DIGITAL MAGIC EXPOSED

Book 3 – The Simple Digital Processor Simulator

Forward

Book 1 introduces the fundamentals of digital processing by describing a simple processor. Book 2 describes an emulator providing a graphical demonstration of the processor calculating each Machine Cycle. This book describes a simulation of the processor based upon an electronic design and implementation. The simulation is provided by a freeware tool.

The tool is available from Logical Circuit on the website www.logiccircuit.org and is maintained by a group of enthusiasts. The tool is excellent and achieves good performance for a free-to-use software application. It is only for educational use. The simulator works with Version 2.21.1.10. An installation Version 2.21.1.10 is included with the Books in case of difficulties with a later version. Note that the tool is not currently available to run in Windows S mode, MAC or Linux.

The tool provides a facility for exporting drawn logic circuits so that pictures can be created for documents. All the circuits presented in these books have been produced on the tool for which Logical Circuits is acknowledged.

Since the simulator is an electronic design of the processor it is necessary to load the Machine Code resulting from the assembled instructions into the simulator memory. This can be done manually but is best achieved using the firmware creation facility available from the emulator as described in Book 2.

This Book consists of two parts:

- Part 1 – This part describes installation of the simulator and its operation.
- Part 2 – This part provides detail on the simulator electronics, including the circuits and how they work.

Part 1 – The Computer Processor Simulator

1 Introduction

The processor operates by a sequence of instructions being fetched and executed as shown in Book 1 Part 1. The procedure is driven by an electronic clock. Every transition of the clock leads to a furthering of the program execution.

Commercial processors use each clock transition to kick-off sequences of electronic signals which operate the registers and gates. Several internal processor transitions occur on each clock transition. Such processors are carefully designed such that the timings of the transitions occur accurately and in sequence. There are intermediate transitory conditions between clock transitions. This type of circuit is known as a “dynamic” processor and limits the clock frequency to a minimum as well as maximum value for the system to work. The technology is built for economy and speed.

The circuit described in this Book is for a “static” processor. That is, each clock transition performs a defined event which alters the processor state into a stable state which it can hold indefinitely. The approach leads to a more complex and slower design of the processor but allows the clock to operate at low frequencies down to zero, which means the clock can be paused. This allows the operations occurring throughout the processor for every transition to be observed in detail at leisure.

2 Logical Circuit

2.1 Installation

The Logic Circuit web-site provides a download file for installation. This is in the form of a zip file, which when unzipped contains a Microsoft installation file (LogicCircuitSetup.msi). Running the file begins the installation wizard which provides a guide through the installation process. The application does not contain a recognised certificate and the installation is accompanied by warnings regarding unknown publishers and trusted sources. Of course, users should not install untrustworthy or uninvited applications.

The simulator has been tested and works with Version 2.21.1.10. An installation Version 2.21.1.10 is included with the Books should there are problems with the current tool.

The tool is ready to use when installed. Run the file “DME Simulator Iss x.x.CircuitProject” to start the simulator.

2.2 Overview

On starting the simulator the main screen appears (Figure 1).

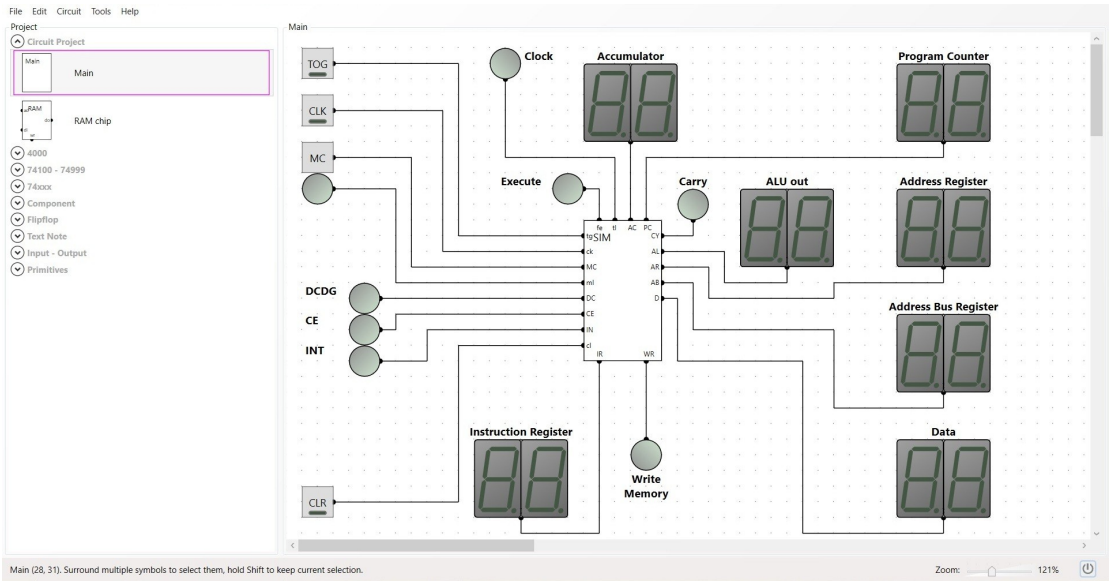


Figure 1 Simulator Main screen

The main screen shows two window panes. The main circuit and display is contained on the right. This consists of the following.

- Four switches
    - TOG – A toggle button switching the clock line to the opposite of its current state (i.e. “on” if “off” and vice versa).

    - CLK – A toggle button which switches the clock into free running mode.
    - MC – a push-button which runs the processor for one Machine Cycle when pressed with the indicator LED alight and CLR reset is released.
    - CLR – A toggle button that holds the processor in reset until pressed. Pressing again re-asserts reset.
- Seven two-digit displays
    - The content of the six main registers of the simple processor.
    - The data copied over the data bus.
- Eight LEDs
    - Clock – An indication of when the Clock pulse is “high” or “low”.
    - Execute – when lit indicates processor is in the Execute Machine Cycle state, otherwise in Reset or Fetch state. The indicator changes at the end of a cycle to show the next cycle.
    - Carry – the status of the Carry flag (LED alight means Carry is “1”).
    - Write Memory – when alight indicates a write to memory. When off the memory is in the read state.
    - MC – when alight the MC button is active.
    - Three timing indicators indicating the status of the key timing pulses used in the operation of the processor.

At the centre of the pane the circuit “SIM” contains the main processing circuits of the simple processor. This circuit is described in Part 2.

The circuit sits on a grid consisting of dots used for reference when the circuit is “powered-up”.

The simulator processor is “powered off” after starting. The processor circuit “power” is controlled by the button to the bottom right of the screen next to the circuit view zoom control. Clicking the power button toggles the power to the simulator “electronics”. This is best used with the Main circuit view (Figure 1).

The displays show the hexadecimal value of the named register or data. The appearance of each of the hex characters 0 to F is as shown in Figure 2.

Figure 2 Hexadecimal digit display formats

The screen is not locked against user modification. Therefore, accidental modification is possible when using the simulator, which could render it broken. Back-up procedures need to be applied when modifying, renaming and filing versions of the simulator.

The left pane contains all the circuits and primitives used within the electronic design. Most of these are “rolled-up” in the drop-down arrows seen below the two circuits marked “Main” and “RAM chip”. The rolled-up circuits are discussed further in the Appendix.

At any time, double-clicking “Main” navigates the simulator to the view seen in Figure 1. Double clicking “RAM chip” brings the circuit containing the RAM component provided in the tool into view. This has been kept very simple and is shown in Figure 3.

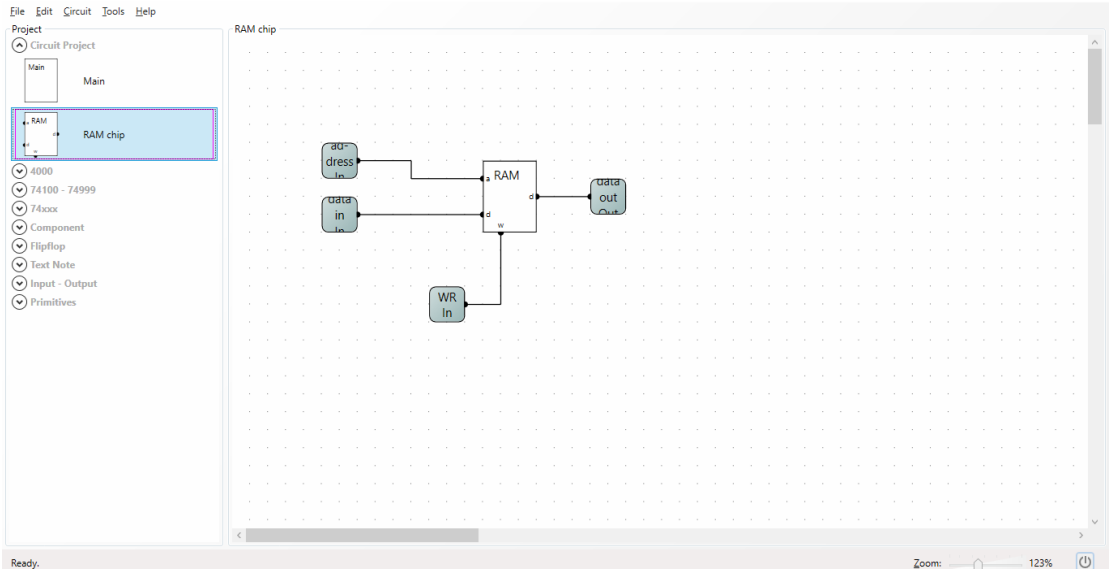


Figure 3 RAM chip screen

The purpose of the circuit and its accessibility under “Main” is to provide the easiest access to the RAM. Double-clicking the RAM within the RAM chip pane starts the RAM content view, shown in Figure 4. (There is no direct access to RAM content in the tool command bar).

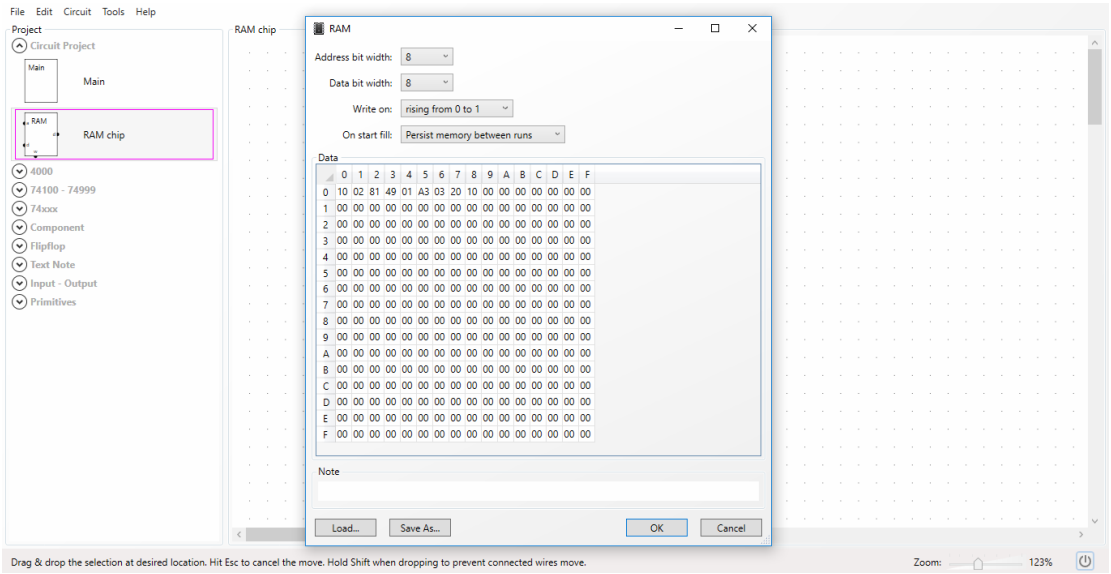


Figure 4 RAM content view

The window size may be adjusted to view all 256 bytes of RAM memory. The content can be edited directly by clicking on cells or navigating with the cursor buttons and typing new values. All entries are shown in hex and must be entered as hex.

The window also contains two buttons on the bottom left: “Load…” and “Save As…”. The buttons are used to load and save a binary image of the RAM content. The binary file produced by the Assembler in the emulator (Book 2) is compatible to the file type required by the simulator. To load the binary file click the “Load” button, select the file (e.g. “Firmware.bin”) in the relevant directory and click “Open”.

The “Save As…” button can be used to save a binary file should the RAM data be updated and a copy required. An appropriate filename and file-path need to be provided.

The four drop-down items at the top of the window set key RAM conditions and must not be modified or the simulator will not function.

The program contained in the RAM in Figure 4 is the “Simple Branch” example in Book 1 Part 1.

3 Operating the Simulator

3.1 Introduction

This section describes the operating options for the simulator along with a note on the function of the timing indicators. Following this the simulator operation is demonstrated by means of a simple program example. The program is the “Basic Operation” seen in Book1 Part 1.

3.2 Operating Modes

The simulator may be operated in any of three modes. The modes may be interchanged during program execution. The three operating modes are: Clock toggle (TOG), Free-running clock (CLK) and Machine Cycle (MC).

On circuit power-up, the CLR key holds the processor in reset until clicked. When clicked again, CLR stops the processor running and resets the Program Counter to zero. (It also resets the electronics).

As described in the Introduction (section 1) the processor is designed as a static device. This means that every state within the electronics is stable and the clock can pause indefinitely at any time. This is not how commercial processors work, which are very economical with clocking and state changing but cannot hold any individual state indefinitely. The static processor requires more clocking since a change in state for any part of the circuit must occur on a clock transition. Consequently the simple processor requires four clock cycles to complete a single Machine Cycle. (The emulator in Book 2 operates with only one clock cycle per Machine Cycle).

The TOG key provides the clock transition manually. Therefore, to complete a Machine Cycle requires eight clicks. Timing pulse changes on the LEDs are best observed in this mode. The actual clock transition is synchronised to the tool clock. If the tool clock frequency is set very low then there can be a noticeable delay between TOG clicks and the clock pulse change.

The MC key runs four clock cycles for a single click of the key. However, it is important that the pulses are applied over a complete Machine Cycle. If the processor is not at the start of a Machine Cycle then the MC key does not operate. The start of the Machine Cycle is indicated by the MC LED being lit. At any time the start of the Machine Cycle can be instated by clicking TOG repeatedly until the LED is lit. The pulses applied by MC are generated by the tool clock.

The CLK key applies the tool clock as a free-running clock to the circuit when CLR is released. This can be halted at any time by clicking CLK again.

3.3 Timing Indicators

Processor timing and control is achieved using three timing signals described as follows.

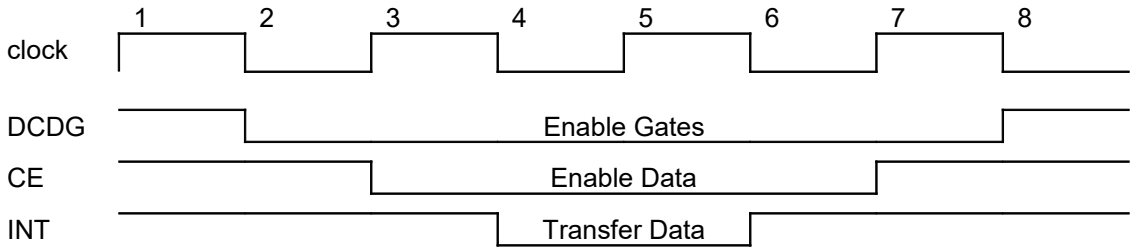


Figure 5 Timing pulses

The diagram Figure 5 shows the events occurring within the processor during the timing pulses (DCDG, CE, INT) for a single Machine Cycle. The timing pulses are generated from the clock at the transitions shown for each group of four clock pulses. The timing pulses are “active low”. This means that the state is enabled for each timing pulse type when the corresponding LED is off.

The timing ensures that the selected Gates are on and stable before memory and data are selected by the control electronics. The key operations for each clock transition are as follows.

1. Start of machine cycle. No operation.
2. DCDG active and required Gates selected.
3. CE active. Memory address set in Address Bus Register and memory enabled.
4. Selected Gates enabled and data transfer begins.
5. No operation.
6. Data fixed into destination registers and memory as required.
7. Memory deselected. Program Counter, Flags Carry and Zero are updated, as required.
8. DCDG inactive and Gates deselected indicating end of cycle.

A program can be stepped through using the TOG button and the register changes observed in relation to the timing indicated in Figure 5. Further detail on the operation of the circuits is provided in Part 2.

3.4 Basic Operation

3.4.1 Starting the Simulator

With the RAM content view open (as per Figure 4), enter the Basic Operation codes into RAM from address 00 (as Book 1 Part 1) and click “OK”. The code reads as shown in Table 1 and appears in RAM as shown in Figure 6.

Instruction Code	Memory Address	Machine Code
LDI #F0	00 01	10 F0
LDA #F0	02 03	11 F0
HLT	04	00

Table 1 Basic Operation program code

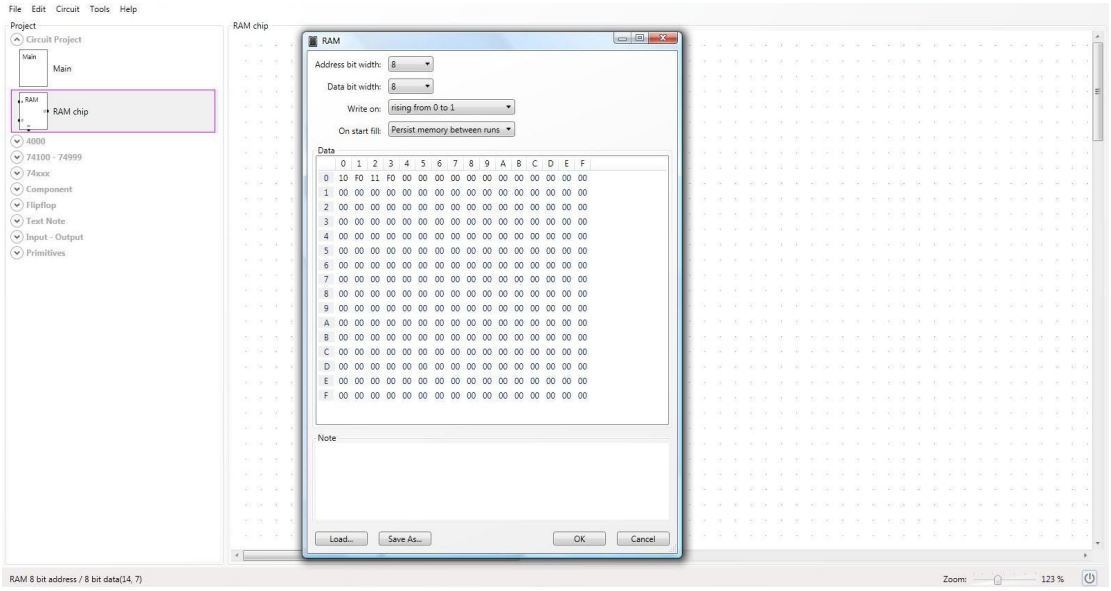


Figure 6 Basic operation program code in RAM

Return to the main screen (double-click “Main”) and apply power (i.e. click the power button at the bottom right of the screen). The display will look something like Figure 7.

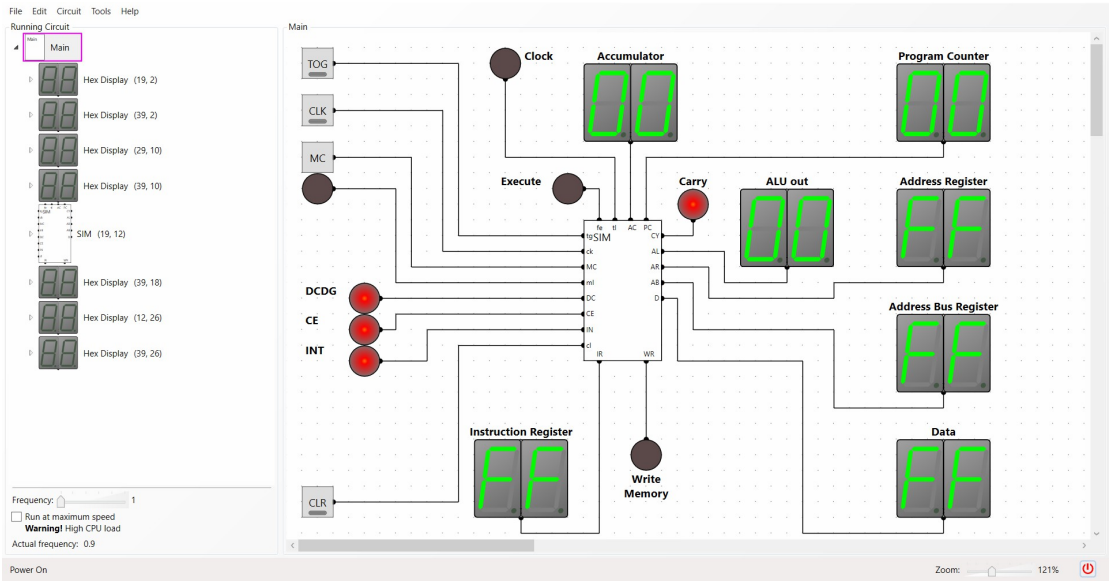


Figure 7 Main screen on power-up

The number values contained in all the registers except Accumulator and Program Counter may vary since they are not preset in the power-up and may be random. The Accumulator and Program Counter registers are reset to zero by CLR (which is holding the circuit in reset).

The Carry LED may be on or off. The other LEDs are as shown in Figure 7. The CLR key is not lit indicating the reset state and the Execute LED is off.

The left pane has altered after power-up. The circuits shown in the right pane are individually displayed. This allows each circuit to be investigated by double-clicking the individual representation in the left pane throughout the power-up phase. The circuits are identified by position using the co-ordinates next to the circuit. For example (19,2) is found on the grid 19 dots horizontally from the left and 2 dots down from the top i.e. Accumulator display.

The clock frequency setting is at the bottom of the left pane. The slider can be used to increase the frequency of the clock when free-running (CLK), the pulse rate during MC and the responsiveness of TOG. The check box will run the simulator at maximum speed. As the warning states, this will seriously load the pc cpu and may impact any other applications running (e.g. slow them down). The check box should be unchecked immediately after program completion when it is used.

3.4.2 Running the Program

Click CLR. The CLR button lights (Figure 8).

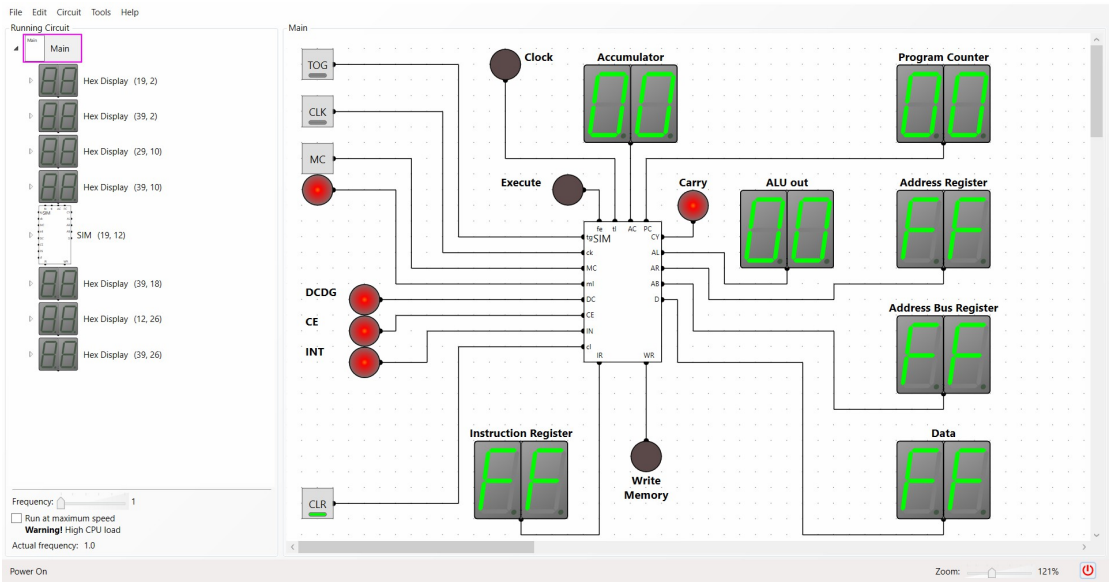


Figure 8 CLR key clicked and lights to indicate Reset is removed

The forced reset is removed and the circuit is ready to run whenever a clock is applied. The Execute LED is off and now indicates the processor is in the Fetch state. The following screens apply where the MC key is clicked each time its LED is lit. The clock frequency is set to the minimum and the Machine Cycle can be observed as the timing pulses proceed.

Click MC. The MC LED turns off.

The timing LEDs sequence through the states shown in Figure 5 and the register displays are updated. The sequence is completed when the MC LED is lit.

The first clock pulse after a CLR reset may settle the timing circuits of the simulator and be observed as the MC LED extinguishing on the first transition and re-lighting on the second. The MC key will run the first cycle correctly.

MC1 – Fetch LDI

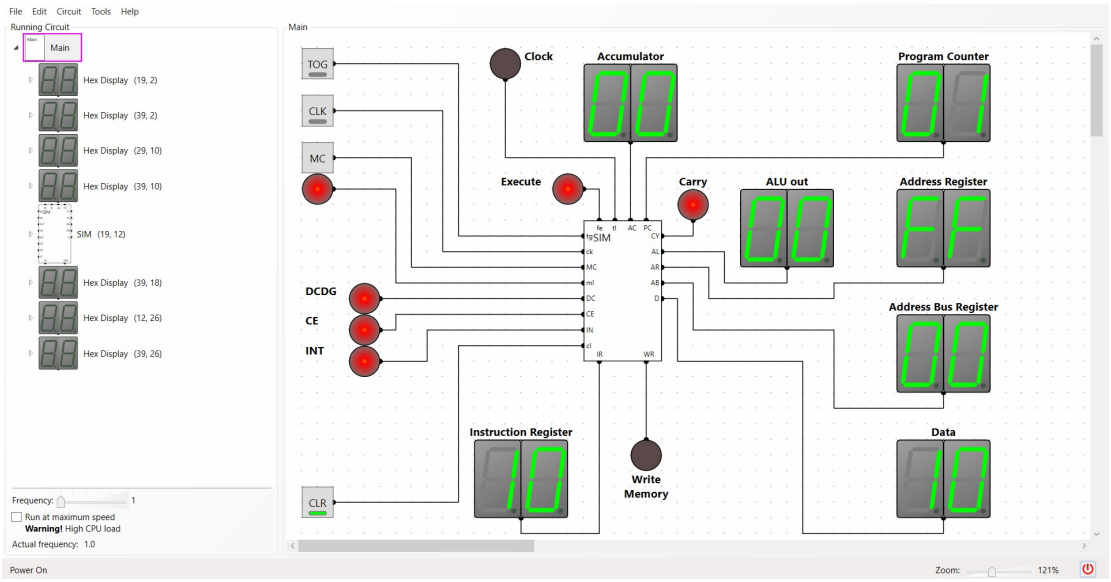


Figure 9 Fetch LDI instruction

In Figure 9:

- The fetch cycle has been processed and the LDI instruction loaded into the Instruction Register.
- The Execute LED is lit indicating the processor is now in the Execute state.
- The Address Bus Register indicates the memory byte selected (00) and the Program Counter is incremented.

Click MC.

MC2 – Execute LDI

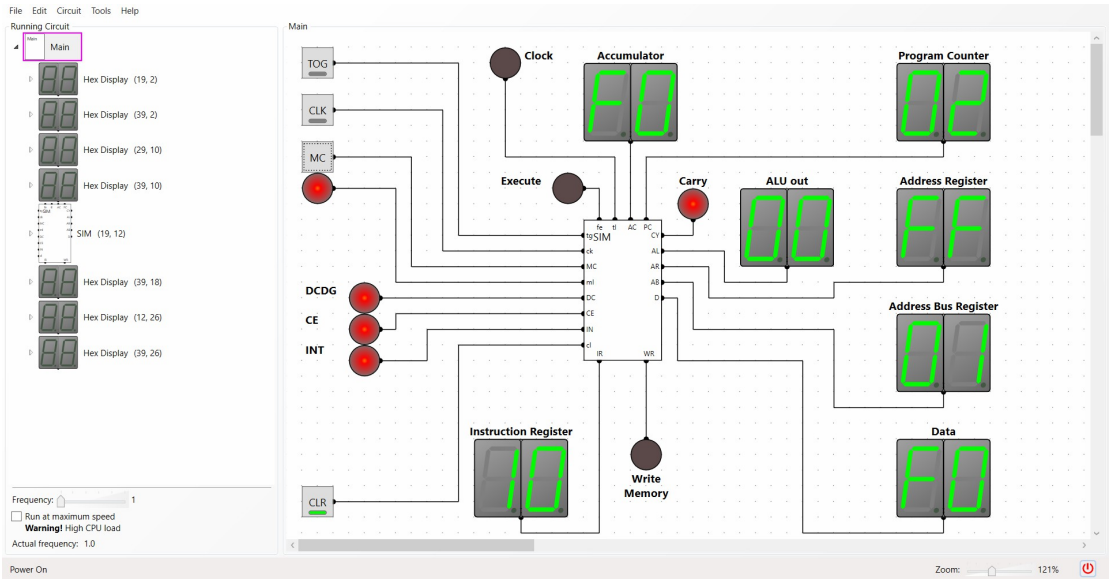


Figure 10 Execute LDI instruction

The execute cycle is complete (LED off). The Accumulator is loaded with F0 from address 01 and the Program Counter incremented.

Click MC.

MC3 – Fetch LDA

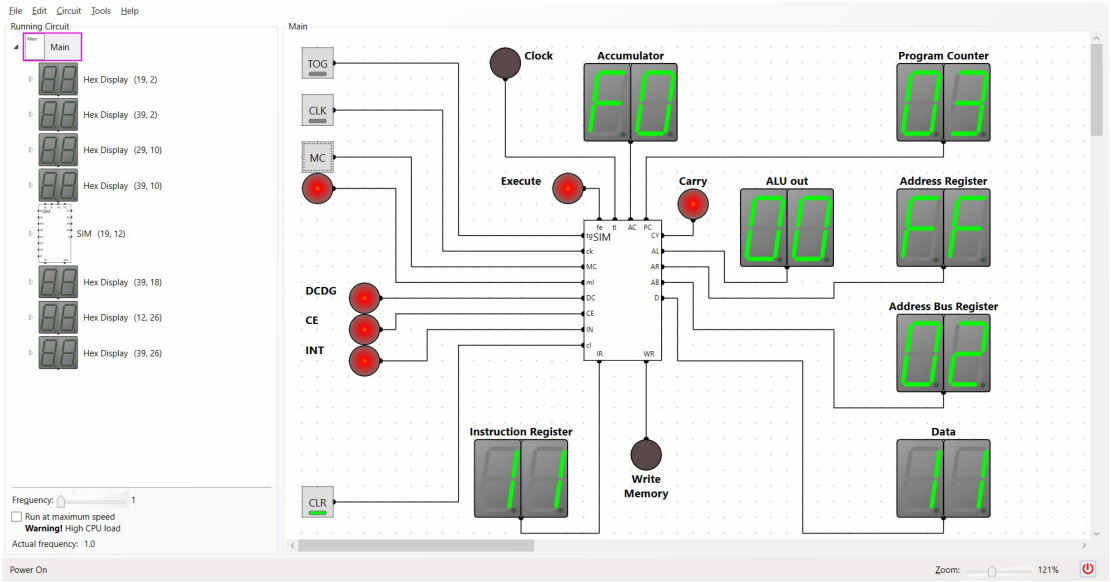


Figure 11 Fetch LDA instruction

The LDA instruction is fetched and placed into the Instruction Register. The Program Counter is incremented.

Click MC.

MC4 – Execute 1 LDA

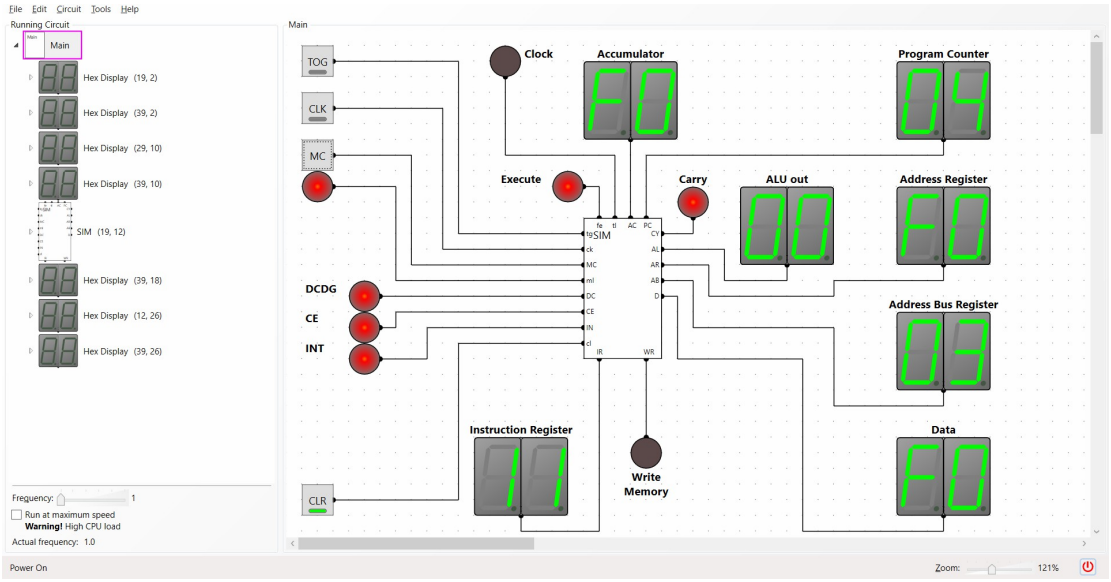


Figure 12 The first execute cycle for the LDA instruction

The operand is loaded into the Address Register and the Program Counter incremented.

Click MC.

MC5 – Execute 2 LDA

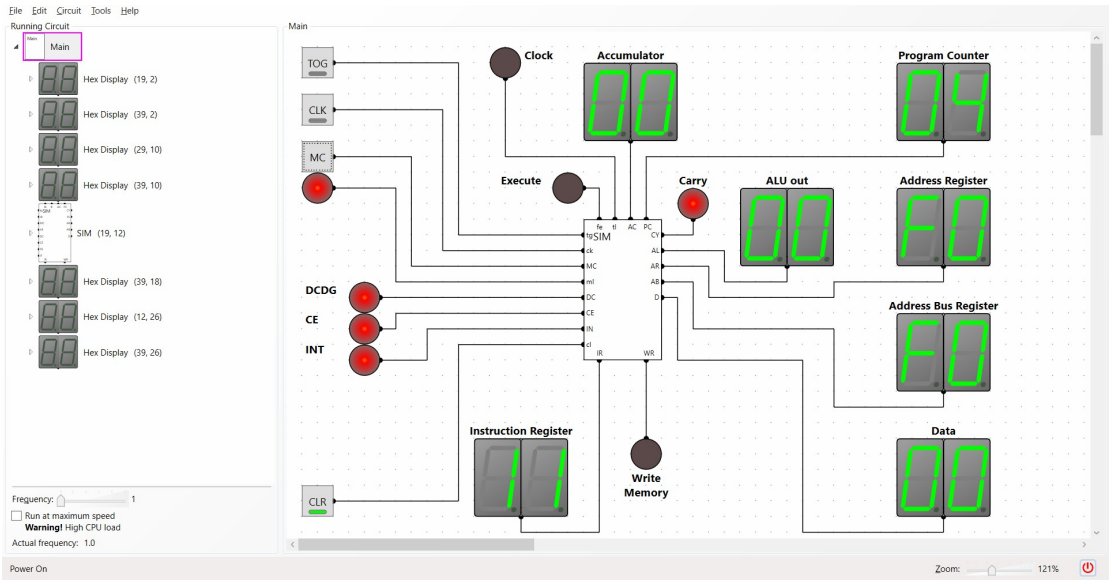


Figure 13 The second execute cycle for the LDA instruction

The data at memory address F0 is loaded into the Accumulator.

Click MC.

MC6 – HLT Halt instruction

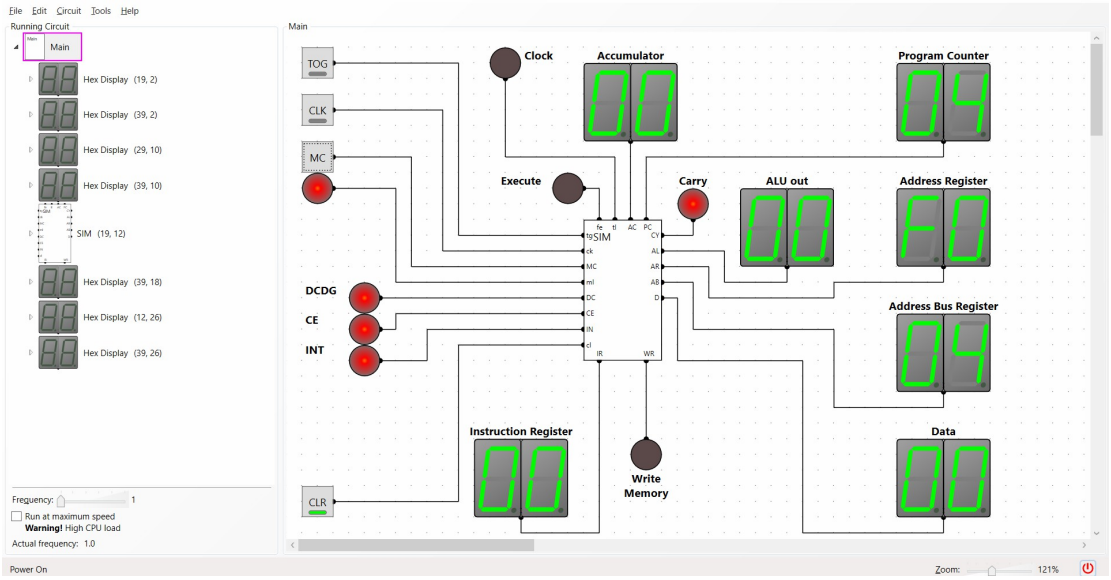


Figure 14 Halt instruction

The Instruction Register is loaded with HLT and the program stops.

Click CLR twice and click CLK. The program runs from start to HLT. Note that HLT is continually processed with no increment of the Program Counter.

CLK can be turned off at any time and the program continued with TOG or MC (when indicated by the LED).

By repeating the steps above using only the TOG key the changing states of the registers and data lines can be closely observed. See Part 2 for the operation of the circuits.

The processing speed may be increased by moving the Frequency slider for both MC and CLK modes.

3.5 Viewing Results

Programs persist in memory between runs of the simulator. However, to observe any data in memory generated by a run program it is necessary to view the RAM whilst the simulator processor is powered-on. If the circuit is powered-off before the RAM is viewed then the data is lost.

There is no short-cut to the RAM when powered-on. Each circuit must be entered until the RAM view is displayed. In the left window pane (Figure 7) double-click “SIM” followed by “CPU”; “RAM cmpnt”; “RAM” (scrolling down as required). With the RAM chip in view double-click the RAM in the right pane to open the RAM content view, as per section 2.2. Once observed, this RAM

persists and is viewable after power-off in the tool without updating the simulator program file.

The data generated does not persist when the simulator is closed. If a copy is required the RAM image may be saved, as per section 2.2, or a new version of the simulator program file saved as described in section 3.6.

3.6 Closing the Simulator

Most of the time when closing the simulator the tool requests whether a save is required. As with the emulator (Book 2) usually the answer will be “No”.

If a saved version is required (because of modification or a particular program or data contained in the RAM) then a relevant filename should be used to create another version.

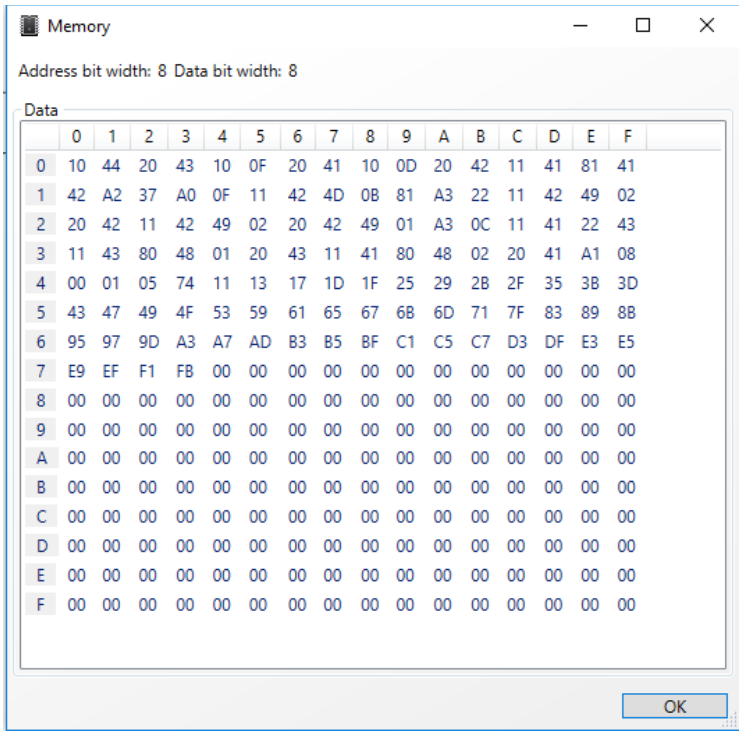
4 More Advanced Computation

This final section contains some examples of computations the simple processor can execute, given the very limited memory space. In fact it is mainly the limitation to memory which prevents the execution of far more complex programmes. The emulator includes the programs which can be used to generate binary files for the simulator.

4.1 Prime Number Generator

The program “primes” generates all prime numbers between 15 and 255 and requires many thousands of machine cycles (about 100,000) to complete.

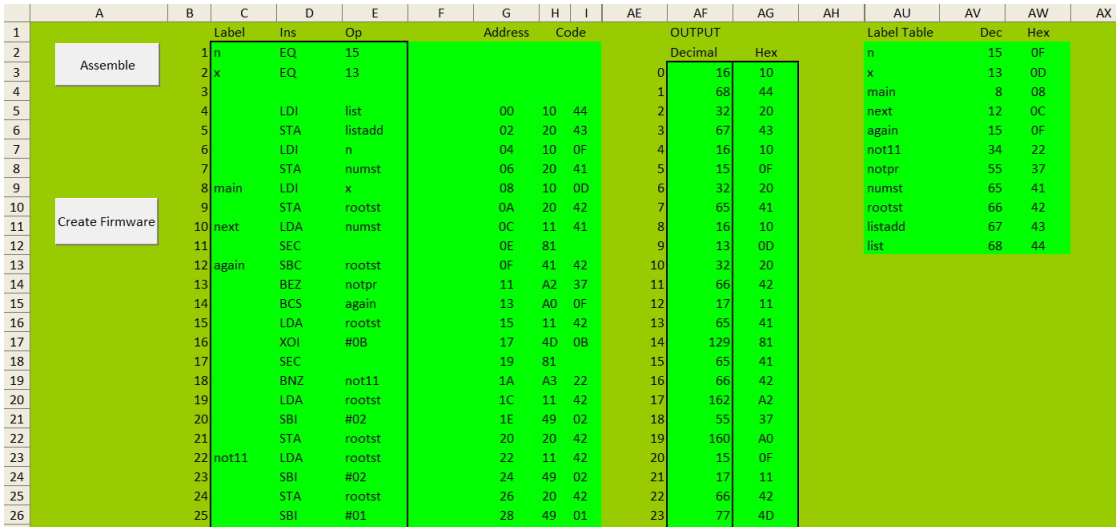
With the simulator clock set to maximum frequency (by clicking the check box) the program takes a few seconds to complete on a Windows 10 lap-top. The program completes when the Instruction Register contains “00”. Uncheck the box applying maximum speed when this occurs. The result is shown in Figure 15 with the generated numbers beginning at location 44. Data needs to be viewed during simulator power-up as per section 3.5.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	10	44	20	43	10	0F	20	41	10	0D	20	42	11	41	81	41
1	42	A2	37	A0	0F	11	42	4D	0B	81	A3	22	11	42	49	02
2	20	42	11	42	49	02	20	42	49	01	A3	0C	11	41	22	43
3	11	43	80	48	01	20	43	11	41	80	48	02	20	41	A1	08
4	00	01	05	74	11	13	17	1D	1F	25	29	2B	2F	35	3B	3D
5	43	47	49	4F	53	59	61	65	67	6B	6D	71	7F	83	89	8B
6	95	97	9D	A3	A7	AD	B3	B5	BF	C1	C5	C7	D3	DF	E3	E5
7	E9	EF	F1	FB	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 15 RAM after primes program run

Searching through the code in Figure 15 is aided through the “Address and Code” view offered by the Assembler in the emulator, partly shown in Figure 16.



	Label	Ins	Op	Address	Code			OUTPUT Decimal	Hex
1	n	EQ	15				0	16	10
2	x	EQ	13				1	68	44
3							2	32	20
4		LDI	list	00	10	44	3	67	43
5		STA	listadd	02	20	43	4	16	10
6		LDI	n	04	10	0F	5	15	0F
7		STA	numst	06	20	41	6	32	20
8	main	LDI	x	08	10	0D	7	65	41
9		STA	rootst	0A	20	42	8	16	10
10	next	LDA	numst	0C	11	41	9	13	0D
11		SEC		0E	81		10	32	20
12	again	SBC	rootst	0F	41	42	11	66	42
13		BEZ	notpr	11	A2	37	12	17	11
14		BCS	again	13	A0	0F	13	65	41
15		LDA	rootst	15	11	42	14	129	81
16		XOI	#0B	17	4D	0B	15	65	41
17		SEC		19	81		16	66	42
18		BNZ	not11	1A	A3	22	17	162	A2
19		LDA	rootst	1C	11	42	18	55	37
20		SBI	#02	1E	49	02	19	160	A0
21		STA	rootst	20	20	42	20	15	0F
22	not11	LDA	rootst	22	11	42	21	17	11
23		SBI	#02	24	49	02	22	66	42
24		STA	rootst	26	20	42	23	77	4D
25		SBI	#01	28	49	01			

Label Table	Dec	Hex
n	15	0F
x	13	0D
main	8	08
next	12	0C
again	15	0F
not11	34	22
notpr	55	37
numst	65	41
rootst	66	42
listadd	67	43
list	68	44

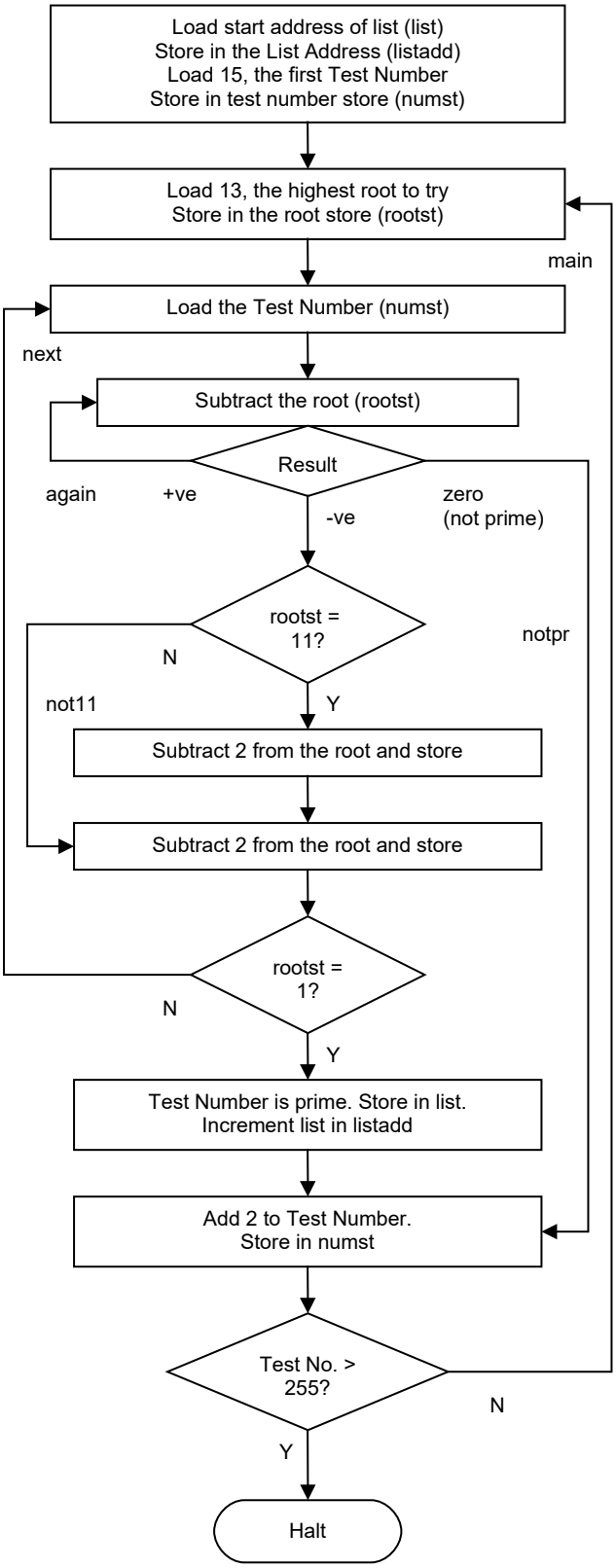
Figure 16 Partial view of Assembled primes Instruction Code in emulator

The program looks for all prime numbers between 15 and 255 (the maximum number represented by a single byte). When testing if a number is prime it is only necessary to test prime root values up to the square root of the tested number.

The algorithm tests all odd numbers from 15 to 255 by continuously subtracting the prime values between 1 and 15. That is 3, 5, 7, 11 and 13.

A flow diagram of the program and the Instructions is shown in Figure 17.

The program stores the prime numbers it finds starting at the address list (44) and each memory byte following as required.




```
n       EQ      15
x       EQ      13

        LDI     list
        STA     listadd
        LDI     n
        STA     numst
main    LDI     x
        STA     rootst
next    LDA     numst
        SEC
again   SBC     rootst
        BEZ     notpr
        BCS     again
        LDA     rootst
        XOI     #0B
        SEC
        BNZ     not11
        LDA     rootst
        SBI     #02
        STA     rootst
not11   LDA     rootst
        SBI     #02
        STA     rootst
        SBI     #01
        BNZ     next
        LDA     numst
        STX     listadd
        LDA     listadd
        CLC
        ADI     #01
        STA     listadd
notpr   LDA     numst
        CLC
        ADI     #02
        STA     numst
        BCN     main
        HLT
numst   &
rootst  &
listadd &
list    &
```

Figure 17 Algorithm and program for finding primes

4.2 Insertion Sort

The program “Insertion Sort” applies an insertion sort to a list of one-byte characters contained in memory under the label “list”. A list of bytes is defined as a sequence of non-null bytes terminated by a null byte i.e. byte-value is 00 (hex). The list is sorted in ascending order and overwrites the starting list.

The insertion sort functions by setting-up two pointers at the beginning of the list (one behind the other) and moving through the list swaps bytes where a lower-value byte is further along the list. When there is a swap the lower of the two bytes is compared to the next byte down the list and a further swap may occur. This repeats until no swap occurs or the start of the list is reached whereupon the pointers progress from the point previously reached in the list. The process completes when all bytes are checked.

Before the sorting algorithm begins the length of the list is calculated using a simple counting loop, shown in Figure 21.

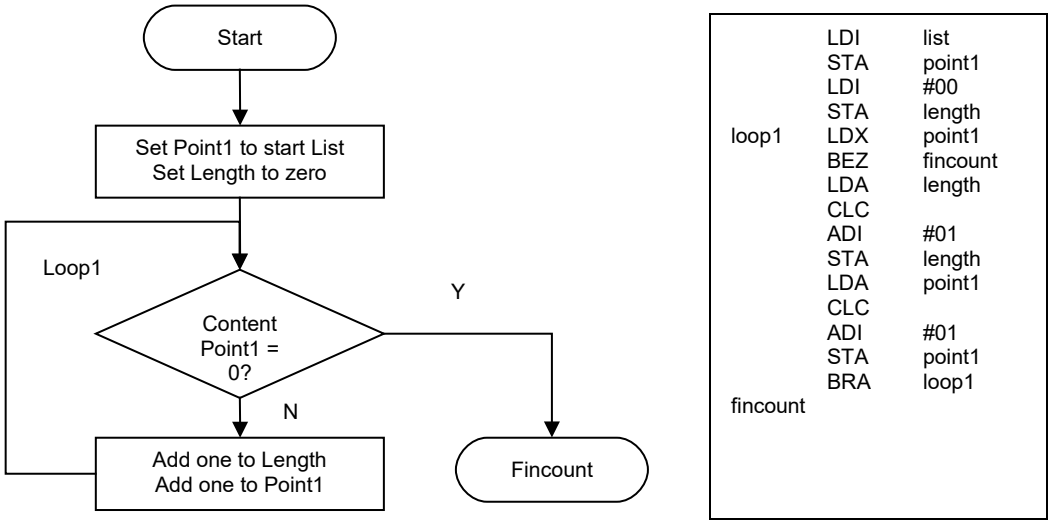
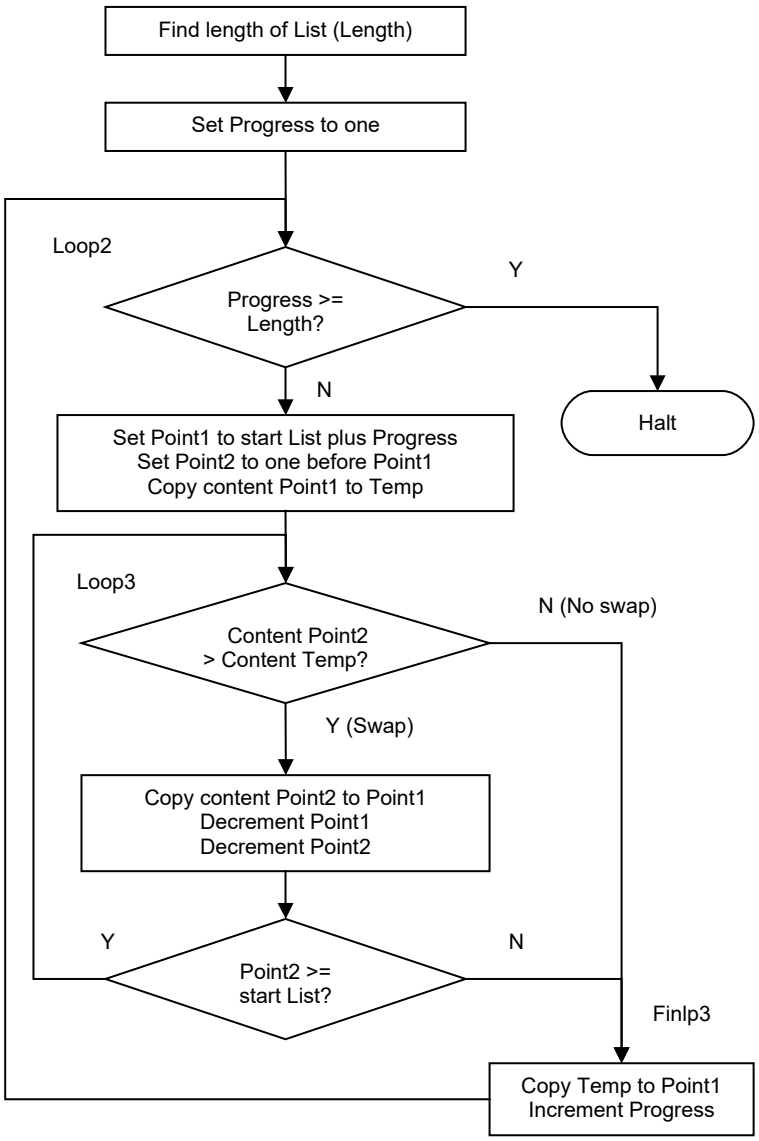


Figure 18 Calculating the length of the list

The emulator insertion sort example includes a data list for sorting. The list is generated using a random spreadsheet function. The function operates every time an event results in a spreadsheet update. Therefore, the data list constantly changes. The user can modify the cells as required for other examples of lists.

Occasionally, the list generator may calculate a null entry, which would terminate the list sooner than expected. The list is easily regenerated to remove the null or the spreadsheet formula modified to overcome this result when trying the sort program.

The sort algorithm is shown in Figure 19.




```
fincount  LDI   #01
          STA   progress
loop2     LDA   progress
          SEC
          SBC   length
          BCS   halt
          LDI   list
          CLC
          ADC   progress
          STA   point1
          SEC
          SBI   #01
          STA   point2
          LDX   point1
          STA   temp
loop3     LDX   point2
          SEC
          SBC   temp
          BCN   finlp3
          LDX   point2
          STX   point1
          LDA   point1
          SEC
          SBI   #01
          STA   point1
          LDA   point2
          SBI   #01
          STA   point2
          SBI   list
          BCS   loop3
finlp3    LDA   temp
          STX   point1
          LDA   progress
          CLC
          ADI   #01
          STA   progress
          BRA   loop2
halt      HLT
point1    &
point2    &
length    &
temp      &
progress  &

list      &
```

Figure 19 Insertion sort algorithm and program

4.3 Shell Sort

Named after its inventor in 1959 the Shellsort greatly increases the performance of a sort on very large lists. Assuming a random distribution of numbers in the list there could be large numbers to the front of the list and small numbers towards the end. The insertion sort moves numbers one space at a time as it progresses through the list. Clearly, a large number towards the front could take a long time to reach its position.

Very simply, the Shellsort performs some pre-processing on the list which tends to place larger numbers towards the end and smaller numbers towards the beginning quickly making the final insertion sort much more efficient with its moves. The algorithm is shown in Figure 20.

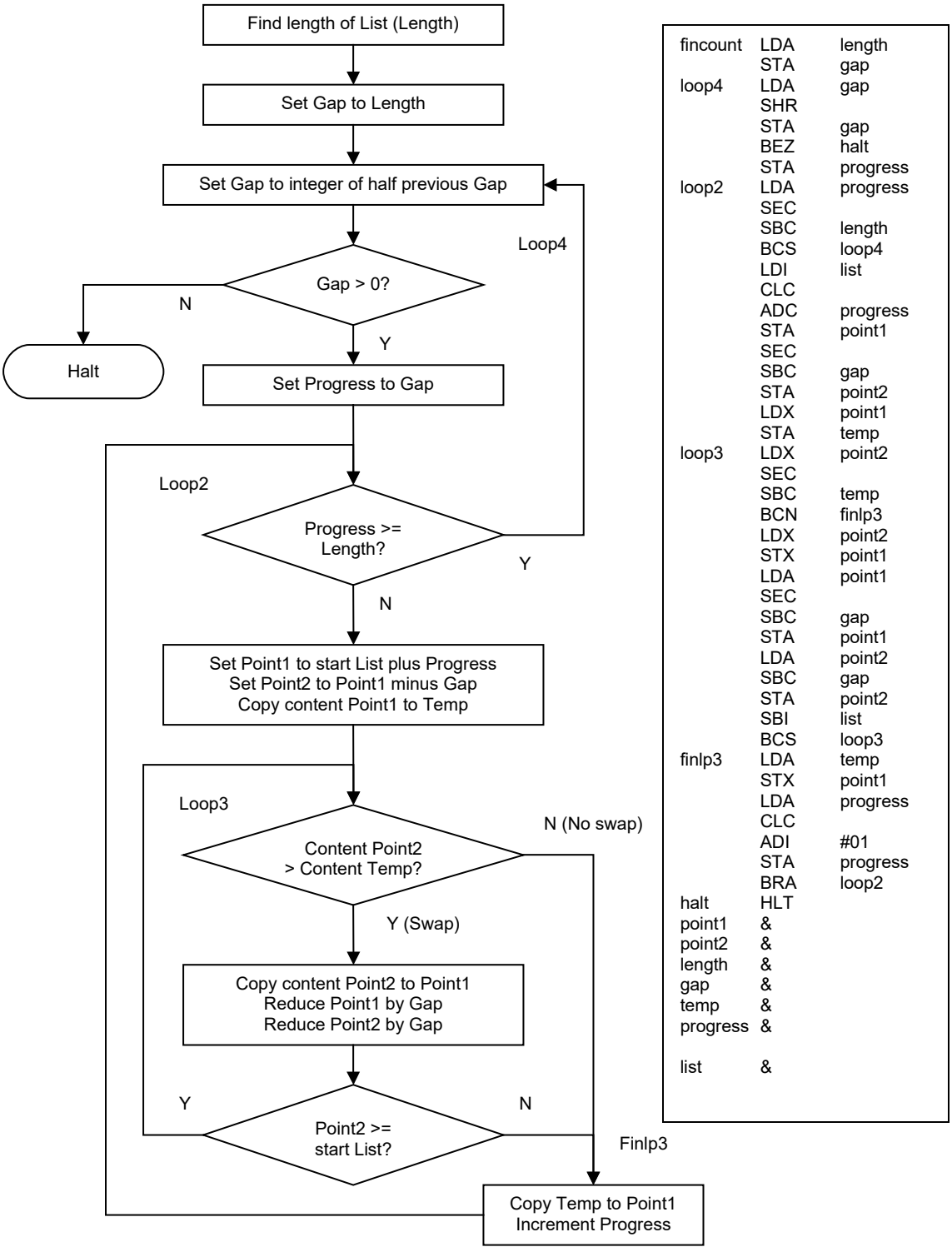


Figure 20 Shell sort algorithm and program

The list is created and counted in the same manner as the insertion sort example.

The pointers comparing bytes in the list are separated by a gap. The size of the gap is the subject of many academic arguments. The gap used here is the gap used originally by the inventor. The size of the list is divided by two and the nearest lower integer used as the gap size. A compare of bytes is made and a swap occurs if the lower byte is further along the list. The pointers move in a similar fashion to the insertion sort section 4.2.

Following a first pass over the entire list the gap is divided by two again and the whole process repeated until the gap is one. At this point, in effect, an insertion sort is performed. The gap will subsequently be halved to zero and the process completes.

The initial swaps move data very quickly to the front or back of the list making the insertion sort more efficient.

Appendix: Rolled-up Circuits

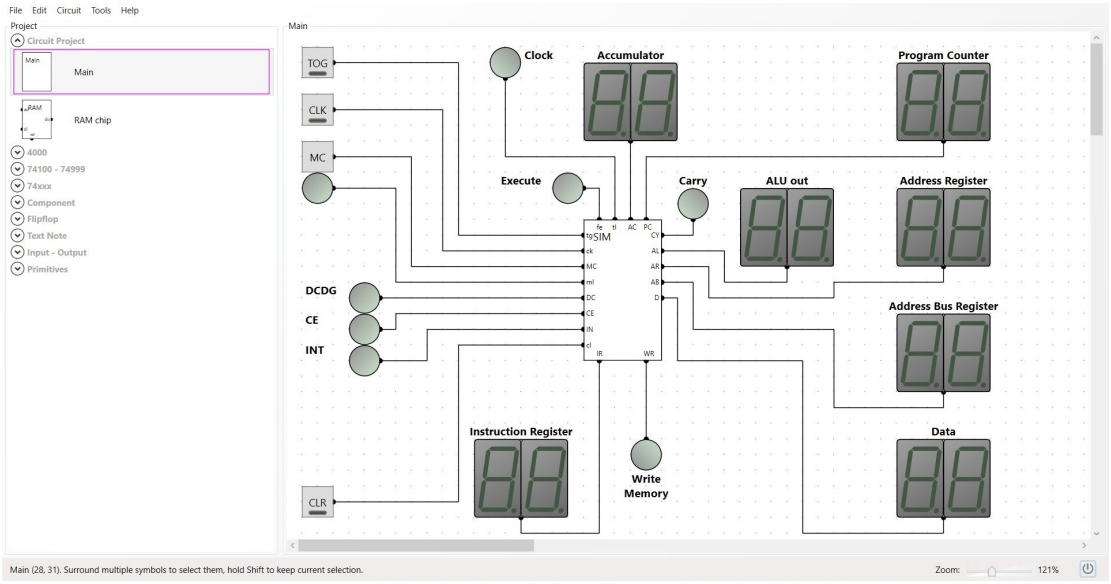


Figure 21 Simulator Main screen

The rolled-up circuits appear in the left pane of the Main screen Figure 21 below the RAM chip. Clicking the down chevron on each of the circuits reveals the outline of the circuits classified by the group name. The groups are categorised as follows.

- 4000, 74100-74999, 74xx – these are families of commercially available integrated circuits. Most of these circuits are available on the LogicCircuit web-site.
- Component – this category contains all the circuits which are effectively sub-assemblies of the Simple Processor.
- Flip-flop – the commercial latches, counters etc contain basic flip-flop assemblies defined by this category.
- Text Note, Input-Output, Primitives – the basic text, buttons, connectors and logic supplied with the LogicCircuit tool.

Any of the circuits may be selected in the left pane or by double-clicking a view of the component in the right pane. All the circuits may be decomposed down ultimately into the primitive forms as defined in the final bullet above.

The flip-flop circuits are devices used within the commercial circuits identified in the first bullet. The variety between the essential types (i.e. the D and JK flip-flops) is simply a reflection of the implementation the manufacturers of the integrated circuits chose at the time.

Ultimately, the flip-flops are the devices in the processor systems that retain the “bits” of data described in Book 1. A brief description of each is included here. The Logic Circuit tool includes a feature in the Circuit menu “Used By…” which identifies where the circuit is used.

D FF level set – the flip-flop switches the output Q to the level provided on D whenever T is high. The level on Q becomes fixed when T goes low.

D flip-flop – this flip-flop includes a reset (RS). With RS low (inactive) the Q output is set by a transition low-to-high on clk which sets Q to the level on D. A high on RS sets Q low when clk is low. A high on RS sets Qbar high.

D flip-flop with set and clear – with set and clr high (inactive) Q is set to D by a transition low-to-high on clk. Q is forced high and Qbar low when set is active (low) and Q is forced low and Qbar high when clr is active (low). (Both outputs forced high if both set and clr active).

JK active low flip-flop m/s with active high set and clear – with J and K inactive (high) clk has no effect. A high on set forces Q high and Qbar low. A high on reset forces Q low and Qbar high. (A high on both forces both outputs high). With set and reset inactive (low), an active J or K (low) is clocked by a low-to-high transition on clk. The output changes when clk changes high-to-low. For J active Q is set high and Qbar low. For K active Q is set low and Qbar high. When J and K are both active the outputs toggle between high and low on every high-to-low transition of clk.

JK flip-flop master/slave with clear – with clr active (low) Q is forced low and Qbar high regardless of other inputs. With clr inactive (high), an active J or K (high) is clocked by a low-to-high transition on clk. The output changes when clk changes high-to-low. For J active Q is set high and Qbar low. For K active Q is set low and Qbar high. When J and K are both active the outputs toggle between high and low on every high-to-low transition of clk.