location, continue running instructions from there, then return once a 'RETURN' or 'RETLA' instruction is executed	2	None
30	RETLA	Return to the very next RAM location after the 'CALL' instruction with a literal value stored in the Accumulator	2	None
31	RETURN	Return to the very next RAM location after the 'CALL' instruction	1	None

32	ADDRPC	Add the contents of a RAM location to the Program Counter	2	None
33	INITSP	Initialise the Stack Pointer	1	None
34	RANDA	Generates a random number between 1 and 255 (00000001 and 11111111) and stores it in the accumulator	1	None

# INSTRUCTION SET

## 0. HALT

**Description:** Stop executing instructions

**Usage:** [HALT]

**Program Bytes Used:** 1

**Status Flags Affected:** None

**Example:**

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	HALT	00000000	0

## 1. NOP

**Description:** Does not perform any operation

**Usage:** [NOP]

**Program Bytes Used:** 1

**Status Flags Affected:** None

**Example:**

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	NOP	00000001	1

## 2. SPEED

**Description:** Set the speed that the Digirule 2A steps between instructions (the higher the number – the slower the speed)

**Usage:** [SPEED] [Value to copy]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Set the speed to 129

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SPEED	00000010	2
00000001	129	10000001	129

**Example 2:** Set the speed to 23

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SPEED	00000010	2
00000001	23	00010111	23



### 3. COPYLR

**Description:** Copy a literal value to the specified RAM location

**Usage:** [COPYLR] [Value to copy] [RAM location to store the Value]

**Program Bytes Used:** 3

**Status Flags Affected:** None

**Example 1:** Copy the number 35 (binary 00100011) to RAM location 240 (binary 11110000)

**Pre-conditions:** RAM location 240 = 0 (binary 00000000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYLR	00000011	3
00000001	35	00100011	35
00000010	240	11110000	240

**Post-conditions:** RAM location 240 = 35 (binary 00100011)

**Example 2:** Copy the number 82 (binary 01010010) to 'myVariable' ('myVariable' is an alias for RAM location 241)

**Pre-conditions:** 'myVariable' = 15 (binary 00001111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYLR	00000011	3
00000001	82	01010010	82
00000010	myVariable	11110001	241

**Post-conditions:** 'myVariable' = 82 (binary 01010010)

#### 4. COPYLA

**Description:** Copy a literal value to the Accumulator.

**Usage:** [COPYLA] [Value to copy]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Copy the number 123 (binary 01111011) to the Accumulator

**Pre-conditions:** Accumulator = 16 (binary 00010000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYLA	00000100	4
00000001	123	01111011	123

**Post-conditions:** Accumulator = 123 (binary 01111011)

## 5. COPYAR

**Description:** Copy the contents of the Accumulator to the specified RAM location

**Usage:** [COPYAR] [RAM location to store the number]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Copy the contents of the Accumulator to RAM location 250 (binary 11111010).

**Pre-conditions:** Accumulator = 2 (binary 00000010)  
RAM location 250 (binary 11111010) = 7 (binary 00000111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYAR	00000101	5
00000001	250	11111010	250

**Post-conditions:** Accumulator = 2 (binary 00000010)  
RAM location 250 (binary 11111010) = 2 (binary 00000010)

**Example 2:** Copy the contents of the Accumulator to 'myVariable' ('myVariable' is an alias for RAM location 252)

**Pre-conditions:** Accumulator = 129 (binary 10000001)  
'myVariable' = 20 (binary 00010100)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYAR	00000101	5
00000001	myVariable	11111100	252

**Post-conditions:** Accumulator = 129 (binary 10000001)  
'myVariable' = 129 (binary 10000001)

## 6. COPYRA

**Description:** Copy the contents of the specified RAM location to the Accumulator

**Usage:** [COPYRA] [RAM location to copy the value from]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** Copy the contents of RAM location 243 (binary 11110011) to the Accumulator

**Pre-conditions:** Accumulator = 16 (binary 00010000)  
RAM location 243 (binary 11110011) = 127 (binary 01111111)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYRA	00000110	6
00000001	243	11110011	243

**Post-conditions:** Accumulator = 127 (binary 01111111)  
RAM location 243 (binary 11110011) = 127 (binary 01111111)  
Zero Flag = 0

**Example 2:** Copy the contents of 'myVariable' to the accumulator

**Pre-conditions:** Accumulator = 56 (binary 00111000)  
'myVariable' = 32 (binary 00100000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYRA	00000110	6
00000001	myVariable	11111100	252

**Post-conditions:** Accumulator = 32 (binary 00100000)  
'myVariable' = 32 (binary 00100000)

## 7. COPYRR

**Description:** Copy the contents of one RAM location to another

**Usage:** [COPYRR] [RAM location to copy the value from] [RAM location to copy the value to]

**Program Bytes Used:** 3

**Status Flags Affected:** Zero Flag

**Example 1:** Copy the contents of RAM location 252 (binary 11111100) to RAM location 253 (binary 11111101)

**Pre-conditions:** RAM location 252 (binary 11111100) = 10 (binary 00001010)  
RAM location 253 (binary 11111101) = 0 (binary 00000000)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYRR	00000111	7
00000001	252	11111100	252
00000010	253	11111101	253

**Post-conditions:** RAM location 252 (binary 11111100) = 10 (binary 00001010)  
RAM location 253 (binary 11111101) = 10 (binary 00001010)  
Zero Flag = 0

**Example 2:** Copy the contents of 'myVariable' to 'myOtherVariable' ('myVariable' is an alias for RAM location 244 while 'myOtherVariable' is an alias for RAM location 245)

**Pre-conditions:** 'myVariable' = 0 (binary 00000000)  
'myOtherVariable' = 25 (binary 00011001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	COPYRR	00000111	7
00000001	myVariable	11110100	244
00000010	myOtherVariable	11110101	245



**Post-conditions:** 'myVariable' = 0 (binary 00000000)  
'myOtherVariable' = 0 (binary 00000000)  
Zero Flag = 1

## 8. ADDLA

**Description:** Add a literal value to the Accumulator

**Usage:** [ADDLA] [Value to add]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag, Carry Flag

**Example 1:** Add the literal value 15 (binary 00001111) to the Accumulator

**Pre-conditions:** Accumulator = 0 (binary 00000000)  
Zero Flag = 1  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ADDLA	00001000	8
00000001	15	00001111	15

**Post-conditions:** Accumulator = 15 (binary 00001111)  
Zero Flag = 0  
Carry Flag = 0

**Example 2:** Add the literal value 1 (binary 00000001) to the Accumulator

**Pre-conditions:** Accumulator = 255 (binary 11111111)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ADDLA	00001000	8
00000001	1	00000001	1

**Post-conditions:** Accumulator = 0 (binary 00000000)  
Zero Flag = 1  
Carry Flag = 1

## 9. ADDRA

**Description:** Add the value contained within a RAM location to the Accumulator

**Usage:** [ADDRA] [RAM location of value to add to the Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag, Carry Flag

**Example 1:** Add the value stored within RAM location 250 (binary 11111010) to the Accumulator

**Pre-conditions:** Accumulator = 100 (binary 01100100)  
RAM location 250 (binary 11111010) = 100 (binary 01100100)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ADDRA	00001001	9
00000001	250	11111010	250

**Post-conditions:** Accumulator = 200 (binary 11001000)  
RAM location 250 (binary 11111010) = 100 (binary 01100100)  
Zero Flag = 0  
Carry Flag = 0

**Example 2:** Add the value stored within 'myVariable' to the Accumulator ('myVariable' is an alias for RAM location 249)

**Pre-conditions:** Accumulator = 40 (binary 00101000)  
'myVariable' = 72 (binary 01001000)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ADDRA	00001001	9
00000001	myVariable	11111001	249

**Post-conditions:** Accumulator = 112 (binary 01110000)  
                      'myVariable' = 72 (binary 01001000)  
                      Zero Flag = 0  
                      Carry Flag = 0

## 10. SUBLA

**Description:** Subtract a literal value from the Accumulator

**Usage:** [SUBLA] [Value to subtract]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag, Carry Flag

**Example 1:** Subtract the literal value 20 (binary 00010100) from the Accumulator

**Pre-conditions:** Accumulator = 20 (binary 00010100)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SUBLA	00001010	10
00000001	20	00010100	20

**Post-conditions:** Accumulator = 0 (binary 00000000)  
Zero Flag = 1  
Carry Flag = 0

**Example 2:** Subtract the literal value 41 (binary 00101001) from the Accumulator

**Pre-conditions:** Accumulator = 40 (binary 00101000)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SUBLA	00001010	10
00000001	41	00101001	41

**Post-conditions:** Accumulator = 255 (binary 11111111)  
Zero Flag = 0  
Carry Flag = 1



## 11. SUBRA

**Description:** Subtract the value stored in a RAM location, from the Accumulator

**Usage:** [SUBRA] [RAM location of value to subtract from the Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag, Carry Flag

**Example 1:** Subtract the value stored within RAM location 250 (binary 11111010) from the Accumulator

**Pre-conditions:** Accumulator = 201 (binary 11001001)  
RAM location 250 (binary 11111010) = 34 (binary 00100010)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SUBRA	00001011	11
00000001	250	11111010	250

**Post-conditions:** Accumulator = 167 (binary 10100111)  
RAM location 250 (binary 11111010) = 34 (binary 00100010)  
Zero Flag = 0  
Carry Flag = 0

**Example 2:** Subtract the value stored within 'myVariable' from the Accumulator ('myVariable' is an alias for RAM location 245)

**Pre-conditions:** Accumulator = 255 (binary 11111111)  
'myVariable' = 255 (binary 11111111)  
Zero Flag = 0  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SUBRA	00001011	11
00000001	myVariable	11110101	245

**Post-conditions:** Accumulator = 0 (binary 00000000)  
                  'myVariable' = 255 (binary 11111111)  
                  Zero Flag = 1  
                  Carry Flag = 0

## 12. ANDLA

**Description:** AND a literal value with the contents of the Accumulator

**Usage:** [ANDLA] [Value to AND with the Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** AND the literal value 58 (binary 00111010) with the Accumulator

**Pre-conditions:** Accumulator = 170 (binary 10101010)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ANDLA	00001100	12
00000001	255	11111111	255

**Post-conditions:** Accumulator = 42 (binary 00101010)  
Zero Flag = 0

**Example 2:** AND the literal value 99 (binary 01100011) with the Accumulator

**Pre-conditions:** Accumulator = 156 (binary 10011100)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ANDLA	00001100	12
00000001	99	01100011	99

**Post-conditions:** Accumulator = 0 (binary 00000000)  
Zero Flag = 1

### 13. ANDRA

**Description:** AND the contents of the specified RAM location with the Accumulator

**Usage:** [ANDRA] [RAM location of value to AND with Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** AND the contents of RAM location 240 (binary 11110000) with the Accumulator

**Pre-conditions:** Accumulator = 48 (binary 00110000)  
RAM location 240 (binary 11110000) = 16 (binary 00010000)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ANDRA	00001101	13
00000001	240	11110000	240

**Post-conditions:** Accumulator = 16 (binary 00010000)  
RAM location 240 (binary 11110000) = 16 (binary 00010000)  
Zero Flag = 0

**Example 2:** AND the contents of RAM location 242 (binary 11110010) with the Accumulator

**Pre-conditions:** Accumulator = 7 (binary 00000111)  
RAM location 242 (binary 11110010) = 129 (binary 10000001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ANDRA	00001101	13
00000001	242	11110010	242

**Post-conditions:** Accumulator = 1 (binary 00000001)  
RAM location 242 (binary 11110010) = 129 (binary 10000001)  
Zero Flag = 0

## 14. ORLA

**Description:** OR a literal value with the contents of the Accumulator

**Usage:** [ORLA] [Value to OR with the Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** OR the literal value 240 (binary 11110000) with the Accumulator

**Pre-conditions:** Accumulator = 15 (binary 00001111)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ORLA	00001110	14
00000001	240	11110000	240

**Post-conditions:** Accumulator = 255 (binary 11111111)  
Zero Flag = 0

**Example 2:** OR the literal value 3 (binary 00000011) with the Accumulator

**Pre-conditions:** Accumulator = 4 (binary 00000100)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ORLA	00001110	14
00000001	3	00000011	3

**Post-conditions:** Accumulator = 7 (binary 00000111)  
Zero Flag = 0

## 15. ORRA

**Description:** OR the contents of the specified RAM location with the Accumulator

**Usage:** [ORRA] [RAM location of value to OR with Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** OR the contents of RAM location 241 (binary 11110001) with the Accumulator

**Pre-conditions:** Accumulator = 12 (binary 00001100)  
RAM location 241 (binary 11110001) = 129 (binary 10000001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ORRA	00001111	15
00000001	241	11110001	241

**Post-conditions:** Accumulator = 141 (binary 10001101)  
RAM location 241 (binary 11110001) = 129 (binary 10000001)  
Zero Flag = 0

**Example 2:** OR the contents of RAM location 249 (binary 11111001) with the Accumulator

**Pre-conditions:** Accumulator = 1 (binary 00000001)  
RAM location 249 (binary 11111001) = 9 (binary 00001001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	ORRA	00001111	15
00000001	249	11111001	249

**Post-conditions:** Accumulator = 9 (binary 00001001)  
RAM location 249 (binary 11111001) = 9 (binary 00001001)  
Zero Flag = 0

## 16. XORLA

**Description:** XOR a literal value with the contents of the Accumulator

**Usage:** [XORLA] [Value to XOR with the Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** XOR the literal value 255 (binary 11111111) with the Accumulator

**Pre-conditions:** Accumulator = 170 (binary 10101010)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	XORLA	00010000	16
00000001	255	11111111	255

**Post-conditions:** Accumulator = 85 (binary 01010101)  
Zero Flag = 0

**Example 2:** XOR the literal value 32 (binary 00100000) with the Accumulator

**Pre-conditions:** Accumulator = 32 (binary 00100000)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	XORLA	00010000	16
00000001	32	00100000	32

**Post-conditions:** Accumulator = 0 (binary 00000000)  
Zero Flag = 1

## 17. XORRA

**Description:** XOR the contents of the specified RAM location with the Accumulator

**Usage:** [XORRA] [RAM location of value to XOR with Accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** XOR the contents of RAM location 245 (binary 11110101) with the Accumulator

**Pre-conditions:** Accumulator = 15 (binary 00001111)  
RAM location 245 (binary 11110101) = 9 (binary 00001001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	XORRA	00010001	17
00000001	245	11110101	245

**Post-conditions:** Accumulator = 6 (binary 00000110)  
RAM location 245 (binary 11110101) = 9 (binary 00001001)  
Zero Flag = 0

**Example 2:** XOR the contents of RAM location 250 (binary 11111010) with the Accumulator

**Pre-conditions:** Accumulator = 21 (binary 00010101)  
RAM location 250 (binary 11111010) = 255 (binary 11111111)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	XORRA	00010001	17
00000001	250	11111010	250

**Post-conditions:** Accumulator = 234 (binary 11101010)  
RAM location 250 (binary 11111010) = 255 (binary 11111111)  
Zero Flag = 0



## 18. DECR

**Description:** Decrement the value stored within a specified RAM location by one.

**Usage:** [DECR] [RAM location of value with which to decrement]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** Decrement the value stored within RAM location 248 (binary 11111000) by one

**Pre-conditions:** RAM location 248 (binary 11111000) = 43 (binary 00101011)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	DECR	00010010	18
00000001	248	11111000	248

**Post-conditions:** RAM location 248 (binary 11111000) = 42 (binary 00101010)  
Zero Flag = 0

**Example 2:** Decrement the value stored within RAM location 241 (binary 11110001) by one

**Pre-conditions:** RAM location 241 (binary 11110001) = 1 (binary 00000001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	DECR	00010010	18
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 0 (binary 00000000)  
Zero Flag = 1

## 19. INCR

**Description:** Increment the value stored within a specified RAM location by one.

**Usage:** [INCR] [RAM location of value with which to increment]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** Increment the value stored within RAM location 248 (binary 11111000) by one

**Pre-conditions:** RAM location 248 (binary 11111000) = 255 (binary 11111111)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	INCR	00010011	19
00000001	248	11111000	248

**Post-conditions:** RAM location 248 (binary 11111000) = 0 (binary 00000000)  
Zero Flag = 1

**Example 2:** Increment the value stored within RAM location 241 (binary 11110001) by one

**Pre-conditions:** RAM location 241 (binary 11110001) = 1 (binary 00000001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	INCR	00010011	19
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 2 (binary 00000010)  
Zero Flag = 0

## 20. DECRJZ

**Description:** Decrement the value stored within a specified RAM location by one then check the result. If the result IS zero – skip the next two lines of code. If the answer is NOT zero – continue with the next line of code.

**Usage:** [DECRJZ] [RAM location of value with which to decrement]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** Decrement the value stored within RAM location 250 (binary 11111010) by one then check the result.

**Pre-conditions:** RAM location 250 (binary 11111010) = 5 (binary 00000101)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	DECRJZ	00010100	20
00000001	250	11111010	250
00000010	JUMP	00011100	28
00000011	0	00000000	0
00000100	HALT	00000000	0

**Post-conditions:** RAM location 250 (binary 11111010) = 4 (binary 00000100)  
Zero Flag = 0

**Notes:** Since the result of the decrement is NOT zero, the very next line of code is executed. In this case the program will JUMP back to line 0 (binary 00000000) which will execute another DECRJZ instruction.

**Example 2:** Decrement the value stored within RAM location 250 (binary 11111010) by one then check the result.

**Pre-conditions:** RAM location 250 (binary 11111010) = 1 (binary 00000001)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	DECRJZ	00010100	20
00000001	250	11111010	250
00000010	JUMP	00011100	28
00000011	0	00000000	0
00000100	HALT	00000000	0

**Post-conditions:** RAM location 250 (binary 11111010) = 0 (binary 00000000)  
Zero Flag = 1

**Notes:** Since the result of the decrement IS zero, the next two lines of code are skipped, and the HALT instruction is executed.

## 21. INCRJZ

**Description:** Increment the value stored within a specified RAM location by one then check the result. If the result IS zero – skip the next two lines of code. If the answer is NOT zero – continue with the next line of code.

**Usage:** [INCRJZ] [RAM location of value with which to increment]

**Program Bytes Used:** 2

**Status Flags Affected:** Zero Flag

**Example 1:** Increment the value stored within RAM location 252 (binary 11111100) by one then check the result.

**Pre-conditions:** RAM location 252 (binary 11111100) = 222 (binary 11011110)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	INCRJZ	00010101	21
00000001	252	11111100	252
00000010	JUMP	00011100	28
00000011	0	00000000	0
00000100	HALT	00000000	0

**Post-conditions:** RAM location 252 (binary 11111100) = 223 (binary 11011111)  
Zero Flag = 0

**Notes:** Since the result of the increment is NOT zero, the very next line of code is executed. In this case the program will JUMP back to line 0 (binary 00000000) which will execute another INCRJZ instruction.

**Example 2:** Increment the value stored within RAM location 252 (binary 11111100) by one then check the result.

**Pre-conditions:** RAM location 252 (binary 11111100) = 252 (binary 11111100)  
Zero Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	INCRJZ	00010101	21
00000001	252	11111100	252
00000010	JUMP	00011100	28
00000011	0	00000000	0
00000100	HALT	00000000	0

**Post-conditions:** RAM location 252 (binary 11111100) = 0 (binary 00000000)  
Zero Flag = 1

**Notes:** Since the result of the increment IS zero, the next two lines of code are skipped, and the HALT instruction is executed.

## 22. SHIFTRL

**Description:** Shift the data stored within a specified RAM location left by one through carry.

**Usage:** [SHIFTRL] [RAM location of data with which to shift]

**Program Bytes Used:** 2

**Status Flags Affected:** Carry Flag

**Example 1:** Shift the value stored within RAM location 241 (binary 11110001) left by one through carry.

**Pre-conditions:** RAM location 241 (binary 11110001) = 85 (binary 01010101)  
Carry Flag = 1

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SHIFTRL	00010110	22
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 171 (binary 10101011)  
Carry Flag = 0

**Example 2:** Shift the value stored within RAM location 241 (binary 11110001) left by one through carry.

**Pre-conditions:** RAM location 241 (binary 11110001) = 171 (binary 10101011)  
Carry Flag = 0

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SHIFTRL	00010110	22
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 86 (binary 01010110)  
Carry Flag = 1

## 23. SHIFTRR

**Description:** Shift the data stored within a specified RAM location right by one through carry.

**Usage:** [SHIFTRR] [RAM location of data with which to shift]

**Program Bytes Used:** 2

**Status Flags Affected:** Carry Flag

**Example 1:** Shift the value stored within RAM location 241 (binary 11110001) right by one through carry.

**Pre-conditions:** RAM location 241 (binary 11110001) = 85 (binary 01010101)  
Carry Flag = 1

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SHIFTRR	00010111	23
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 170 (binary 10101010)  
Carry Flag = 1

**Example 2:** Shift the value stored within RAM location 241 (binary 11110001) right by one through carry.

**Pre-conditions:** RAM location 241 (binary 11110001) = 170 (binary 10101010)  
Carry Flag = 1

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SHIFTRR	00010111	23
00000001	241	11110001	241

**Post-conditions:** RAM location 241 (binary 11110001) = 213 (binary 11010101)  
Carry Flag = 0



## 24. CBR

**Description:** Clear a specific bit within a specified RAM location.

**Usage:** [CBR] [Specified bit with which to clear] [Specified RAM location of data]

**Program Bytes Used:** 3

**Status Flags Affected:** None

**Example 1:** Clear bit 2 (binary 00000010) in RAM location 244 (binary 11110100).

**Pre-conditions:** RAM location 244 (binary 11110100) = 15 (binary 00001111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	CBR	00011000	24
00000001	2	00000010	2
00000010	244	11110100	244

**Post-conditions:** RAM location 244 (binary 11110100) = 11 (binary 00001011)

**Example 2:** Clear bit 7 (binary 00000111) in RAM location 240 (binary 11110000).

**Pre-conditions:** RAM location 240 (binary 11110000) = 255 (binary 11111111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	CBR	00011000	24
00000001	7	00000111	7
00000010	240	11110000	240

**Post-conditions:** RAM location 240 (binary 11110000) = 127 (binary 01111111)

## 25. SBR

**Description:** Set a specific bit within a specified RAM location.

**Usage:** [SBR] [Specified bit with which to set] [Specified RAM location of data]

**Program Bytes Used:** 3

**Status Flags Affected:** None

**Example 1:** Set bit 5 (binary 00000101) in RAM location 251 (binary 11111011).

**Pre-conditions:** RAM location 251 (binary 11111011) = 0 (binary 00000000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SBR	00011001	25
00000001	5	00000101	5
00000010	251	11111011	251

**Post-conditions:** RAM location 251 (binary 11111011) = 32 (binary 00100000)

**Example 2:** Set bit 0 (binary 00000000) in RAM location 243 (binary 11110011).

**Pre-conditions:** RAM location 243 (binary 11110011) = 254 (binary 11111110)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	SBR	00011001	25
00000001	0	00000000	0
00000010	243	11110011	243

**Post-conditions:** RAM location 243 (binary 11110011) = 255 (binary 11111111)

## 26. BCRSC

**Description:** Check a specific bit within a specified RAM location. If the bit IS a ZERO – skip the next two lines of code. If the bit is NOT a ZERO – continue with the next line of code.

**Usage:** [BCRSC] [bit to check] [Specified RAM location]

**Program Bytes Used:** 3

**Status Flags Affected:** None

**Example 1:** Check bit 2 (binary 00000010) within RAM location 249 (binary 11111001).

**Pre-conditions:** RAM location 249 (binary 11111001) = 3 (binary 00000011)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	BCRSC	00011010	26
00000001	2	00000010	2
00000010	249	11111001	249
00000011	JUMP	00011100	28
00000100	16	00010000	16
00000101	HALT	00000000	0

**Post-conditions:** RAM location 249 (binary 11111001) = 3 (binary 00000011)

**Notes:** Since bit 2 within RAM location 249 IS a ZERO, the code will skip straight to the HALT instruction.

**Example 2:** Check bit 2 (binary 00000010) within RAM location 249 (binary 11111001).

**Pre-conditions:** RAM location 249 (binary 11111001) = 15 (binary 00001111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	BCRSC	00011010	26
00000001	2	00000010	2
00000010	249	11111001	249
00000011	JUMP	00011100	28
00000100	16	00010000	16
00000101	HALT	00000000	0

**Post-conditions:** RAM location 249 (binary 11111001) = 15 (binary 00001111)

**Notes:** Since bit 2 within RAM location 249 is NOT a ZERO, the code will continue with the JUMP instruction.

## 27. BCRSS

**Description:** Check a specific bit within a specified RAM location. If the bit IS a ONE – skip the next two lines of code. If the bit is NOT a ONE – continue with the next line of code.

**Usage:** [BCRSS] [bit to check] [Specified RAM location]

**Program Bytes Used:** 3

**Status Flags Affected:** None

**Example 1:** Check bit 4 (binary 00000100) within RAM location 248 (binary 11111000).

**Pre-conditions:** RAM location 248 (binary 11111000) = 31 (binary 00011111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	BCRSS	00011011	27
00000001	4	00000100	4
00000010	248	11111000	248
00000011	JUMP	00011100	28
00000100	20	00010100	20
00000101	HALT	00000000	0

**Post-conditions:** RAM location 248 (binary 11111000) = 31 (binary 00011111)

**Notes:** Since bit 4 within RAM location 248 IS a ONE, the code will skip straight to the HALT instruction.

**Example 2:** Check bit 7 (binary 00000111) within RAM location 248 (binary 11111000).

**Pre-conditions:** RAM location 248 (binary 11111000) = 31 (binary 00011111)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	BCRSS	00011011	27
00000001	7	00000111	7
00000010	248	11111000	248
00000011	JUMP	00011100	28
00000100	20	00010100	20
00000101	HALT	00000000	0

**Post-conditions:** RAM location 248 (binary 11111000) = 31 (binary 00011111)

**Notes:** Since bit 7 within RAM location 248 IS not a ONE, the code will continue with the JUMP instruction.

## 28. JUMP

**Description:** Go to a specific RAM location and continue executing instructions from there.

**Usage:** [JUMP] [Specified RAM location to jump to]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Jump to RAM location 6 (binary 00000110) and continue executing code from there.

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	JUMP	00011100	28
00000001	6	00000110	6
00000010			0
00000011			0
00000100			0
00000101			0
00000110	JUMP	00011100	28
00000111	0	00000000	0

**Notes:** This example demonstrates an infinite loop. When first run, the program will jump to line 6 (binary 00000110) which then executes another JUMP instruction, causing the program to jump back to line 0 (binary 00000000).

## 29. CALL

**Description:** This instruction is almost identical to the JUMP instruction in that it will jump to a specific RAM location to keep executing code. The difference here is that the call instruction will save the current RAM location so that we may later return to the code that used the CALL instruction later. (The RETURN or RETLA instruction is used to return to our calling code).

**Usage:** [CALL] [Specified RAM location to jump to]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Jump to RAM location 6 (binary 00000110), continue executing code from there then return to continue executing code as normal.

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	CALL	00011101	29
00000001	6	00000110	6
00000010	HALT	00000000	0
00000011			0
00000100			0
00000101			0
00000110	COPYRR	00000111	7
00000111	BUTTONREGISTER	11111101	253
00001000	DATALEDREGISTER	11111111	255
00001001	RETURN	00011111	31

**Notes:** When the program is run, the code will jump to RAM location 6 (binary 00000110) to continue executing code. It will also store the number 2 (binary 00000010) in the background since this is our return address for when a RETURN or RETLA instruction is executed. The code will continue executing from RAM location 6 (binary 00000110) and will copy the contents of the ButtonRegister to the DataLEDRegister). The code will then return to RAM location 2 (binary 00000010) to continue executing instructions which in this case – will HALT the CPU.



**Example 2:** Jump to RAM location 6 (binary 00000110), continue executing code from there then return to continue executing code as normal.

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	CALL	00011101	29
00000001	6	00000110	6
00000010	COPYAR	00000101	5
00000011	DataLEDRegister	11111111	255
00000100	HALT	00000000	0
00000101			0
00000110	RETLA	00011110	30
00000111	16	00010000	16

**Notes:** When the program is run, the code will jump to RAM location 6 (binary 00000110) to continue executing code. It will also store the number 2 (binary 00000010) in the background since this is our return address for when a RETURN or RETLA instruction is executed. The code will continue executing from RAM location 6 (binary 00000110) which in this case will immediately return with the number 16 (binary 00010000) in the accumulator. The code will then continue from line 2 (binary 00000010) where the contents of the accumulator will be copied to the DataLEDRegister. The code will then finish with the HALT instruction.

### 30. RETLA

**Description:** This instruction will return with an 8-bit number in the accumulator, to the next RAM location in sequence after a call instruction is executed.

**Usage:** [RETLa] [Value to store in the accumulator]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** See example 2 of the CALL instruction (instruction 29)

### 31. RETURN

**Description:** This instruction will return to the next RAM location in sequence after a call instruction is executed.

**Usage:** [RETURN]

**Program Bytes Used:** 1

**Status Flags Affected:** None

**Example 1:** See example 1 of the CALL instruction (instruction 29)

## 32. ADDRPC

**Description:** Add the contents of a RAM location to the program counter

**Usage:** [ADDRPC][RAM location which contains the number that will be added to the program counter]

**Program Bytes Used:** 2

**Status Flags Affected:** None

**Example 1:** Add the value stored within 'myVariable' to the program counter ('myVariable' is an alias for RAM location 240)

**Pre-conditions:** 'myVariable' = 0 (binary 00000000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00010000	ADDRPC	00100000	32
00010001	myVariable	11110000	240
00010010	RETLA	00011110	30
00010011	120	01111000	120
00010100	RETLA	00011110	30
00010101	201	11001001	201
00010110	RETLA	00011110	30
00010111	234	11101010	234

**Notes:** This part of the program starts with the program counter at 00010000 which means a call would have been executed previously to get here. When ADDRPC is executed, the program counter increments to 00010001 to get the register which contains the number to add to the program counter. myVariable contains the number 0 (binary 00000000) which results in nothing being added to the program counter, and the program counter will simply execute the next instruction in sequence as normal. i.e. the next instruction will be program counter = 00010010.

See the POV Smiley Face program for an example of where ADDRPC is used.

**Example 2:** Add the value stored within 'myVariable' to the program counter ('myVariable' is an alias for RAM location 240)

**Pre-conditions:** 'myVariable' = 4 (binary 00000100)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00010000	ADDRPC	00100000	32
00010001	myVariable	11110000	240
00010010	RETLA	00011110	30
00010011	120	01111000	120
00010100	RETLA	00011110	30
00010101	201	11001001	201
00010110	RETLA	00011110	30
00010111	234	11101010	234

**Notes:** This part of the program starts with the program counter at 00010000 which means a call would have been executed previously to get here. When ADDRPC is executed, the program counter increments to 00010001 to get the register which contains the number to add to the program counter. myVariable contains the number 4 (binary 00000100) which results in the program counter jumping ahead four spaces (to 00010101) and THEN executing the next instruction in sequence. Therefore, after the ADDRPC instruction has been run, the next line to execute is 00010110.

### 33. INITSP

**Description:** Initialises the internal stack pointer to point to the top of the internal stack. In other words - resets the internal stack pointer to zero (decimal 0) (binary 00000000).

**Usage:** [INITSP]

**Program Bytes Used:** 1

**Status Flags Affected:** None

**Example 1:** Initialise the stack pointer at the start of a program

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	INITSP	00100001	33

**Notes:** The stack pointer will now point to the top of the stack. In other words - the stack pointer will be reset to zero.

### 34. RANDA

**Description:** Generates a random number between 1 – 255 (binary 00000001 – 11111111) and places the number in the accumulator.

**Usage:** [RANDA]

**Program Bytes Used:** 1

**Status Flags Affected:** None

**Example 1:** Generate a random number and store the result in memory location 4 (binary 00000100)

**Pre-conditions:** Memory location 4 (binary 00000100) = 0 (binary 00000000)  
Accumulator = 0 (binary 00000000)

PC Binary	Instruction	Machine Code (Binary)	Machine Code (Decimal)
00000000	RANDA	00100010	34
00000001	COPYAR	00000101	5
00000010	4	00000100	4
00000011	HALT	00000000	0

**Post-conditions:** Memory location 4 (binary 00000100) = random number between 1 – 255 (binary 00000001 – 11111111)  
Accumulator = random number between 1 – 255 (binary 00000001 – 11111111)

**Notes:** Once the program has been run and halted, the memory location displayed will be memory location 4 (binary 00000100). The information displayed on the Data LED's will be the random number that was just generated.