A Technical Introduction To Programming The Baby Computer

Creating and Modifying Programs for the Baby Computer Using the PC-based "Baby Simulator"

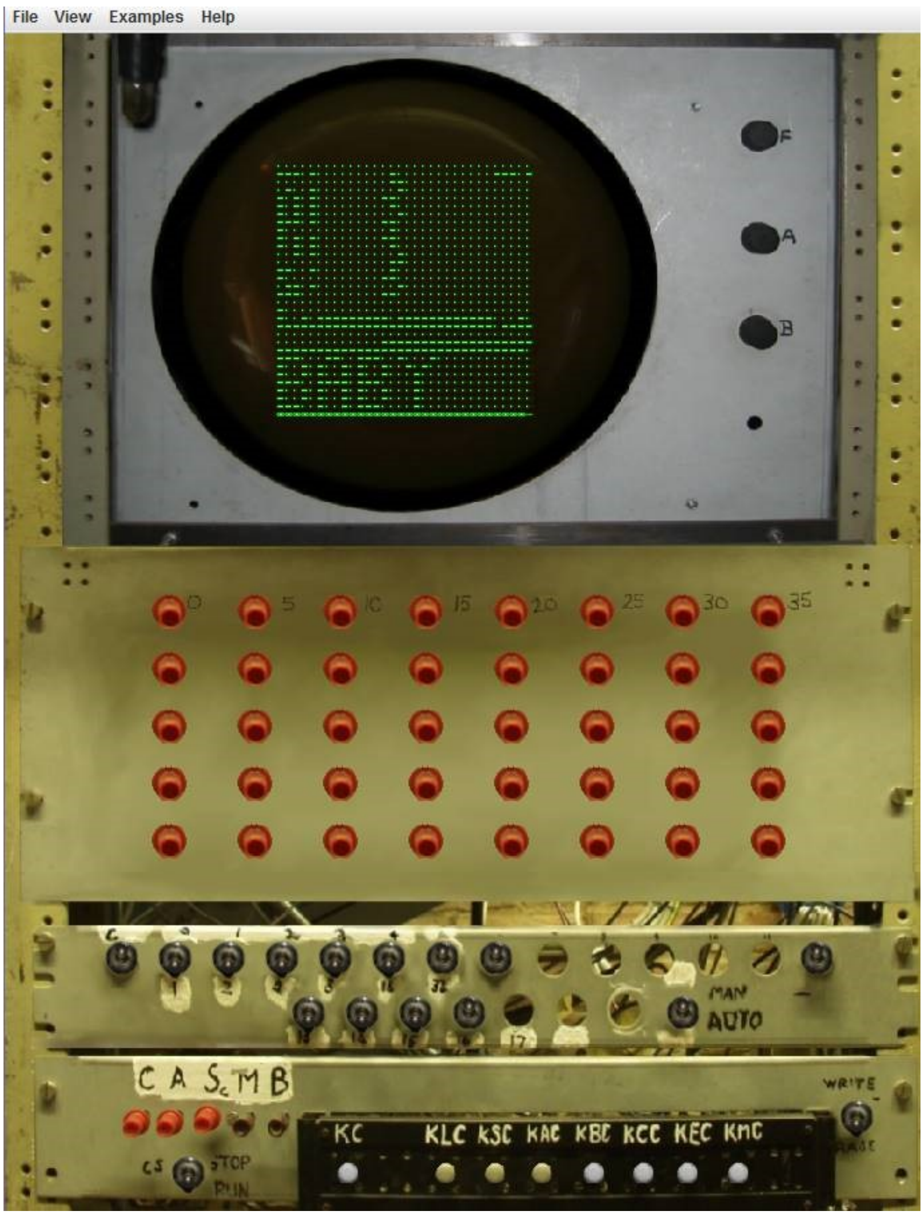


Fig. 1 - Photo of the real Baby control panel, as used in the latest PC-based Baby Simulator

Who this guide is for

This guide explains the basics of programming the Baby computer. It gives an introduction to programming concepts applicable to both the PC-based Baby simulator, and the physically recreated Baby running in MOSI. It is intended particularly for use as an aid for new members of the Baby volunteer team, and will help such colleagues to get familiar with the fundamentals of programming to a level needed to support the machine. It assumes no prior knowledge of computer programming.

Operation of Baby control panel switches in order to run the programs is also covered. Because the simulator is close in its implementation to the actual machine, much of this information is equally applicable to running programs on the latest Baby simulator, and on the actual Baby machine. Appendix B provides a reference of the control panel switches and buttons.

If you do already have programming knowledge, you will find that the main 'Reference Manual' contains extra technical detail and programming tips. (See either http://www.digital60.org/rebuild/50th/competition/ssemref.html or file **refman.jar** that is contained in the downloaded zipfile referred to in Appendix A).

For others, this guide will serve as a primer, so that you can later refer to the more detailed Reference Manual.

Setting the scene

The work of the Baby volunteer team involves the demonstration of programs running on the PC-based simulator, and also on the replica Baby itself. We also need to be able to write and run extra programs in order to debug the Baby machine when needed.

Specifically:

- We load programs into the actual Baby computer in MOSI by means of a Baby support PC. This is located behind the machine racks, and it has a direct load route into the Baby store;

- The same programs can be loaded into and run on the PC-based Baby simulator in a Web Browser or freestanding Java applet;

- The programs can be modified, or new ones can be created, using the PC simulator;

- Programs thus produced can be transferred back to the main library on the Baby support PC, and loaded from there for running on the Baby computer itself.

So, why is it useful to be able to create and modify Baby computer programs?

- Under fault conditions, it is useful to be able to run the same program in both the Baby computer and on the Baby simulator. This approach offers useful diagnostic information if divergence of behaviour occurs whilst running the same code in parallel on both the simulator and on the actual machine. In such cases, parallel running of code on the two platforms will often give an indication of which underlying functions on the Baby are exhibiting unexpected behaviour, possibly indicating specific circuitry or components for further investigations.

- It is often desirable to put a failing program into a smaller loop (for example, one with a smaller number of repeating functions) to aid the diagnostic process. Thus, modifications can be made to the code version on the simulator, prior to reloading it into the Baby computer.

The ability to run new or modified programs in this way relies on the ability to transfer code easily from the simulator into the actual Baby. This we can do, because programs from the PC simulator can be transferred to floppy disc, USB drive or CD, and these can then be used as the means of input to the Baby computer using the Baby support PC.

Familiarity with the use of the latest simulator is therefore a pre-requisite for Baby volunteers to be able to demonstrate effectively, and to be capable of debugging operational problems. To that end, the following sections will guide the reader in Baby simulator operation, and in basic programming techniques using the simulator's features.

Introducing the Baby simulator

A Java based Baby simulator has been written that runs on Windows based PCs, both in a web browser, and as a freestanding Java program. It is a very close representation of both the 1948 'Small Scale Experimental Machine' (the original 'Baby'), and of the working replica in MOSI.

It is interesting to note that the Baby simulator described in detail in the rest of this guide is the latest iteration of a number of simulators that have been produced over the years, and of these, it is the one that is visually and operationally closest to the actual Baby machine. It is now used daily by support team members. It was originally produced for a programming competition that was run in association with the University of Manchester at the time of the Baby's 60th Anniversary celebrations in 2008, and it re-uses underlying code that had been developed for earlier command line and Java simulators, adding photo-realistic depictions of the Baby control panel interface. It also adds easier to use programming and control interfaces, with a close simulation of the operation of the actual Baby's control panel switches.

The latest Baby simulator is a Java applet that may be run in three different configurations:

A) **Offline local running as a freestanding Java applet**

   This is useful when you want to be able to run the simulator offline locally, and without the need to use a Web browser. A disadvantage is that the vertical screen resolution requirements are higher than accessing it through a Web browser window, because the simulator window must all fit on the screen.

B) **Offline local running using a Web browser**

   This is useful when you want to be able to run the simulator offline locally, but in a Web browser. An advantage is that the vertical screen resolution requirements are a bit lower than running it freestanding, because the applet may be scrolled up and down in the Web browser window.

C) **Online running from an existing Internet or Intranet Web page**

   This is useful if you do not want to install code on the local PC, and when you are sure you will always have Internet or Intranet access to a Web page with the Java applet on it whenever you want to run the simulator. One such web page is at http://www.davidsharp.com/baby/. You can also use this setup to run the simulator remotely from a network share holding the Baby applet installation on another PC on the network by using a browser URL to access it. For example: *\\192.168.0.106\ssem\index.htm*, where *ssem* is the network share holding the Baby Java installation on the PC at address *192.168.0.106*.

Depending on your PC, you may find that one setup works better on your device than another. For example, when running offline locally, the simulator may run better in a local Web browser configuration than running the Java applet in freestanding mode, or vice versa.

The installation setup for each of the three configuration choices is summarised in Appendix A.

Earlier simulators are still available on the support PCs if needed by support team members. In particular, an earlier Java VM based simulator is also available that may be useful if it is found that the latest simulation will not install or run on a particular PC. Because of its photo-realistic similarity to the actual Baby machine and its operation, the use of the latest simulator is covered in detail in this guide. If required, however, Appendix C summarises how to download and set up the earlier Java VM based simulator. The operation of this is quite similar to that described in this guide, although the exact layout of the screens, and the behaviour of controls is a bit different.

Installing the simulator

To download and install the latest Baby simulator, see Appendix A.

This is the Java applet version of the simulator that lets you work with it offline, without needing to access a public website, and without requiring the use of a browser.

With reference to Appendix A, for the purposes of running these example programs on your own PC, it is therefore suggested that you install it in the first instance in the configuration for "*Offline local running as a freestanding Java applet*".

First steps using the simulator

Before we start programming, we will first get a general overview of the simulator and how it relates to the actual MOSI Baby. In working through this section, you should ideally have the Baby simulator ready to run.

The simulator behaves like the original Baby, but with an additional feature: Rather than programming in binary 0's and 1's, we can also use Assembly code. This language allows us to type in meaningful 'instructions' and normal decimal numbers to program the Baby simulator, and to save programs to files that can also be used to load the same programs on the actual Baby machine. The instructions are 3 letter abbreviations, usually written in capital letters (for example, `STP` tells the computer to stop). There are 7 of these instructions used to program the Baby.

Fig. 2 - Monitor Display Selector Buttons

The Baby main memory is called the *Store*. The monitor in the Baby control panel is the circular display with green characters on it (see Fig. 1), and it shows what is happening in the Store whenever the red button labelled **Sc** at the bottom left of the control panel is pushed in (Fig. 2). The **Sc** button selection is the usual setting to see what is happening when programs are being loaded and run.

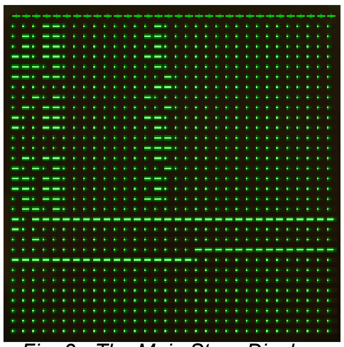


Fig. 3 - The Main Store Display

It is also possible to use the red buttons, labelled **A** and **C** (next to the **Sc** button), to change the monitor display to show two other special areas of memory in the Baby: the *Accumulator* (used to keep track of calculations currently in progress), and the *Control* (used to keep track of the current program line that is being executed).

Referring now to the picture of the Main Store in Fig. 3:

- There are 32 horizontal rows, and each row has 32 positions. In each position there is a dot or a dash. A dot represents a '0' in the memory and a dash a '1'.
- Each '0' or 1' in the memory is a *bit*, so there are 32 *bits* on each row. This group of 32 bits is a *word*, so each line on the screen is one *word*.
- The whole memory has 32 *words* each of 32 bits. The rows are numbered from 0 at the top to 31 at the bottom. The bits on a row are numbered from left to right, with bit 0 on the extreme left and bit 31 at the extreme right.
- Each *word* (a row on the screen) can hold an instruction, or a number.
- Most rows are instruction lines, and on these, bits 0 to 4 are an *address*, as represented in binary notation. This address is the number of another row in memory which will be used by that instruction to find extra information to operate on. The instruction itself is identified by settings of bits 13, 14 and 15 in the line.
- Numbers are stored in binary form, from left to right along a row. In normal numbers they can range from 2,147,483,647 to minus 2,147,483,648.
- The Baby simulator assumes (as does the actual Baby) that the first instruction line for a program is in row 1 of the store (the second one down in the display), and then assumes by default that it will read in the rows below it sequentially downwards for further instructions. This applies unless the contents of one of the instruction lines read in that way tells it to read the next instruction from a different store line than the next one down. This means that Baby program instructions will *usually* appear in the higher rows on the display (from row 1 reading down), with any numbers used and written by the program when it runs appearing in the lower rows on the display (those closer to row 31 at the bottom).

If you are already familiar with binary numbers and how we now usually write them, it is interesting to note that numbers are stored, and shown in the store, with the lowest value (least significant) bit to the left, and with the highest value (most significant) bit to the right. So, for example, the binary number corresponding to decimal 6 would be shown as 011 in the Baby store. Compare this to modern day notation, which would usually document the same number as 110 (with the binary digits reversed in order and significance).

When the simulator is first loaded it automatically loads into the memory a demonstration example program (***diffeqt.asm***), which can draw a graph on the store display.

To run the program: click on the switch, near the bottom left of the control panel, marked **STOP RUN** (Fig. 4).

Each time you click on the switch, the lever position changes from **STOP** to **RUN** (switch down), or from **RUN** to **STOP** (switch up). Click the switch to set it to **RUN**, and the program draws a graph. When it has finished the **Stop Light** at the top left comes on (Fig. 5). Click on the switch again to move it back to **STOP**.

Fig. 4 – STOP / RUN Switch

Do not click on any of the other switches at this stage. Some do nothing, but others will stop a program from working. If things do seem to be going wrong, you may have accidentally clicked on a switch. Appendix B gives their functions, and says how they should be reset for normal operation. If you find you do need to reset the positions of any switches, manually reload the same program after doing so by choosing file ***diffeqt.asm*** from the **Examples** menu (by following the procedure described below for ***Baby9.snp*** and substituting ***diffeqt.asm*** for ***Baby9.snp***), then rerun the program by clicking on the **STOP**/**RUN** switch to change its position from **STOP** to **RUN**.

Fig. 5 - Stop Neon

To run another Example program:

1. Click on **Examples** (the menu item just above the control panel) and then on ***Baby9.snp*** in the window which opens.
2. Move the **STOP/RUN** switch to **RUN**.
3. The program starts, and keeps running until you move the switch back to **STOP**.

A version of this program (***slidex.snp***) is often used on the Museum Baby to show that it is working.

In 1948, the only way to program the actual Baby was by pressing the red buttons on the control pan (these are the 5 rows of 8 red buttons, as shown in Fig. 6).

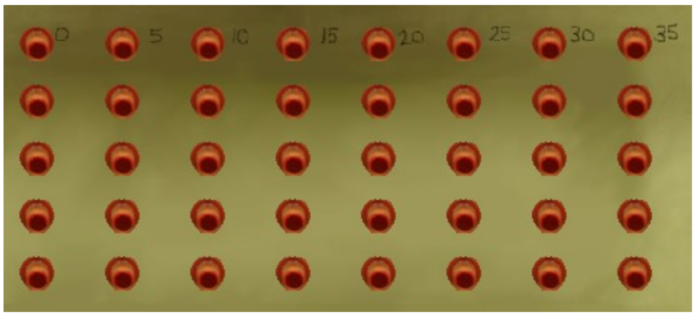


Fig. 6 – The Control Panel Typewriter Buttons

It took a while to enter a new program into store one bit at a time by pressing individual buttons: If you look closely, you will see that these 40 buttons are numbered vertically in sets of 5. By pressing a button in the range 0 to 31, it is possible to change the setting of the corresponding bit on the line of store that is currently selected (the "Action Line") once the machine switches have been set to the necessary positions to allow manual input on that store line.

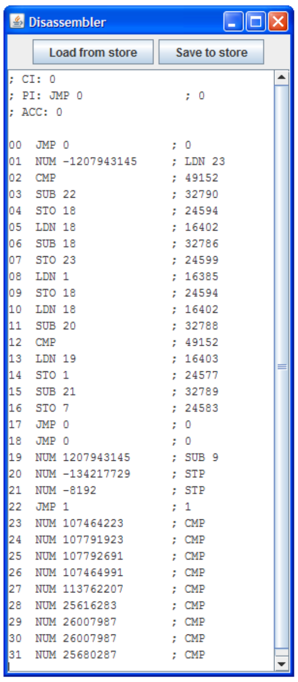
See Appendix B for fuller descriptions of the Baby switch and button functions.

Introducing the Disassembler interface

On the 1948 machine, the only way to read the result of a calculation was to look at the monitor and to manually convert from binary into normal numbers the dots and dashes on the screen on the line in store where the program had written the answer. This still has to be done now when running the actual Baby computer, but fortunately, when running the simulation, we can use the Disassembler interface. This interface – which is included in the Baby simulator itself - helps us with inputting of new programs, and also with reading the results of running programs.

To see the ***Baby9*** example program loaded earlier in its assembly code form, click on the **View** menu item (on the left just above the control panel) and then choose **Disassembler** from the drop-down.

A window opens, like the one shown in Fig. 7. This is the program in assembly code. (You may have to drag the bottom of the window down to see all 31 lines of the program. Also, it is usually easiest, if possible, to move the Disassembler window so that it does not overlap the control panel image on your display.)



```
; CI: 0
; PI: JMP 0                ; 0
; ACC: 0

00  JMP 0                 ; 0
01  NUM -1207943145       ; LDN 23
02  CMP                   ; 49152
03  SUB 22                ; 32790
04  STO 18                ; 24594
05  LDN 18                ; 16402
06  SUB 18                ; 32786
07  STO 23                ; 24599
08  LDN 1                 ; 16385
09  STO 18                ; 24594
10  LDN 18                ; 16402
11  SUB 20                ; 32788
12  CMP                   ; 49152
13  LDN 19                ; 16403
14  STO 1                 ; 24577
15  SUB 21                ; 32789
16  STO 7                 ; 24583
17  JMP 0                 ; 0
18  JMP 0                 ; 0
19  NUM 1207943145        ; SUB 9
20  NUM -134217729        ; STP
21  NUM -8192             ; STP
22  JMP 1                 ; 1
23  NUM 107464223         ; CMP
24  NUM 107791923         ; CMP
25  NUM 107792691         ; CMP
26  NUM 107464991         ; CMP
27  NUM 113762207         ; CMP
28  NUM 25616283          ; CMP
29  NUM 26007987          ; CMP
30  NUM 26007987          ; CMP
31  NUM 25680287          ; CMP
```

Fig. 7 – The Simulator's Disassembler Window

From now on, we will refer to Disassembler rows as lines:

- We can read anything after a semicolon as being a comment added by the Disassembler, and it does not directly affect the running of the program. So for now, we can completely ignore the top three lines in the display. Also, anything on the other lines in the Disassembler display after a semicolon is just documentary, so you can either ignore it or take note of the information there if it's helpful. (We will see later that the comments on the lower lines can be helpful in giving alternative interpretations of what a line of store might be representing).

- Each line starts with a number running from 0 to 31. This is the *address* of the program line in the store.

- After this, there are spaces, followed by the 3 letter Assembler code instruction. If the instruction is `NUM` (standing for "a Number"), then what follows it is not an instruction, but data for the program to use. Otherwise, the 3 letter instruction in the Disassembler indicates one of the 7 functions or instructions that the Baby is able to perform.

- Instructions that are not numbers are usually followed by a number between 0 and 31. This is the *address*, corresponding to a target line number elsewhere in Baby's store that holds any extra information that the instruction will need to complete its task.

- The Disassembler adds a semicolon and a comment to the right of it for each store line entry to indicate alternative interpretations of what the bits in that line of store *might* represent. This is particularly useful when the contents of the line may represent *either* a Number *or* a Program instruction, or (in some programs) where a line can be used as both at different times during program execution.

We will return later to more advanced features of the Disassembler interface that help in programming the Baby simulator with Assembler instructions. But first, we will get familiar with the various Assembler instructions used in programs.

Introducing the Assembler instructions

NUM	Stands For: Number

Following the 3 letter `NUM` code there is a number (data) that will be used by other instruction lines in the program.

JMP	Stands For: Indirect Jump	Baby Function Code 0 (000)

This instruction looks in a given store line that has been set up to contain a number that is *one less than* the number of the next program line to be read and executed. `JMP` causes the reading of the number held in the store line whose own address is given in the `JMP` instruction line, and it then sets the value for the Current Instruction (CI) address to be equal to that number:

New Current Instruction (CI) Address =
Number held in the store address given on the JMP instruction line

Remember that it does not jump to execute a program line at the store address given in the JMP instruction line.

Here is the reason for jumping to an address one further back than that containing the next instruction:

It must be one less than the next instruction line address because the Current Instruction line counter value is always automatically incremented by the Baby just before it reads in the next instruction from the store. So when the next instruction is about to be read, the first action is to add one to the Current Instruction address that applied when the last instruction (in this case, the `JMP`) finished its execution, and then to read the next instruction in from there. This also explains why we start all programs on line 1 not line 0: the computer starts running with Current Instruction set at 0, but it immediately adds one to it before it reads in the first instruction from store line 1.

Example:

Suppose on line 1 of a program there is:

```
01 JMP 25
```

And, on lines 2, 3, 4, 5 and 25 are:

```
02 STO 31
03 STP
04 LDN 6
05 STP
25 NUM 3
```

The effect is as follows:

1. The instruction on line 1 of the program increments the program counter to be 3.
2. Baby automatically adds 1 to the value of 3, indicating that it is line 4 to be read next.
3. Instruction line 4 is read and executed (i.e. `LDN 6`).

JRP	**Stands For: Indirect Relative Jump**	**Baby Function Code 1 (100)**

This instruction looks in a given store address that holds a number for calculation of the next program line to be read and executed: `JRP` causes the reading of the number that has been set in the store line whose address is included in the `JRP` instruction line, and the addition of that number to the present Current Instruction (CI) address (which will always be the line number holding the `JRP` instruction). This gives a new value for the Current Instruction address that is *one less* than the address of the next instruction to be executed, as follows:

New Current Instruction (CI) Address =
Present Current Instruction Line Address (i.e. the one containing the JRP instruction) +
Number in the store address given on the JRP instruction line

Remember again that it does not jump directly to the address given in the JRP instruction line.

Example:

Suppose on line 1 of a program there is:

```
01 JRP 25
```

And, on lines 2, 3, 4, 5 and 25 are:

```
02 STO 31
03 STP
04 LDN 6
05 STP
25 NUM 2
```

The effect is as follows:

1. The instruction on line 1 of the program increments the program counter to be 1 (the current program line number) plus the number held in line 25 (i.e. 1+2 = 3).
2. Baby automatically adds 1 to the value of 3, indicating that it is line 4 to be read next.
3. Instruction line 4 is read and executed (i.e. `LDN 6`).

LDN	**Stands For: Load Negative**	**Baby Function Code 2 (010)**

This instruction uses a part of the computer called the Accumulator. The Accumulator is an extra memory location which can temporarily hold a single number to be used during an instruction. The `LDN` instruction causes a number value that is held in a particular line of the store to be read and negated, and the resulting number to be written into the Accumulator, replacing what was there before.

Example:

Suppose on line 1 of a program there is:

```
01 LDN 25
```

And, on line 25, is:

```
25 NUM 123
```

The effect of this is as follows:

1. The instruction on line 1 of the program loads the number held on line 25 into the Accumulator.

   Note that it does not copy the number 25 itself into the *Accumulator*: Instead, 25 is the program (store line) address at which to find the number to put into the Accumulator. Also, note that “Load” means “copy” the value in the program (store) line being referenced into the Accumulator, not “move”, so the number 123 is still present on line 25 after the instruction is completed.

2. As the number is copied it into the *Accumulator* its sign is changed: Positive numbers in Store become negative in the Accumulator, and negative numbers become positive. This is why the instruction is called Load Negative. There will then be -123 stored in the Accumulator, and 123 is still stored on line 25.

Example:

The instructions

```
02 LDN 26
26 NUM -8
```

will load a value of 8 into the Accumulator, and -8 will remain stored on line 26.

STO	**Stands For: Store**	**Baby Function Code 3 (110)**

This instruction causes the value currently held in the Accumulator to be copied and written into a specified line in the Store.

Example:

Suppose line 8 of a program is:

```
08 STO 27
```

The effect will be to use 27 as an address and store (i.e. copy) the number in the Accumulator to line 27 of the program. The number replaces whatever was on line 27, and the number in the Accumulator remains unchanged.

Example:

If the Accumulator holds 1234, and on line 6 of a program there is:

```
06 STO 21
```

The instruction will replace whatever was on store line 21 with the value corresponding to 1234. There will still be 1234 in the Accumulator.

SUB	**Stands For: Subtract**	**Baby Function Code 4 (001)**

This instruction causes the value currently held in the Accumulator to have subtracted from it the value that is held in the specified line in the main Store, with the result of the calculation being stored back in the Accumulator. The previous value held in the Accumulator is overwritten with the resulting value.

Example:

Suppose line 4 of a program is:

```
O4 SUB 27
```

and on line 27 is:

```
27 NUM 123
```

The instruction uses 27 as a store address, containing the number to be subtracted from that in the Accumulator, and it subtracts 123 (found on line 27) from the number currently in the Accumulator, leaving the number resulting from the calculation in the Accumulator.

STP	**Stands For: Stop**	**Baby Function Code 7 (111)**

This stops the program, so it will be the last instruction to be obeyed in the program (though not necessarily the last line of the program as displayed). The **Stop Light** comes on when it is reached.

CMP	**Stands For: Compare**	**Baby Function Code 6 (011)**

This causes the program to check whether the number currently stored in the Accumulator has a negative value. If so, it causes the next instruction line in the program to be skipped, with the one on the line below that executed next instead.

Example:

Suppose the program has the following lines in it:

```
03    CMP
04    STP
05    LDN 25
25    NUM 987
```

a) *Assuming the Accumulator contains a value of 0:*

When line 3 is executed, the `CMP` instruction causes a check of whether the Accumulator is negative (which it is not), and so the program next executes line 4 of the program (Stop), causing it to end, and the Stop light to come on.

b) *Assuming the Accumulator contains a value of -1*:

When line 3 is executed, the `CMP` instruction causes a check of whether the Accumulator is negative (which it then is), and so the program next executes line 5 of the program, causing the value of -987 to be loaded into the Accumulator store.

Getting more familiar with the Disassembler

When you load an existing program into the Disassembler window (either from the store, or from a file):

- As we have noted, the Disassembler makes a “best guess” of whether the bit settings in a given line of store are more likely to be a command (instruction), or a number, and then puts a likely interpretation at the start of the line, after the number of the store line. It also adds any alternative interpretation to the right of the line, including it after a semi-colon.

- If there is *only* a number to the right of the semicolon in a line, then that is the decimal conversion of the binary value of the bits in that line (just like a number that a `NUM` command would provide).

- The Disassembler usually assumes the line contents to most likely be a command or an instruction line rather than a `NUM` number statement if any of the bits in positions 13, 14 and 15 are set to be 1.

  For example: For a line holding the number -1 (a case when all 32 bits in the line are set), it indicates the possibility that it might be a `STOP` instruction line instead of a number line by adding “`; STP`” at the end of the display for that line. (In that case, the line contents *might* mean `STOP` because the bits 13-15 are all set, just as they would also be in a `STOP` command line):

```
NUM -1      ; STP
```

- If the most likely interpretation instead was that the bits represented a command (an instruction), then the Disassembler indicates the possible alternative decimal numeric interpretation of the bit settings in the form of a comment to the right of the semicolon in the display.

For example:

```
STO 2       ; 24578
```

- The Disassembler assumes that a line where the bit settings in a store line can convert to a decimal number in the range 0 to 31 most probably contains a `JMP` instruction line instead of a number.

  This seems to be a likely interpretation to the Disassembler because in that case bits 13-15 in the line are zeroes (just as they would be for a `JMP` instruction) and the bit settings in positions 0-4 of the line also form a valid target line number (an address in store) that a `JMP` instruction could use. It therefore puts the interpretation that the line is a `JMP` command to the left of the display, but also indicates an alternative possibility that it is a `NUM` type number statement to the right of the semicolon.

  For example:

  ```
  JMP 4       ; 4
  ```

- The Disassembler window can be refreshed at any time by clicking on the **Load from store** button at the top. This causes it to be synchronised with the bit values in the store lines currently in the main Baby simulator window. (Note also that if you redisplay the Disassembler window using the Disassembler menu option from the View menu, it automatically causes a refresh from store of the current values.)

  This is useful in the following circumstances:

  i. When a program has been running, and it has written changes back into the store:

     To read the Disassembler interpretation of what those changed store line values represent. For example: to quickly read a binary result that has been written into a line in store by the program in the form of its decimal equivalent.

  ii. Whilst a program is running, stopped, or whilst it is being manually stepped through:

     To quickly check the values currently held in the CI & PI (Control) and A (Accumulator) working stores at that particular point in the execution of the program. On loading the Disassembler window, and on later performing a **Load from store**, the current values in these stores are listed in the top three lines of the Disassembler display, as shown in the following example:

     ```
     ; CI: 5
     ; PI: JMP 0 ; 0
     ; ACC: -1
     ```

  iii. When preparing to type a fresh program into the Disassembler window:

     To quickly set all the Disassembler lines to default zero (`JMP 0`) values before starting to type. This can be done by clearing the store (by flicking the **KSc** switch in the main simulator window) just before clicking on the **Load from store** button.

- The Disassembler can be used to update all lines in the main store of the simulator to match with the commands and values in the Disassembler window by clicking on the **Save to store** button.

  This is useful in the following circumstances:

  i. When a program whose store values are held in the Disassembler window has just been running and it has written changes back into the store lines:

     In order to quickly reset the store values back to the initial settings ready to start another run.

  ii. When you have typed in a program using the Disassembler window:

     To load the lines typed into the Disassembler into the main Store ready for running, or so that the new program lines can be saved to a file from there for later use. (Saving is performed using the simulator's **Save Snapshot** or **Save Assembler** options from the **File** menu).

If you are typing in a set of instructions using the Disassembler, then you can type anything you like to the right of a semicolon. It will be ignored when later loading the code into the simulator's store, or when it is saved to file from the simulator. Because of this, the usefulness of typing comments into the Disassembler is limited, because the comments can't be saved. They will be lost whenever you close the window or refresh it (for example, by using the **Load from store** option).

Adding numbers together

The purpose of the original Baby design was to test using Cathode Ray Tubes as computer memory. Everything was built using the least number of parts. Because of that, no instruction to add data numbers was provided, only the instructions to subtract, and to change a number to the negative of itself. The reason is that you can always add numbers using subtraction instructions as long as you can negate the answers, but you cannot subtract without an instruction to subtract.

Below is a program to add two numbers. As there is no add instruction, it uses subtraction to get the answer.

We will add 136 to 478 (but any numbers could be used - as long as the result is smaller than 2,147,483,647!):

Program	Explanation
`01 LDN 10`	Load into the Accumulator the number found on line 10, changing its sign. The Accumulator becomes -136.
`02 SUB 11`	Subtracts the number on line 11 from the Accumulator. Subtract 478 from -136 leaving the answer, -614 in the Accumulator
`03 STO 15`	Copy the number in the Accumulator to line 15 Copy -614 onto line 15
`04 LDN 15`	Copy into the Accumulator the number on line 15 and change its sign. We now have 614 in the Accumulator.
`05 STO 15`	Copy the number in the Accumulator to line 15. *The answer (614) is now on line 15.*
`06 STP`	Stop.
`...`	
`10 NUM 136`	One of the numbers to be added
`11 NUM 478`	The other number to be added.

Preparing to type in and run a program on the simulator

At the bottom of the control panel is a row of switches called key switches. The main ones used to run programs are shown in Fig. 8. When you click on one of these in the simulator, it flips down and then returns back up.

(Appendix B has fuller details of switch functions and their locations on the control panel.)

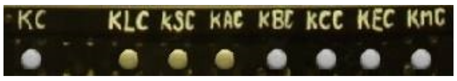


Fig. 8 - Control Panel Switches

In order to load and run a program on the simulator, standard switch settings must first be configured, as follows:

1. Check that the switch positions are at the Normal Run setting, which is:

   - **STOP/RUN** switch in the Up position (**STOP**).
   - All Line switches in the Down position.
   - All Function switches in the Down position.
   - **WRITE/ERASE** switch in the Up (**WRITE**) position.
   - **AUTO/MANUAL** switch in the Down (**AUTO**) position.

2. Click on the key switch marked **KSC**. This clears the main Store memory. The Monitor should show only dots, indicating that the memory now contains all 0's.

3. Click on **KCC**. Nothing seems to happen, but that clears both the Accumulator and Control stores and it prepares the computer to enter a new program.

4. At the top of the control panel click on **View** and then **Disassembler**. The window loads in the values currently in the main Store, and each line of the program will read "`JMP 0; 0`".

5. If the Stop Light is on now, use the **KC** switch to extinguish it, then **KCC** to reset the line counter.

An example program to add numbers

After preparing the switch positions as above, we will now type in a program to add two numbers:

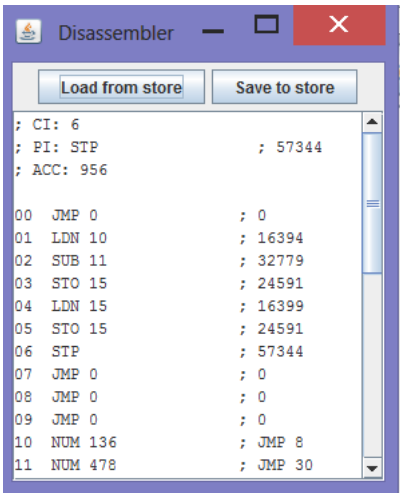
(Note that we do not put anything in on line 0. You will see why later).

1. On line 01: First erase `JMP 0` and replace it with `LDN 10`.

2. On line 02: Erase `JMP 0` and replace it with `SUB 11`.

3. Continue until all the program lines are typed in as shown to the left of the semicolons for each line (see Fig. 9). Any unused lines can be left as they were, and you can leave anything to the right of semicolons on the edited lines unchanged.

4. When you have entered the program it should look like the image on the right, except for the details to the right of the semicolons.



```
; CI: 6
; PI: STP                     ; 57344
; ACC: 956

00  JMP 0                   ; 0
01  LDN 10                  ; 16394
02  SUB 11                  ; 32779
03  STO 15                  ; 24591
04  LDN 15                  ; 16399
05  STO 15                  ; 24591
06  STP                     ; 57344
07  JMP 0                   ; 0
08  JMP 0                   ; 0
09  JMP 0                   ; 0
10  NUM 136                 ; JMP 8
11  NUM 478                 ; JMP 30
```

Fig. 9 - Example Program Lines in the Disassembler

5. Click on **Save to store** at the top of the Disassembler window. This copies the program to the Baby simulator's memory store.

6. Click on **Load from store** to load back the same values from the store into the Disassembler, this time with the comments to the right of the semicolons added automatically by the Disassembler, as is shown in the picture.

7. To save the program so that you can load it for running later: Click on **File** in the main simulator window (on the menu line right above the control panel photo in the simulator) then choose either **Save snapshot** or **Save assembly**. It does not matter which of the two file formats you save in when working with the Baby simulator, but note that any program that is to be loaded into the *actual* Baby from the support PC in the museum will need it to be saved in Snapshot format. Enter a meaningful filename of your choice, include a file extension of either "**.snp**" for snapshot, or "**.asm**" for assembly, and save your program.

8. At any time, you can reload such a saved program into the Baby simulator's store by using **File** and **Load snapshot/assembly**.

9. Now click the **STOP/RUN** switch.

10. The program should then run, and the **Stop Light** should come on. The answer will be displayed on line 15 of the monitor in binary.

    We can read the answer directly from the screen in binary and convert it manually to a decimal, but we can also use the Disassembler to quickly convert it for us, as follows:

11. At the top of the Disassembler window click on **Load from store.**

12. On line 15 in the Disassembler will be the answer to the addition.

If you do want to manually convert the result on the lines shown in binary to a decimal number, you need to remember that the least significant bit is on the left and the most significant on the right.

To perform another addition:

1. Change the numbers on lines 10 and 11 in the Disassembler window**.**
2. Copy the program to the Baby simulator memory store by clicking **Save to Store**.
3. The **STOP/RUN** switch should be at **STOP**, if not, change it to **STOP**.
4. The **Stop light** is probably still on. If so, click on the key switch **KC** (to the right of the **STOP/RUN** switch) to turn the **Stop light** off, and then click on **KCC** to reset the line counter.
5. Switch the **STOP/RUN** switch to **RUN**. The program runs until the **Stop Light** comes on.
6. See the result in the Disassembler window by clicking **Load from store**.

If you add small numbers in this way (values less than 32), you will notice that on lines 10 and 11, what you entered as `NUM` is changed to `JMP` in the Disassembler window when you click **Load from store**. This happens with numbers between 0 and 31, as we explained earlier when explaining the Disassembler. It is safest though to type `NUM` whenever typing in a number yourself using the Disassembler, as a reminder that you are not inputting an instruction line.

An example program to subtract numbers

In this example we subtract 36 from 89. Once again we will put the numbers in line 10 and 11.

Follow the process above that was used for the input and running of the addition program using the Disassembler window, and try inputting and running the program below in a similar way. Remember to clear the memory as described earlier (see "***Preparing to type in and run a program"***) before inputting this new program.

Program	**Explanation**
`01 LDN 10`	Load into the Accumulator the number in line 10 changing its sign. -89 is now in the Accumulator.
`02 STO 12`	Store the number in the Accumulator to line 12.
`03 LDN 12`	Load into the Accumulator the number on line 12 changing its sign. (89 is now in the Accumulator.)
`04 SUB 11`	Subtract the number in line 11 from the Accumulator
`05 STO 15`	Store the number in the Accumulator to line 15.
	The answer, 53, is on line 15.
`06 STP`	Stop
`...`	
`10 NUM 89`	The first number.
`11 NUM 36`	The number to be subtracted from the first number.

Reviewing the main steps to type in, load and run programs

1. Check that the switch positions are at the Normal Run setting:
   - **STOP/RUN** switch in the Up position (**STOP**).
   - All Line switches in the Down position.
   - All Function switches in the Down position.
   - **WRITE/ERASE** switch in the Up (**WRITE**) position.
   - **AUTO/MANUAL** switch in the Down (**AUTO**) position.
2. Click on key switches **KSC** and **KCC**. This clears the computer ready to receive a new program.

3. Click on **Load from store** in the Disassembler window to clear the window ready to enter a new program.
4. Type in the program.
5. Click **Save to Store** in the Disassembler window.
6. If the **Stop Light** is on use the **KC** switch to extinguish it, then use **KCC** to reset the line counter.
7. If not there already, change the **STOP/RUN** switch back to **STOP**.
8. Change the **STOP/RUN** switch to **RUN**.
9. When the **Stop Light** comes on switch the **STOP/RUN** switch to **STOP**.
10. Click on **Load from store** in the Disassembler window to read the result.

Single-stepping programs

We normally use the **STOP/RUN** switch to run straight through a program. However, it is also possible to step through a program one instruction at a time, and this is particularly helpful when we need to diagnose problems with programs, or problems with the actual Baby.

To single step a program in this way, we use the **KC** key switch.

With a program in memory, and with **STOP/RUN** set to **STOP**: Repeatedly click on the **KC** switch while watching the monitor display. You will then see the program being executed, an instruction at a time.

Normally, when a `STP` instruction is reached, the program stops and the **Stop Light** neon comes on. However, when we single step a program using the **KC** switch, we do *not* stop single stepping after hitting the Stop instruction, but carry on to read the next store line after the one with the Stop instruction. If that next line is a blank line of all zeros (which, as we will explain later, can be interpreted as a `JMP 0` instruction), the computer will then automatically jump back to read in again next the first instruction line of the program from line 1.

By pressing the **KCC** switch to reset the line counter to zero right after using **KC** to step past `STP` (so extinguishing the Stop neon), we make sure that the program counter is set to 0 ready to read the next program instruction from line 1, *even if* the line after the Stop line was *not* a blank line.

Why a jump back happens if the next line is blank will become clear by examining the Jump instruction in detail:

As explained earlier in the description of the `JMP` instruction, it allows us to move forwards or backwards to a different line of the program. It is described as an indirect jump. Again, remember that it does not jump to execute an instruction held in the address given in the `JMP` instruction line itself, but looks at the line in store given by the `JMP` line and reads the numeric value held in it. It then jumps to execute an instruction held in the line of store one higher than the numeric value that it has read.

To illustrate, suppose there are these lines in a program:

```
03 JMP 20
...
20 NUM 7
```

The computer does not jump from line 3 to line 20: Instead, when executing the `JMP` it sets the Current Instruction line counter to 7, using the number found in line 20. There is then the complication to remember (as we explained for the `JMP` instruction), that we must target the line number *one above* the one we want to run next. So for this example, we targeted line number 7 because we wanted the program to continue its execution starting from line 8, *not* line 7.

To demonstrate for yourself the effect of single stepping a program that includes `JMP` instructions, enter and run this example program:

1. Clear the memory with **KSC** and **KCC.**

2. **Load from store** to clear the **Disassembler** window.

3. Type in the program below:

Program	**Explanation**
`01 JMP 25`	Jump to execute line 5, one more than the number in line 25
`05 JMP 26`	Jump to execute line 10 one more than the number in line 26
`10 JMP 27`	Jump to execute line 15, one more than the number in line 27
`15 JMP 28`	Jump to execute line 1, one more than the number in line 28
`25 NUM 4`	
`26 NUM 9`	
`27 NUM 14`	
`28 NUM 0`	

4. Select **Save to store**.

5. Using the **KC** key switch, single step through the program. It will jump from line to line and never stop, with the highlight in the store line showing you the Current Instruction line number at the *end* of executing each of the `JMP` instructions. If you switch to **RUN** nothing seems to happen, because it jumps so quickly that it does not show on the monitor.

6. Switch back to **STOP** and use key switches **KSC** and **KCC** to clear the program from memory.

Clearing the memory sets all the bits to 0 (which you can confirm after clearing the memory by using **Load from store** in the **Disassembler** window), and the **Disassembler** window shows the cleared lines as `JMP 0` instead of `NUM 0`. The reasons for this were explained earlier. Whether a row of store that is all zeros will *actually* be used as a `JMP 0` instruction instead of a number 0 depends on whether the Baby reads that particular line in when it is looking to read an instruction to obey, rather than when it is looking for data.

So what happens when the Baby executes an instruction line that is all zeroes?

As explained in the Assembler Instructions section, a pattern of 000 in bit positions 13 to 15 of an instruction line is interpreted as a `JMP`, and when the Baby executes this line, it refers to the first 5 bits address (i.e. bits 0-4) in the same row to get the store address needing to be read that itself indicates the store location of the next program instruction.

In this case, because the address bits 0-4 at the start of the blank row are all zeros, the address part of the `JMP` instruction is also 0, and the instruction is `JMP 0`. This means that the computer looks in line 0 (the address part of the `JMP` instruction) to read a number telling it the row address of the next instruction to follow. Because that line is usually all 0's, the number it finds there is zero, and the `JMP` sets the CI line counter to 0, ready to load in and execute the next instruction from line 1 after the automatic CI increment has taken place. (Remember that `JMP` jumps back to the line number *one below* that of the next instruction line to be executed).

The overall effect is that when the program reaches a line of all 0's, and when line 0 of the program also is all 0's, the program jumps to line 0 as its Current Instruction line number after executing the `JMP` instruction, ready to load in and execute the instruction on line 1.

So how does this relate to single-stepping a program past the Stop instruction?

Remember that when the Baby stops running at a **Stop** or `STP` instruction, pressing the **KC** switch then causes it to go to on to execute the next line in store regardless. If this line is blank (which it often is right after a `STP` line), and if line 0 is blank also, then the program jumps to line 0 (i.e. with CI set to 0) ready to run the program again from line 1. So if you single step a program past the `STP` instruction, it will often jump back to start executing again the instructions from line 1.

To illustrate the effect of single stepping through a program that instead contains the `CMP` (**Compare**) instruction:

As we described previously in the description of the `CMP` assembler instruction, when the instruction `CMP` is reached the Accumulator is examined. If the number in the Accumulator is found to be negative, the program in effect jumps over the next line to execute the next line but one.

Read again the description of the `CMP` (**Compare**) instruction earlier, and then look at the following program, before inputting it, to work out where it will stop, and what number will be displayed on line 20 when it stops:

```
01 LDN 30
02 CMP
03 JMP 29
04 STO 20
05 STP
06 LDN 31
07 CMP
08 JMP 28
09 STO 20
10 STP
11 LDN 29
12 STO 20
13 STP
…
28 NUM 10
29 NUM 5
30 NUM -1
31 NUM 1
```

Remember that:

- If you single step the program it will jump over the **Stop** (`STP`) instruction, and when it reaches an empty line, it will read this as a jump to line zero and will then carry on again from the top.
- If you want the program to run and stop normally, use the **RUN** switch.

Here is what you should find happens when you run it with the **RUN** switch:

1. Line 1: Loads 1 into the Accumulator (remember that the sign changes when `LDN` is used).
2. Line 2: Test the Accumulator.
   [It is positive - so do not skip a line].
3. Line 3: Jump to the address given in line 29.
   [It references line number 5 - so the program will continue from line 6].
4. Line 6: Load -1 into the Accumulator.
5. Line 7: Test the Accumulator.
   [Accumulator is negative - so jump over the following line to the next].
6. Line 9: Store what is in the Accumulator
   [Stores -1 in line 20]
7. Line 10: Stop

So the answer is: The program places **-1** on line 20.

Other things to try:

- Try changing the sign of the numbers on lines 30 and 31 to get other results.
- Instead of using the **RUN** switch: Try single stepping through the program using the **KC** switch to compare what happens after the **Stop** line is reached, and when **KC** is then pressed again.

A look at the original 1948 Baby program

This program finds the highest factor of a number set in Store line 23, and writes the answer into Store line 27.

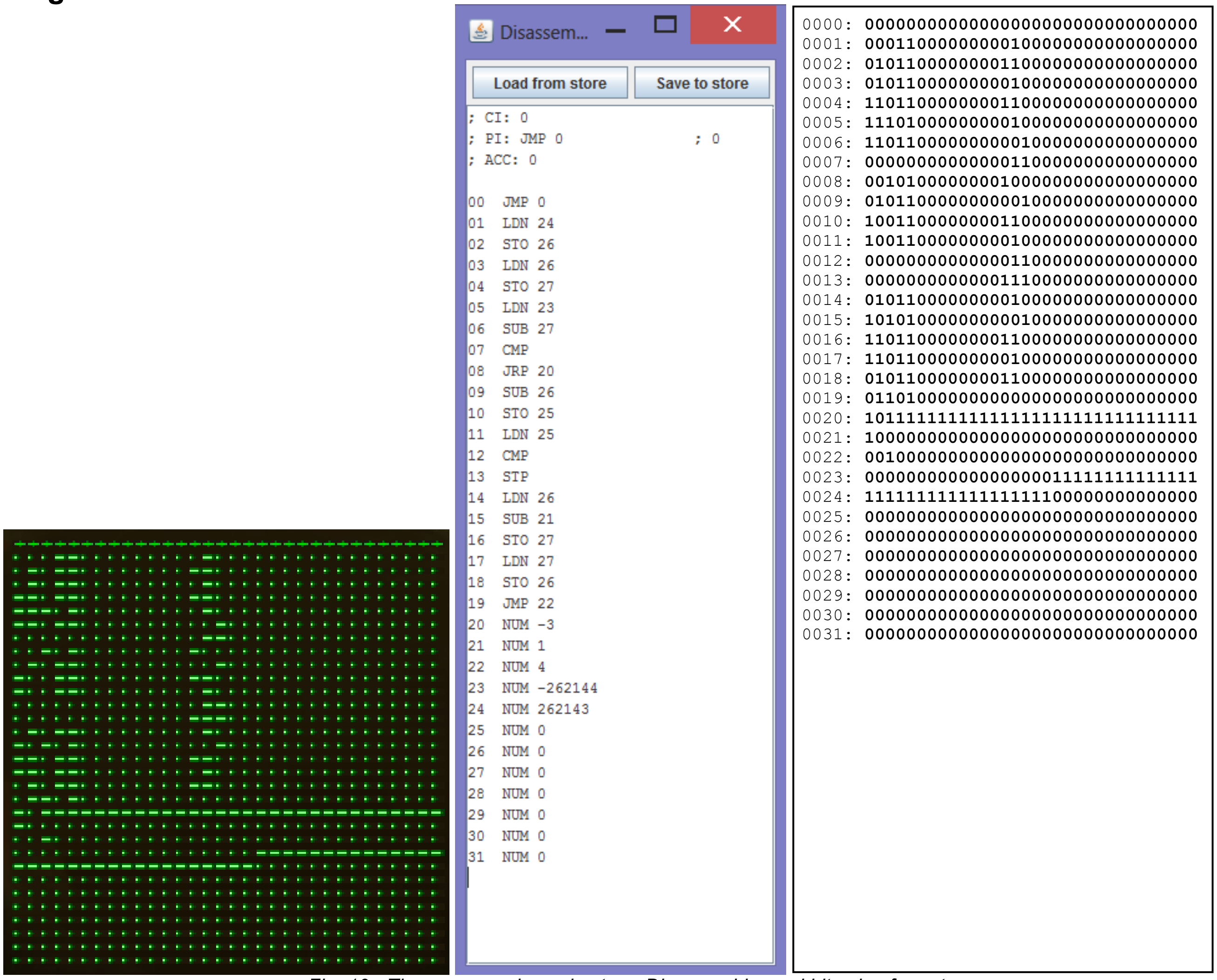
The code is based on Geoff Tootill's "Amended Version" of the original code from 18 July 1948 notes that used 262144 (= 2^18) – set up in Store line 23 – as the number to find the highest factor of. The answer is 131072 (= 2^17). The amended version tidied up variable initialisation, but did not change the coding logic of the original.

The original run took 52 minutes on June 21st 1948, using over 130,000 subtraction loops and more than 3 million CRT store accesses. Equivalent code is loadable into the Baby from Disassembler snapshot file **FACTORCT.SNP**, which also runs on the reconstructed Baby machine in around 52 minutes. There is also a faster-running version that can be loaded from file **PROG1.SNP**. This one finds the highest factor of 989 (using 989 in line 23 instead of 262144), giving the answer of 43 in around a minute. Both snapshot files are on the Baby support PCs in MOSI.

To run in the simulator:

i. Set all the switches to the Normal Run position.
ii. Load **FACTORCT.SNP** (for the original run simulation) or **PROG1.SNP** (for the shorter run demo version).
iii. If the Stop light is lit, press the **KC** switch, then **KCC**.
iv. Set the switch to **RUN**.
v. Monitor Store (line 27 holds the answer when Stop is reached).

Program Code



```
; CI: 0
; PI: JMP 0                    ; 0
; ACC: 0

00  JMP 0
01  LDN 24
02  STO 26
03  LDN 26
04  STO 27
05  LDN 23
06  SUB 27
07  CMP
08  JRP 20
09  SUB 26
10  STO 25
11  LDN 25
12  CMP
13  STP
14  LDN 26
15  SUB 21
16  STO 27
17  LDN 27
18  STO 26
19  JMP 22
20  NUM -3
21  NUM 1
22  NUM 4
23  NUM -262144
24  NUM 262143
25  NUM 0
26  NUM 0
27  NUM 0
28  NUM 0
29  NUM 0
30  NUM 0
31  NUM 0
```

```
0000: 00000000000000000000000000000000
0001: 00011000000000100000000000000000
0002: 01011000000001100000000000000000
0003: 01011000000000100000000000000000
0004: 11011000000001100000000000000000
0005: 11101000000000100000000000000000
0006: 11011000000000010000000000000000
0007: 00000000000000110000000000000000
0008: 00101000000001000000000000000000
0009: 01011000000000010000000000000000
0010: 10011000000001100000000000000000
0011: 10011000000000100000000000000000
0012: 00000000000000110000000000000000
0013: 00000000000001110000000000000000
0014: 01011000000000100000000000000000
0015: 10101000000000010000000000000000
0016: 11011000000001100000000000000000
0017: 11011000000000100000000000000000
0018: 01011000000001100000000000000000
0019: 01101000000000000000000000000000
0020: 10111111111111111111111111111111
0021: 10000000000000000000000000000000
0022: 00100000000000000000000000000000
0023: 00000000000000000011111111111111
0024: 11111111111111111100000000000000
0025: 00000000000000000000000000000000
0026: 00000000000000000000000000000000
0027: 00000000000000000000000000000000
0028: 00000000000000000000000000000000
0029: 00000000000000000000000000000000
0030: 00000000000000000000000000000000
0031: 00000000000000000000000000000000
```

Fig. 10 - The program shown in store, Disassembler and bit value formats

Note: The disassembler display in Fig. 10 has been simplified to show only relevant assembler interpretations for each line. Note also that the actual snapshot file FACTORCT.SNP also sets the bit 5 (value 32) on some of the program lines, but this bit is ignored when the program runs, so it has been omitted here for reasons of clarity.

Program Logic

Lines 1 - 4: Initialisation

```
01  LDN 24  ; Loads to Acc -(no. to be factored - 1) = initial -b test value
02  STO 26  ; Stores the initial –b test value in line 26
03  LDN 26  ; Loads initial +b value into Acc
04  STO 27  ; Stores the initial +b test value in line 27
```

Lines 6 - 8: Do subtractions using the current b test value, check sign of difference, jump back if 0 is not passed yet

```
05  LDN 23  ; Loads in no. to be factored to Acc.
06  SUB 27  ; Subtracts the latest +b test value from the current Acc value
07  CMP     ; Jumps to execute line 9 if Acc is now –ve.
08  JRP 20  ; Loops back to execute from line 6 again if Acc value not yet –ve.
```

Lines 9-13: Form a remainder, Test it and Stop if it is Zero (because we have a result then)

```
09  SUB 26  ; Subtract current -b test value from Acc (so adds +b back on).
            ; By adding +b back on, we identify if subtractions have overshot 0
            ; by less than the amount +b, in which case b isn't a factor.
            ; If instead we get back to 0 exactly, then it must be a factor.
10  STO 25  ; Stores the calculated overshoot difference value in line 25.
            ; If this is 0, we've found the factor. If it's +ve, we haven't.
11  LDN 25  ; Loads –ve of line 25 overshoot difference value.
            ; If a –ve no is loaded now, then we haven't got a factor.
            ; If a non –ve no is loaded it must be 0 and we have the factor.
12  CMP     ; If Acc is –ve: Jumps to execute from 14 with a new test divisor.
            ; If Acc is not –ve: Execute next line 13 to Stop.
13  STP     ; STOP. Acc was NOT –ve so Divisor was found. Answer is in Line 27.
```

Lines 14 – 19: Form a new divisor **b** to be tested, then jump back and test it as a possible factor using subtractions

```
14  LDN 26  ; Load the last tested b value as a positive Acc value.
15  SUB 21  ; Decrement the last tested b value by 1.
16  STO 27  ; Store new +b test value in line 27.
17  LDN 27  ; Load new –b test value into Acc.
18  STO 26  ; Store new -b test value in line 26.
19  JMP 22  ; Execute subtractions from line 5 again using new test b value.
```

Lines 20 – 24: Fixed data

```
20  NUM -3  ; Value for use in the JRP jump instruction in line 8.
21  NUM 1   ; Value for decrementing value of the test b value in line 15.
22  NUM 4   ; Value for use in the JMP jump instruction in line 19.
23  NUM -262144; Negative form of the number to be factored.
24  NUM 262143; First b value to check as being a factor of number in line 23.
```

Lines 25 – 27: Variable data written to during execution (initially all zero)

```
25  NUM 0   ; Latest overshoot difference (written by line 10).
26  NUM 0   ; Latest -b value under test (written by line 2 or 18).
27  NUM 0   ; Latest +b value under test (written by line 4 or 16).*
28  NUM 0   ; Not used.
29  NUM 0   ; Not used.
30  NUM 0   ; Not used.
31  NUM 0   ; Not used.
```

* The answer is left displayed in store line 27 at end of run.

APPENDIX A: Setting up the latest Java Baby simulator

PC System requirements

- Windows XP, Windows Vista, Windows 7, Windows 8 or Windows 8.1.
- Java SE Runtime version 6 or later installed.

  (If this is not already installed, see the section below on "*Downloading and installing the Java Runtime*").
- For running the simulator offline locally as a freestanding application:

  A Windows Desktop resolution set to a minimum of 900 pixels in height.
  It may be necessary to auto-hide the Windows Taskbar if it overlaps the simulator display.
- For running the simulator in a Web browser window (offline locally or online):

  A Web browser equivalent to Internet Explorer 7 or later;
  The browser's Zoom display setting at 100%;
  The Windows Desktop resolution at a minimum of 768 pixels in height, but recommended 800 or above;
  If necessary, run the browser in Full Screen mode (to avoid the need to scroll the applet up and down).

Installation overview

This Baby simulator is a Java applet that may be set up to run in three different configurations:

A) Offline local running as a freestanding Java applet

B) Offline local running using a Web browser

C) Online running from an existing Internet or Intranet Web page

The installation setup for each of these three configuration choices is summarised in the following sections.

A) Setup for offline local running as a freestanding Java applet

1. The Java applet itself may be downloaded from David Sharp's archive website by going to the following link: http://www.davidsharp.com/baby/ssem.zip .

2. A message similar to the one shown below will appear (the exact message will vary depending on the version of your Operating System and web browser). Choose the option to **Save as**, and store the file in a suitable folder on your PC's local drive, noting where this is.

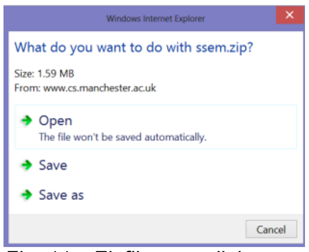
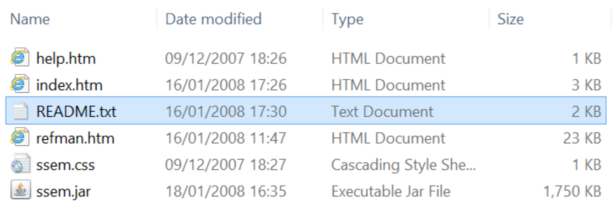

Windows Internet Explorer
What do you want to do with ssem.zip?
Size: 1.59 MB
From: www.cs.manchester.ac.uk
Open
The file won't be saved automatically.
Save
Save as
Cancel

Fig. 11 – Zipfile save dialogue

3. Find the downloaded **SSEM.ZIP** file on your PC's local drive and extract all of its contents (as shown below) into a new folder on your local drive (for example **C:\ssem**).



Name	Date modified	Type	Size
help.htm	09/12/2007 18:26	HTML Document	1 KB
index.htm	16/01/2008 17:26	HTML Document	3 KB
README.txt	16/01/2008 17:30	Text Document	2 KB
refman.htm	16/01/2008 11:47	HTML Document	23 KB
ssem.css	09/12/2007 18:27	Cascading Style She...	1 KB
ssem.jar	18/01/2008 16:35	Executable Jar File	1,750 KB

Fig. 12 – Zipfile contents

4. Once all the files are in the folder, the **ssem.jar** file is the Applet to open in order to start up the simulator locally. You will need to have Java Runtime installed in order to do this.

   - If you do <u>not</u> have Java Runtime installed on your PC already:

     Download and install the latest version as described below for "*Downloading and installing the Java Runtime*".

   - If you are not sure whether you have Java installed:

     Try opening **ssem.jar** in the extract folder on your PC by double clicking it in Windows explorer, and it will be indicated if there is no program installed to open it. In that case, cancel the offer that Windows makes to find a program for you, and follow the instructions below for "*Downloading and installing the Java Runtime*".

5. When you have Java Runtime installed on your PC:

   You will be able to open the **ssem.jar** file using Windows Explorer. This will start the freestanding version of the Baby simulator.

   You can also create a Desktop shortcut to the file for easy running later.

   Depending on your PC's folder layout and the security settings on the folder that you chose to extract the simulator into, you may also find that you need to review your folder permissions so that you can save and load Snapshot and Assembler program files at the chosen location using the simulator.

Note that if you also want to access the simulator offline locally with a Web browser, you will need to configure Java security settings to allow it – see the section below headed "*Setup for offline local running using a Web browser*".

B) Setup for offline local running using a Web browser

This method of running has two stages to the configuration:

1. Install the Java applet locally as described above for "*Setup for offline local running as a freestanding Java applet*".

2. Configure the Java security settings as needed to allow the browser to access the **index.htm** web page installed on the local PC's filestore as part of the main local Java applet installation. This file has coded in it a reference to run the **ssem.jar** file as a web page applet whenever the local **index.htm** page is displayed in a browser.

   To set up the Java security settings for this to work:

   i. Confirm the full local filestore path to the **index.htm** web page, *including the exact capitalisation of any folder and drive names in the file path*.

      The easiest way to determine this is to use Windows Explorer to locate the **index.htm** file in the folder holding the extracted Baby simulator files on your PC (for example, **C:\ssem**), then to double click on it to open in the browser. If prompted, click to **Allow Blocked Content**.

   ii. If (as it usually will) the page fails to load with **Application Blocked** messages, then you will need to configure the Java security settings for the page as in the following steps.

      If instead it loads cleanly at this point, and if the Baby simulator appears on the web page, then your Java setup needs no further configuration.

   iii. Before clearing down the **Application Blocked** messages, note the exact file path to the **index.htm** file as shown in the browser URL line, *including the case of all characters in it* – for the example below, this is **C:\ssem\index.htm**.

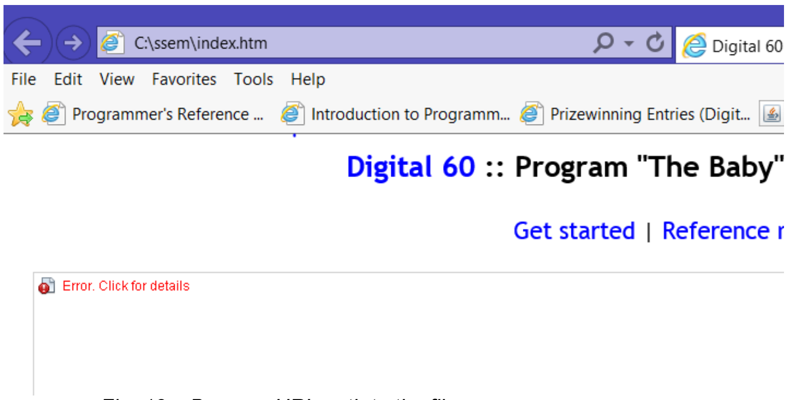
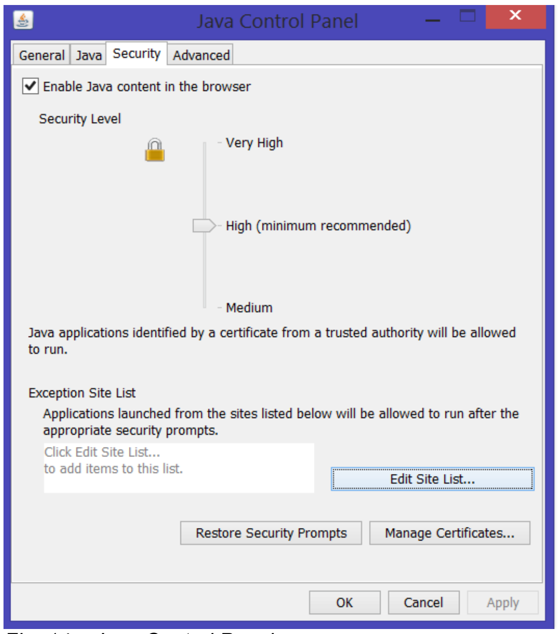

C:\ssem\index.htm
Digital 60
File Edit View Favorites Tools Help
Programmer's Reference ...
Introduction to Programm...
Prizewinning Entries (Digit...
Digital 60 :: Program "The Baby"
Get started | Reference r
Error. Click for details

Fig. 13 – Browser URL path to the file

iv. Open the **Configure Java** utility.

- For Window XP, Windows Vista and Windows 7, this is in **Start** > **All Programs** > **Java**.
- For Windows 8 or 8.1, use Search and type in **Configure Java** to find it.

v. When the **Java Control Panel** opens, click on the **Security** tab, as below, then on **Edit Site List**.



Java applications identified by a certificate from a trusted authority will be allowed to run.

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

Fig. 14 – Java Control Panel

vi. The **Exceptions Site List** will open.

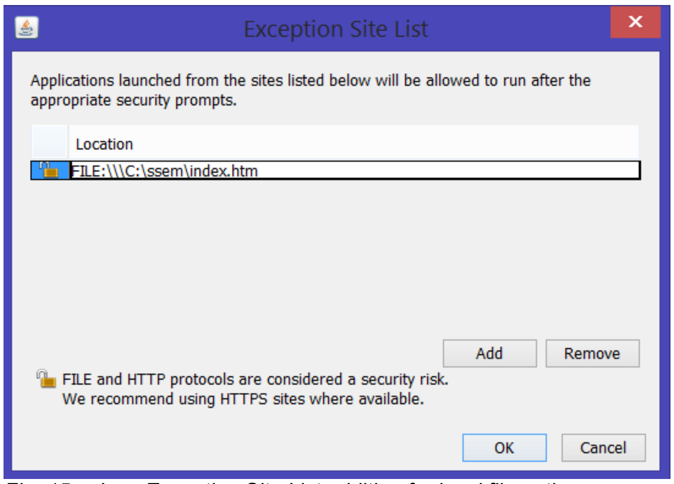
vii. Click on **Add**, then in the **Location** box type the characters **FILE:\\\** followed by the exact syntax of the drive path to the index.htm file as you noted it previously in the browser URL (in the example above, **C:\ssem\index.htm**).

This results in an entry needed, for this example, of

FILE:\\\C:\ssem\index.htm

Note again that the URL supplied is case sensitive throughout and the part after the three backslashes must match that appearing in the browser – in this example, if a **c** is typed instead of **C** it will not work.

When it appears similar to the entry below (using whatever the path to the file is on your own PC), click on **OK** to return to the main list:


Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

Location

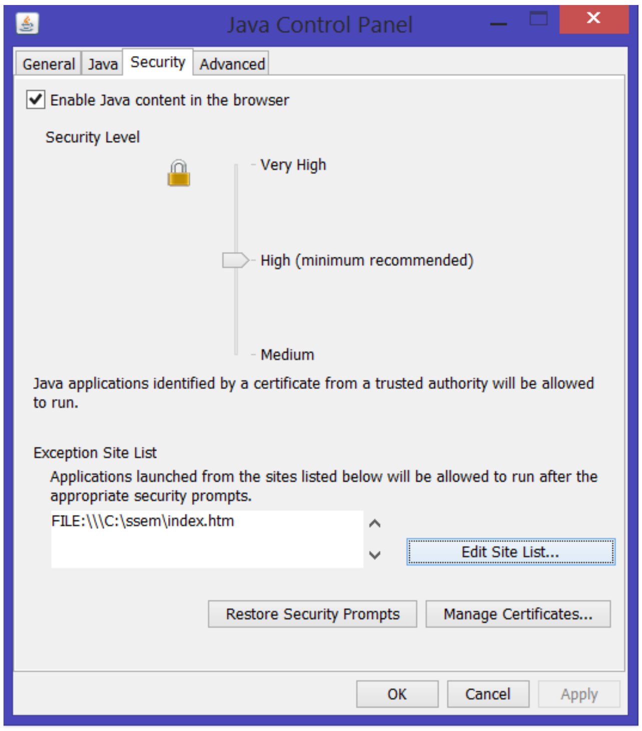
FILE:\\\C:\ssem\index.htm

FILE and HTTP protocols are considered a security risk.
We recommend using HTTPS sites where available.

Fig. 15 – Java Exception Site List addition for local file path

viii. You may receive a Security Warning now that including a FILE location on the Exception Site List is considered as security risk. If so, click on **Continue**. (Only do this for cases like this where the page location is well known and trusted).

ix. You will be returned to the first screen, now showing the site included in the current list of excluded sites, as below:


Java Control Panel

☑ Enable Java content in the browser

Security Level

Very High

High (minimum recommended)

Medium

Java applications identified by a certificate from a trusted authority will be allowed to run.

Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

FILE:\\\C:\ssem\index.htm

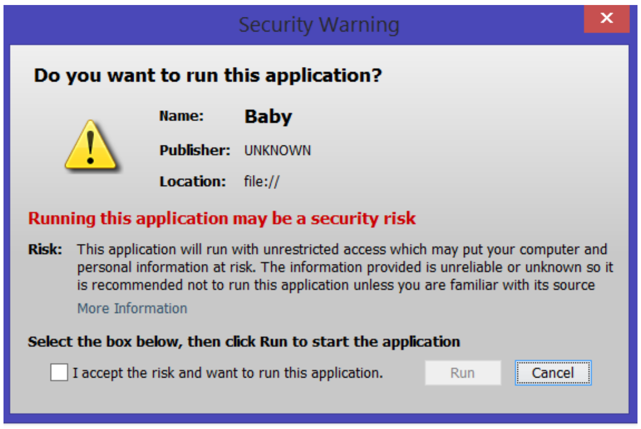
Fig. 16 – Java Control Panel with local file path added

x. Click on **OK** to complete the setup.

xi. Confirm access to the page using a fresh Web browser window.

You should now receive a warning message like the one below.

On this, tick the **I accept the risk** check box, and then click **Run**.


Security Warning

Do you want to run this application?

Name: Baby

Publisher: UNKNOWN

Location: file://

Running this application may be a security risk

Risk: This application will run with unrestricted access which may put your computer and personal information at risk. The information provided is unreliable or unknown so it is recommended not to run this application unless you are familiar with its source

More Information

Select the box below, then click Run to start the application

☐ I accept the risk and want to run this application.

Fig. 17 – Security Warning Message for local file access

xii. The **index.htm** page should then load normally, and the Baby applet should run from it.

You can then also create a bookmark entry in your browser Favourites to run it directly from the browser in the future, without the need to navigate to it with Windows Explorer.

Depending on your PC's folder layout and the security settings on the folder that you chose to extract the simulator into, you may find that you need to review your folder permissions so that you can save and load Snapshot and Assembler program files at the chosen location using the simulator.

C) Setup for online running from an existing Internet or Intranet Web page

For this mode of running, no local installation of the Baby Applet is required: instead, it just needs an appropriately configured Java SE Runtime security configuration for the target Web page URL, as follows:

1. Confirm that Java SE Runtime 6 or later is installed. If unsure, you can follow the installation process under "*Downloading and installing the Java Runtime*" below, and it will tell you if already present when it starts installing, at which point you can quit the setup.

2. The latest versions of Java SE will block the running of the Baby applet on an untrusted Web page because the applet is not signed with a known Certificate. Opening such a Web page with the applet on it will cause the error box "*Application Blocked by Security Settings*" to appear, and the applet will appear as a dotted outline, with a red icon and the word "Error" in it.

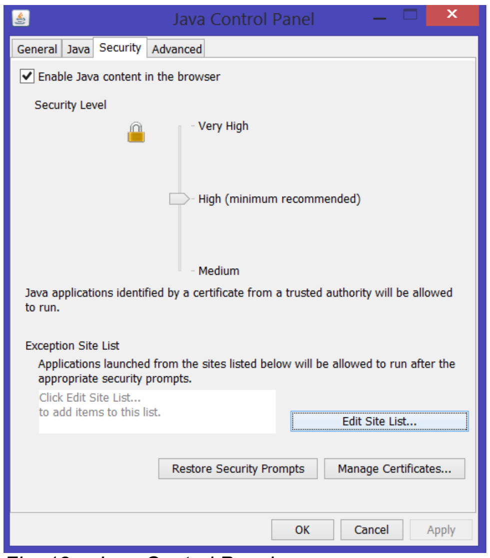
   You will then need to enable running for the URL of the containing Web page as follows:

   For the example of the target page holding the applet at *http://www.davidsharp.com/baby/* :

   i. Open the **Configure Java** utility.

      - For Window XP, Windows Vista and Windows 7, this is in **Start** > **All Programs** > **Java**.
      - For Windows 8 or 8.1, use Search and type in **Configure Java** to find it.

   ii. When the **Java Control Panel** opens, click on the **Security** tab, as below, then on **Edit Site List,** as shown:


Java Control Panel

General | Java | Security | Advanced

☑ Enable Java content in the browser

Security Level

Very High

High (minimum recommended)

Medium

Java applications identified by a certificate from a trusted authority will be allowed to run.

Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

Click Edit Site List...
to add items to this list.

Edit Site List...

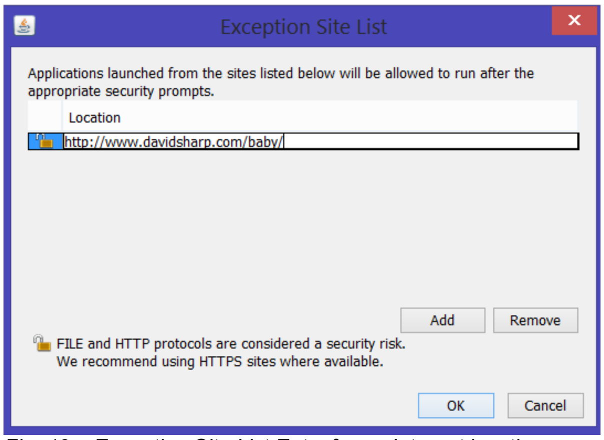
Restore Security Prompts | Manage Certificates...

OK | Cancel | Apply

Fig. 18 – Java Control Panel

iii. The **Exceptions Site List** will open. Click on **Add**, then in the **Location** box type the Web page URL containing the Baby applet – in this case *http://www.davidsharp.com/baby/*. Then click on **OK**.

Note that the URL supplied is case sensitive throughout – it must exactly match the way it appears in the browser's URL line when accessing the site to work. In this case, all must be in lowercase.


Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

Location

http://www.davidsharp.com/baby/

Add | Remove

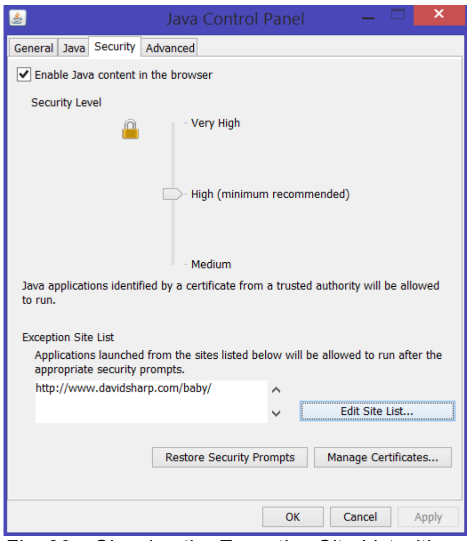
FILE and HTTP protocols are considered a security risk.
We recommend using HTTPS sites where available.

OK | Cancel

Fig. 19 – Exception Site List Entry for an Internet location

iv. You may receive a Security Warning that including an http location on the Exception Site List is considered as security risk. If so, click on **Continue**. (Only do this for cases like this example where the Web page is well known and is trusted).

v. You will be returned to the first screen, now showing the site in the list of excluded URLs, as below:


Java Control Panel

General | Java | Security | Advanced

☑ Enable Java content in the browser

Security Level

Very High

High (minimum recommended)

Medium

Java applications identified by a certificate from a trusted authority will be allowed to run.

Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

http://www.davidsharp.com/baby/

Edit Site List...

Restore Security Prompts | Manage Certificates...

OK | Cancel | Apply

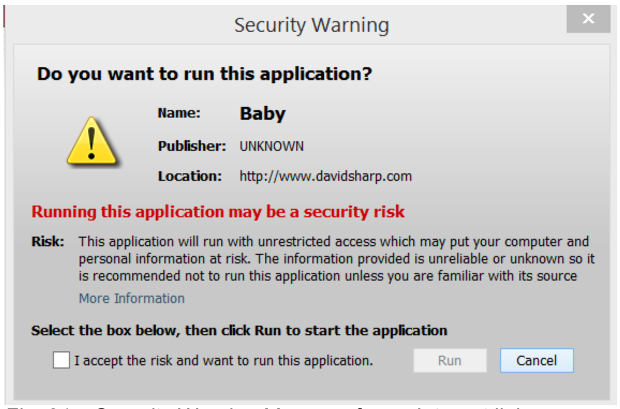
Fig. 20 – Showing the Exception Site List with an Internet location

Click on **OK** to complete the setup.

vi. Confirm access to the just configured URL using a fresh Web browser window.

You should now receive a warning message like the one below.

On this, tick the **I accept the risk** check box, and then click **Run**.



Security Warning

Do you want to run this application?

Name: **Baby**

Publisher: UNKNOWN

Location: http://www.davidsharp.com

Running this application may be a security risk

Risk: This application will run with unrestricted access which may put your computer and personal information at risk. The information provided is unreliable or unknown so it is recommended not to run this application unless you are familiar with its source

More Information

Select the box below, then click Run to start the application

☐ I accept the risk and want to run this application.

Fig. 21 – Security Warning Message for an Internet link

vii. The page should then load normally, and the Baby applet should run from it.

Note that a very similar configuration can be used to enable browser access to the Baby simulator held on a network share holding the Baby applet installation on another PC on the local network.

For example: replacing the above example string *http://www.davidsharp.com/baby/* in the Exception Site List entry by the string *FILE:\\\\192.168.0.106\ssem\index.htm* will allow browser access to the Baby Java simulator installation in network share *ssem* on the PC at address *192.168.0.106* using the browser URL *\\192.168.0.106\ssem\index.htm*. (You will note that the browser URL *\\192.168.0.106\ssem\index.htm* used to access the page on the remote PC is prefixed by the characters *FILE:\\* in the corresponding Exception Site List entry).

Depending on your PC's folder layout and the security settings on them, you may also find that you need to review your folder permissions settings so that you can save and load Snapshot and Assembler program files at the chosen location using the simulator.

Downloading and installing the Java Runtime (if required)

1. Go to http://java.com/en/download/.
2. Select the link for “Free Java Download”.
3. Click on “Agree and Start Free Download”.
4. Choose “Run” to download and install the program directly.
(Alternatively, if you prefer, you can save it to your local drive and execute it manually to install later).

APPENDIX B: Baby control panel switch and button reference

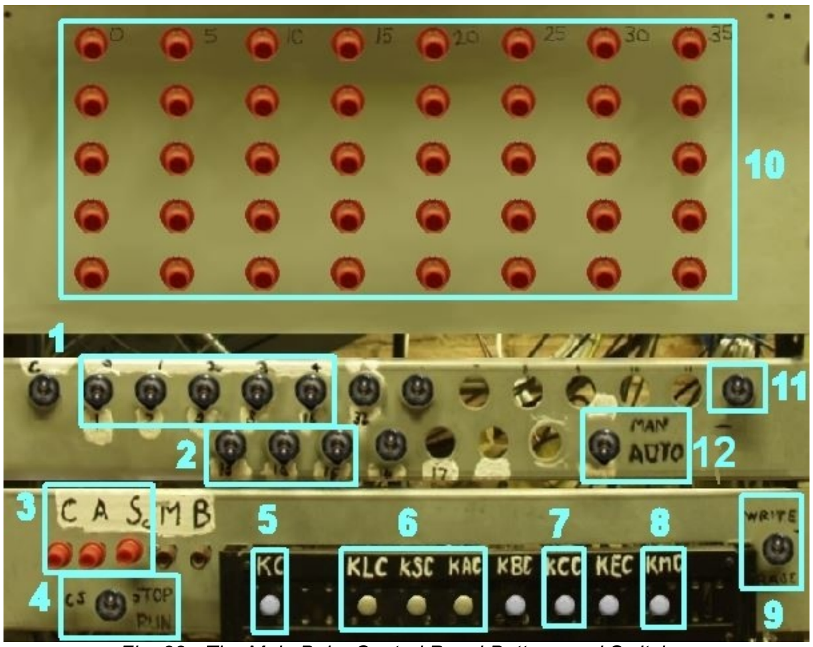


Fig. 22 - The Main Baby Control Panel Buttons and Switches

The uses of the main switches and buttons on the Baby control panel (using the numeric references to their positions in Fig. 22) are summarised below. These are fully functional on the Baby simulator, just as they worked on the original Baby machine in 1948, and on the recreated Baby in MOSI.

The details here are at a high level, and more details of Baby switch and button operation are available in the full SSEM Pogrammers' Reference (either http://www.digital60.org/rebuild/50th/competition/ssemref.html or the file **refman.jar** that is contained in the downloaded zipfile referred to in Appendix A).

Note also that the switches and buttons that are not picked out with numbers in Fig. 22 are there to accurately represent the physical switch layout of the original Baby in 1948, but they were not electrically connected on the actual machine, and had no effect on the machine's running. They similarly have no effect on operation of the Baby simulator.

1 Line (L) switches

When running in Manual mode, these 5 switches select a line number in Store represented in binary, with a Down switch position representing a 1, and an Up position a 0. The leftmost switch represents the least significant digit (1's), and the rightmost switch the most significant (16's). If you look closely, you can see that the numbers 1, 2, 4, 8 and 16 are written underneath them as a reminder.

2 Function (F) switches

When running in Manual mode, these 3 switches select a Function number to be executed, represented in binary, with a Down switch position representing a 1, and an Up position a 0. Where the function needs it, the Line switch positions will indicate the store location information needed to perform the Function. The leftmost switch represents the least significant digit (1's), and the rightmost switch the most significant (4's). If you look closely, you can see the numbers 13, 14, and 15 are written underneath them as a reminder of the positions those 3 Function bits take in a stored instruction line.

3 Monitor Display Selector Buttons

These buttons allow the display tube at the top of the control panel to be switched to show the contents of each of the three store tubes: Select the appropriate display (Main Store, Accumulator or Control) by pressing in one of the three red buttons: **Sc** for Store, **A** for Accumulator or **C** for Control. Only one of these buttons can be pressed in at a time.

4 STOP/RUN switch

For the continuous running of a program or function, setting this switch Up selects **STOP**, and setting it Down to **RUN** starts the program or function defined in switches running. Also labelled **CS** (standing for "Completion Signal").

5 KC switch (Key Completion)

If the program is not currently running then pressing this switch down and allowing it to return to the horizontal causes the Baby to start processing the next instruction, either from Store (when the machine is switched to **AUTO** mode), or from the Line and Function switch settings (if in **MANUAL** mode).

6 KLC switch (Key Line Clear)

Clears the selected Action line in the store display.

KSC switch (Key Store Clear)

Clears the whole Main Store to zeroes, deleting any programs and data held there.

KAC switch (Key Accumulator Clear)

Clears the Accumulator

7 KCC switch (Key Control Clear)

Clears both the Control and Accumulator stores.

8 KMC switch

This switch has no function on the Baby simulator, and it had none on the 1948 Baby machine.
On the MOSI Baby, it enables quick loading into store of prepared test programs from the support PC.

9 WRITE/ERASE switch

As in the description of the Typewriter (see below), this switch determines whether a "one" or a "zero" is "typed" into the Store in the selected line when the red typewriter buttons are pressed.

10 Typewriter Buttons

The panel contains 40 red push-buttons arranged as eight columns of five buttons. The buttons are numbered 0 to 31 starting at top left and running down a column and then down succeeding columns from left to right. Only the first 32 buttons are connected, corresponding to the 32 bits in the store line display. The Typewriter is normally used while the machine is at Stop, i.e. not executing instructions. If the **WRITE/ERASE** Switch is at **WRITE**, then pushing a typewriter button causes a "one" to be written into the corresponding digit position of the Action Line. If the Switch is at **ERASE**, then a zero is written to the store position.

11 Store Display Line Highlighter switch

This switch has no effect on the Baby simulator. On the actual MOSI Baby, it allows a bold highlight to be applied to the current program action line on the CRT monitor display of the Store.

12 MANUAL/AUTO switch

a) Switch at **MANUAL**, with **STOP/RUN** at **RUN**:

The instruction represented in the current settings of the Line and Function switches is executed repeatedly.

b) Switch at **MANUAL**, with **STOP/RUN** at **STOP**:

This sets the machine into "Single Step" mode using the physical switch settings to identify the function to be run. This is a special case where one instruction action is executed per single step. The instruction represented by the current setting of the Line and Function switches is executed once whenever the **KC** switch is pressed down.

c) Switch at **AUTO**, with **STOP/RUN** at **RUN**:

Instructions are read from the Store and executed normally to run the program. All Line and Function switches must also be set Down for this to work.

d) Switch at **AUTO**, with **STOP/RUN** at **STOP**:

This sets the machine into "Single Step" mode, reading the program lines in Store for the functions to be exectuted. This is a special case of running in which just one program instruction is executed per single step. The instruction represented by the next program instruction line in Store is executed when the **KC** switch is pressed down. All Line and Function switches must also be set Down for this to work.

To clear the Store ready for loading of a fresh program:

1. Click down (and allow to return) the switch labelled **KSC** at position **6**.

To set the Normal Run Setting Switch Positions for automated running of a program in Store:

1. Set **STOP/RUN** Switch at **4** to the Up position (**STOP**).
2. Set all Line switches at **1** to the Down position (selecting binary Line 31).
3. Set all Function switches at **2** to the Down position (selecting binary Function 7).
4. Set **WRITE/ERASE** switch to the Up (**WRITE**) position.
5. Set the **AUTO/MANUAL** switch at **12** to the Down (**AUTO**) position.

To start the program currently in Store running:

1. Clear the Accumulator and Control by clicking down (and allowing to return) the switch labelled **KCC** at position **7**.
2. Check whether the Stop neon is currently lit.
   If so:
   - ➢ Press and allow to return switch **KC** at position **5** (which will extinguish it)
   - ➢ Press and allow to return switch **KCC** at position **7** (this resets the program counter).
3. Set **STOP/RUN** Switch at **4** to the Down position (**RUN**).
4. If you want to stop the program running before it automatically stops: set the **STOP/RUN** Switch at **4** to the Up position (**STOP**). You can then choose to restart it from where it is by setting the switch Down again (**RUN**), or by single-stepping it from where it was stopped by repeatedly pressing the **KC** switch at position **5** and allowing it to return. If you have single stepped the program in that way, you can still switch back to automated running again by setting the **STOP/RUN** switch back to Down (**RUN**).

To Single-Step a program that is currently in Store:

1. Set **STOP/RUN** Switch at **4** to the Up position (**STOP**).
2. Set all Line switches at **1** to the Down position (selecting binary Line 31).
3. Set all Function switches at **2** to the Down position (selecting binary Function 7).
4. Set **WRITE/ERASE** switch to the Up (**WRITE**) position.
5. Set the **AUTO/MANUAL** switch at **12** to the Up (**MANUAL**) position.

When ready to single-step the program in Store:

1. Clear the Accumulator and Control by clicking down (and allowing to return) the switch labelled **KCC** at position **7**.
2. Check whether the Stop neon is currently lit.
   If so:
   - Press and allow to return switch **KC** at position **5** (which will extinguish it)
   - Press and allow to return switch **KCC** at position **7** (this resets the program line counter).
3. Repeatedly press and allow to return switch **KC** at position **5**. Each press of the switch causes the next program instruction line to be read in and executed.

4. Select the appropriate display (Main Store, Accumulator or Control) to watch the results of the individual program lines executing when **KC** is being pressed by pressing in one of the three red buttons at position **3** (**Sc** for Store, **A** for Accumulator or **C** for Control).
5. If you want to switch to automated running after single-stepping part of the program, you can do this by setting the **STOP/RUN** switch in position **4** to Down (**RUN**).

APPENDIX C: Setting up the old Java Baby simulator

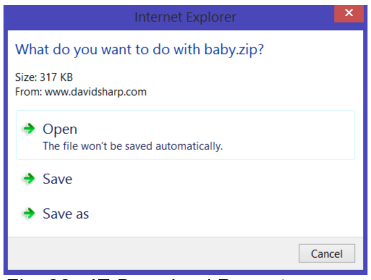
PC System requirements:

- Windows XP, Windows Vista, Windows 7, Windows 8 or Windows 8.1.
- A Java Runtime installation of version 1.2 or later. (If not already installed, see the section below on "*Downloading and installing the Java VM environment*").

To download and set up the original Baby Simulator:

1. The original simulator itself (originally written in 2001) may be downloaded from David Sharp's archive website by going to the following link[1]: http://www.davidsharp.com/baby/baby.zip.

2. A message similar to the one shown will appear (the exact message will vary depending on the version of your Operating System and web browser). Choose the option to **Save as**, and store the zip file in a suitable folder on your PC's local drive, noting where this is.



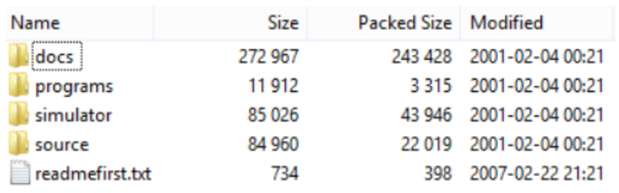
Internet Explorer

What do you want to do with baby.zip?

Size: 317 KB
From: www.davidsharp.com

Fig. 23 - IE Download Prompt

3. Find the just downloaded **BABY.ZIP** file on your PC's local drive and extract all of its contents (as shown below) into a new folder on your local drive (for example **C:\Babysim**).



Name	Size	Packed Size	Modified
docs	272 967	243 428	2001-02-04 00:21
programs	11 912	3 315	2001-02-04 00:21
simulator	85 026	43 946	2001-02-04 00:21
source	84 960	22 019	2001-02-04 00:21
readmefirst.txt	734	398	2007-02-22 21:21

Fig. 24 - Zip File Contents

4. Once all the files are extracted under the chosen folder, the **Baby.class** Java file in the **simulator** sub-folder is the one that needs to be run to start the simulator. The following steps describe how to set up a shortcut to allow this to be easily run from the Windows Desktop.

5. Continuing the setup requires a Java VM environment to be on your PC of version 1.2 or later.

   - To check whether you have Java installed already: Go to http://java.com/en/download/ , then choose the link to verify the version of Java you have installed (the link is normally labelled "**Do I have Java?**").
   - If you do not have Java installed already: See the section at the end on how to set up the Java VM environment before continuing.
   - If you do have Java installed already: Make a note of the full path of the **java.exe** file on your PC (this will often be **C:\Program Files (x86)\Java\jre7\bin\java.exe**).
   - Also note the path holding the just-extracted **Baby.class** file (in the example above, this was **C:\Babysim\simulator**). You will need to type this in when setting up the shortcut later on.

6. To set up the Desktop shortcut link to be able to easily start up the emulator:

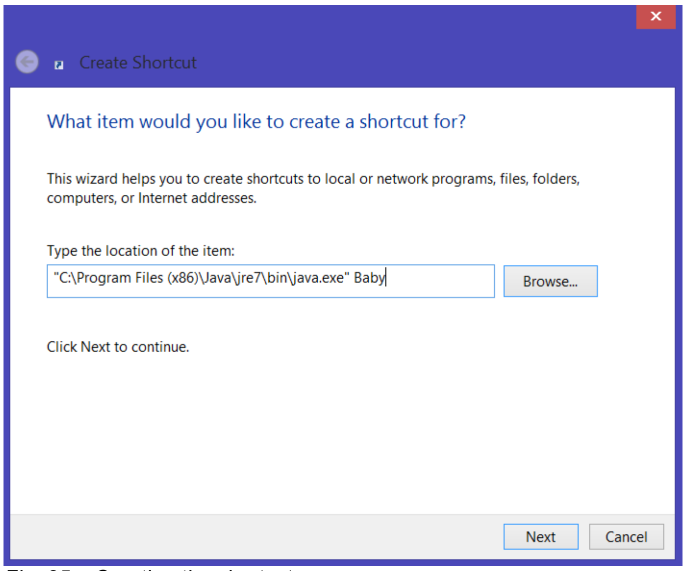
   i. Right-click on a free space on the Windows Desktop, and choose **New,** then **Shortcut**.

[1] For background information, see the main page at http://www.davidsharp.com/baby/.

ii. When the **Create Shortcut** window opens, click on the Browse button and navigate the folders to the location of the installed **java.exe** file (as noted above), then click **OK**. This will enter the path to the file for you into the location box, adding quotes around it if it contains spaces.

iii. After it does this, type in a space after it, followed by the word Baby **(note that it <u>must</u> be typed in with a capital "B" and the rest in lowercase)**.

It should then look like the following (check particularly that there is a capital "B" on Baby):



What item would you like to create a shortcut for?

This wizard helps you to create shortcuts to local or network programs, files, folders, computers, or Internet addresses.

Type the location of the item:

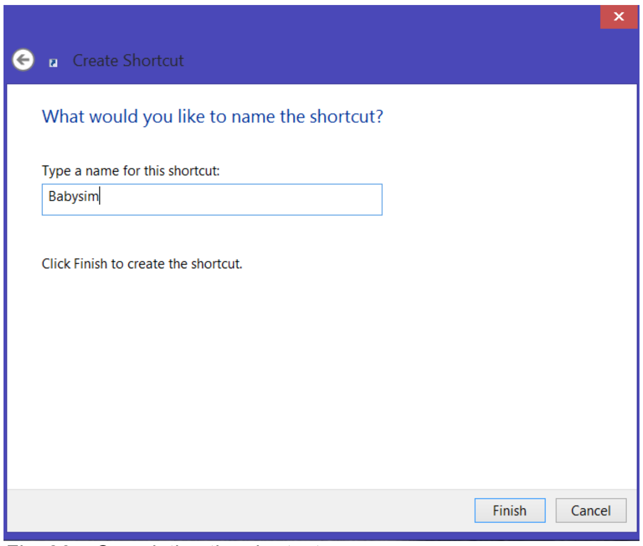
"C:\Program Files (x86)\Java\jre7\bin\java.exe" Baby

Click Next to continue.

Fig. 25 – Creating the shortcut

Click **Next** to continue.

iv. On the next screen, provide the Shortcut name on the next screen – for example **Babysim**, as is shown below:



What would you like to name the shortcut?

Type a name for this shortcut:

Babysim

Click Finish to create the shortcut.

Fig. 26 – Completing the shortcut

Click on **Finish**. The shortcut appears on the Windows Desktop.

v. Before the simulator will run, you now need to change the **Start in** folder setting on the shortcut you have just created.

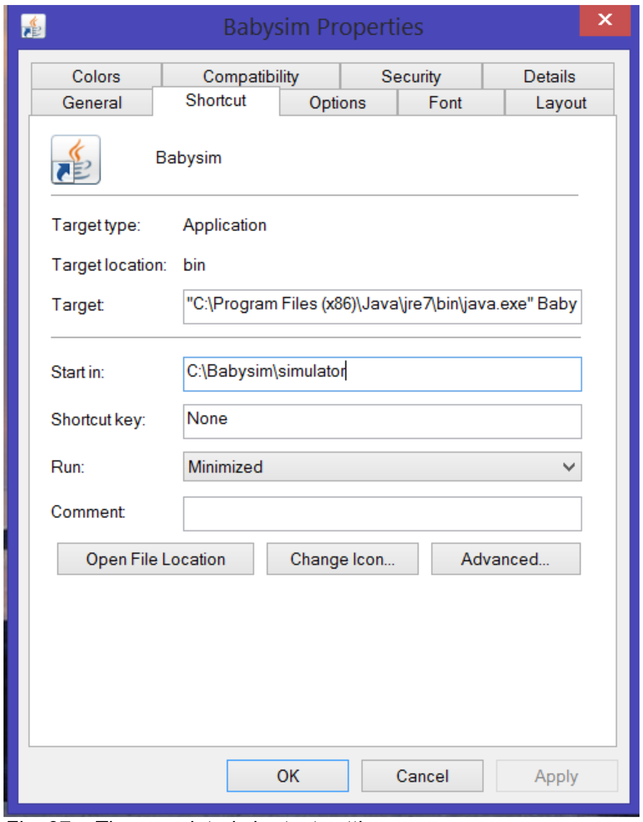
To do that:

- Right-click on the shortcut just created on the Desktop, then choose **Properties**.
- In the window that appears, overtype the **Start in** setting with the path you noted to the extracted **Baby.class** file, but *without* the Baby.class name at the end. If the path contains a

space, you should include quotes around it. (For the example above, the result will read **C:\Babysim\simulator**).

- Change the **Run** setting from **Normal Window** to instead read **Minimized**.

  When done, it will look like the following display:



Babysim

Target type: Application

Target location: bin

Target: "C:\Program Files (x86)\Java\jre7\bin\java.exe" Baby

Start in: C:\Babysim\simulator

Shortcut key: None

Run: Minimized

Comment:

Fig. 27 – The completed shortcut settings

After checking, click **Apply**, then click **OK** to save it.

7. The simulator should now be runnable from the Desktop icon.

Depending on your PC's folder layout and the security settings on the folder that you chose to extract the simulator into, you may also find that you need to review your folder permissions so that you can save and load Snapshot and Assembler program files at the chosen location using the simulator.

Starting the simulator:

1. Locate the Desktop icon (set up as described above) and double-click on it to open the simulator.

2. Three windows will open:

   - A DOS command window (automatically minimised)
   - A **Baby Simulator** store display and menu options window
   - A **Switch panel** window

3. Leave the DOS window minimised in the background (if you close it, the simulator windows will close), then size and drag the other two windows so that they sit side by side, as shown below:

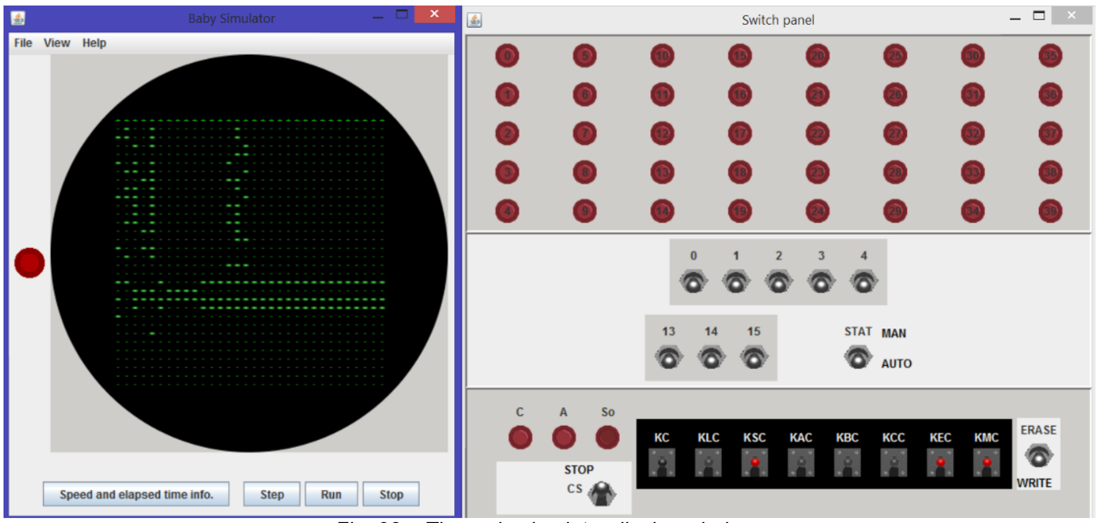


Fig. 28 – The main simulator display windows

Downloading and installing the Java VM environment (if required):

1. Go to http://java.com/en/download/.
2. Select the link for "Free Java Download".
3. Click on "Agree and Start Free Download".
4. Choose "Run" to download and install the program directly. (Alternatively, if you prefer, you can save it to your local drive and execute it manually to install later).