DIGITAL MAGIC EXPOSED

Book 2 – The Simple Digital Processor Emulator

Forward

Book 1 introduces the fundamentals of digital processing by describing a simple processor. This book describes an emulation of the simple processor through the use of one of two widely available spreadsheet tools: Microsoft Excel and Apache OpenOffice Calc. The emulator provides a graphical demonstration of the processor as it calculates each machine cycle.

The use of a spreadsheet program may appear a little odd. However, both the spreadsheet programs are extremely feature-rich and easy to use. It is intended that the emulator is modifiable without the need for additional tools.

The emulator may be used as an illustration to Book 1 Part 1. The examples given in Book 1 are included in the emulator (although it is simple to add them directly). Other examples from the books are included and additional programs can be added by the user as required.

The emulator includes a mnemonic source code editor and assembler. Additionally, binary files can be created to load programs directly into the electronic simulation of the simple processor in Book 3.

Microsoft Excel is part of the Microsoft Office suite of programs available widely. Calc is part of the Open Office open source suite available freely from the Openoffice web-site.

This book consists of two parts:

- Part 1 – This part describes the emulator and its operation. Installation requirements for each tool (using Excel or Calc) are detailed.
- Part 2 – This part provides some detail on the emulator’s spreadsheets and macros to aid further development of the emulator.

Part 1 – Installing and Operating the Emulator

1 Installation

The emulator requires that Microsoft Excel or Apache Open Office is installed on the pc. Although each of these spreadsheet tools is capable of opening files of the other type it is important that the corresponding file type is run or the emulator will not function. Before either of the emulators can be run attention is drawn to a few important properties.

1.1 Macros

The emulators make use of unsigned macros in their operation. The default security level of both Excel and Calc will block the use of unsigned macros and the emulator will not function. Note that it is not advisable to run macros that cannot be trusted since it is possible to write malicious macros.

Obtaining a "Trusted Authority" signature for general use is costly for an educational tool offered for free. The macros used in the emulator are harmless and fully described in Part 2 of this book. Published versions are shown, so it is possible to check the macros within the spreadsheet before allowing them to run.

For the emulator to run the spreadsheet program must be satisfied the macros are trusted to be safe. This can be achieved either by running the emulator from a "Trusted Location" or altering the "Trust Centre" settings for the spreadsheet. Generally, the first option is preferred but is not available to Microsoft Office 2003. Both techniques are effective for successful operation of the emulator and are described for each spreadsheet in the following.

Note that it is necessary to restart the spreadsheet and possibly the computer for the new security settings to take effect before the emulator can be opened.

1.2 Notes for Excel

1.2.1 Version

The emulator was developed using Microsoft Office 2003. Therefore, it is expected that Excel 2003 or later is used.

1.2.2 Setting the Trusted Location

For Excel versions after 2003 the "Trust Centre" can be updated to include the folder (file path) into which the emulator is placed. To do this, open Excel (blank sheet) and select the Trust Centre as follows.

File > Options > Trust Centre

Select "Trust Centre Settings". In the left pane select "Trusted Locations" and (using "Add new location") add the file path for the emulator. (An easy way to do this is to view the emulator file in Explorer and click an empty space in the file path viewer. The path is shown selected and can be copied for pasting in the Trusted Locations). Click “OK”. Close Excel.

1.2.3 Setting the Trust Centre

This is the only option for Excel 2003. To observe and modify the macro security level, open Excel (blank sheet). On the menu bar select

Tools > Macro > Security

The “Security Level” tab shows the level on a radio button. The default setting is “High” and the macros will not run. The “Medium” setting provides an alert that macros are present and buttons for enabling or disabling them when the emulator is opened. Select the Medium option; click “OK” and close Excel. To run the emulator, choose the “Enable Macros” button only when starting-up the emulator.

1.2.4 Analysis ToolPak

The emulator makes extensive use of an Excel function which converts decimal numbers to hexadecimal numbers. This function has been a standard Excel function in versions of Excel after 2003, but not in 2003. If Excel 2003 is used, then the “Analysis Toolpak” add-in needs to be selected. This is achieved by opening Excel and on the menu bar select

Tools > Add-ins…

Check the “Analysis ToolPak” box and click “OK”. Note that if Add-ins are not loaded then Excel will attempt to load them, possibly from the Office installation disk. This procedure must complete. The emulator will not function properly on Excel 2003 without Analysis ToolPak. If this remains unavailable then using the Calc version is recommended (i.e. install Open Office).

Once added, Analysis ToolPak is always included whenever Excel runs on the pc.

1.2.5 Program Close

The emulator very often requests a save when it is closed. The response selected should always be “no”. Version control is outlined in section 1.4.

1.3 Notes for Calc

1.3.1 Version

The emulator was developed on Apache OpenOffice 4.1.5.

1.3.2 Setting the Trusted Location

To observe and modify trusted locations, open Calc (blank sheet) spreadsheet. On the menu bar select

Tools > Options

In the left-hand pane under “OpenOffice” select “Security”. The security options appear in the right-hand pane. Click on the “Macro Security” button.

The “Trusted Sources” tab shows the “Trusted file locations”. Select “Add…” to add the file path of the emulator and click “OK”, followed by “OK” on the Options window and close Calc. (An easy way to do this is to view the emulator file in Explorer and click an empty space in the file path viewer. The path is shown selected and can be copied for pasting in the Trusted file locations).

1.3.3 Setting the Trust Centre

Included for completeness but updating the Trusted Location is the recommended option.

To observe and modify the macro security level, open Calc (blank sheet) spreadsheet. On the menu bar select

Tools > Options

In the left-hand pane under “OpenOffice” select “Security”. The security options appear in the right-hand pane. Click on the “Macro Security” button.

The “Security Level” tab shows the level on a radio button. The default setting is “High” and the macros will not run. The “Medium” setting provides an alert that macros are present and buttons for enabling or disabling them when the emulator is opened. Select the Medium option; click “OK”, followed by “OK” on the Options window and close Calc. To run the emulator, choose the “Enable Macros” button only when starting-up the emulator.

1.3.4 Using the Calc Emulator for the First Time

During first-use a dialog box may appear warning of cell data over-writes. Check the “Do not show warning again” box and click “Yes”.

1.3.5 Program Close

The emulator very often requests a save when it is closed. The response selected should always be “Discard”. Version control is outlined in section 1.4.

1.4 Creating versions, keeping back-ups

The emulator spreadsheet requests a save most of the times it is closed. Therefore, it is easy to save a changed emulator with the same name as supplied. It is for the user to manage emulator versions. As a minimum, a copy of the supplied emulator should be kept safe somewhere. It is recommended that changed emulators have changed filenames.

For modified versions which are to be kept (e.g. added code), the "save as" option should be used with an appropriate filename.

If versions are saved elsewhere such as a different directory, attention is drawn to the security restrictions on macros noted earlier.

1.5 Document protection (sheets)

All the sheets in the emulator have sheet protection applied. Only the sheets "Source" and "Program Store" contain cells (in white) which can be edited directly. The sheet protection for the supplied emulator does not include a password on any of the sheets.

The purpose of the protection is to prevent accidental alteration of the sheets during normal use. Clearly, it would be very easy to disrupt the operation of the emulator by altering a cell without protection.

To change a sheet or view hidden rows or columns, a sheet can be unprotected as follows:

- For Excel: Select Tools > Protection > Unprotect Sheet…
- For Calc: Select Tools > Protect Document > Sheet…

Protection may be reapplied by the same procedure to reverse the action. An additional dialog box appears during this procedure which requires no modification (i.e. click "OK").

There are no hidden sheets.

2 Overview

2.1 Introduction

Book 1 Part 1 introduces the basic concepts at the heart of today's technology devices. Fundamentally, the processor is a general purpose machine which can perform calculations limited only to its designed capability and the imagination of the programmer. Without the program, and the Machine Code it generates, the processor has no purpose.

Software is not in the scope of these books. Software applies a level of abstraction which makes the inner workings of the processor invisible and greatly increases the productivity of software engineering. Ultimately, software does generate the Machine Code which runs on the processor.

In this book the programs are written using the simple processor Instruction Codes (instruction mnemonics and any data) developed in Book 1 since these are much easier for a person to handle than Machine Code. The emulator tool provides a set of screens which enables instructions to be written, converted to Machine Code, loaded into the emulator and the program stepped through as it runs.

2.2 Main View – Processor Sheet

When the emulator is opened the processor sheet appears (Figure 1).

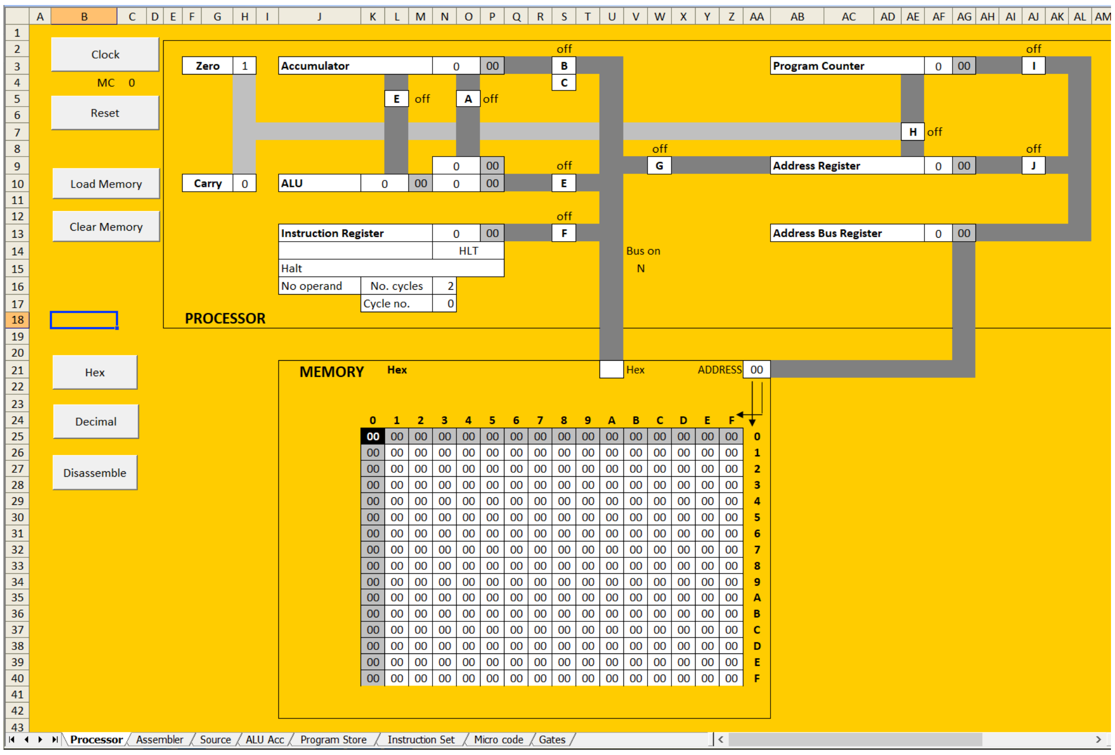
The screen provides a representation of the simple processor described in Book 1. In Figure 1 there is currently no program data loaded into memory. All 256 bytes of memory can be seen with the data displayed in hexadecimal. The shading in memory indicates the row and column decoded and hence the memory location (with black background) which could be selected for a read or write.

The main features of the sheet are described here. A detail description of the sheet features is contained in section 4.

Each one of the six registers has shaded cells containing the current hex values in the registers. The number on the left adjoining cell is the decimal equivalent. The Gates are all off, as shown.

The processor sheet is only updated by clicking any of the seven buttons on the left: Clock, Reset, Load Memory, Clear Memory, Hex, Decimal, Disassemble. All cells are locked against direct update.

The sheet tabs at the bottom of the sheet provide access to the other emulator features.



0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	6
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	7
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	8
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	A
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	B
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	C
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	D
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	E
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	F

Figure 1 Processor sheet

- Clock - Clicking the Clock button advances the processor by one machine cycle. The MC count below the button indicates the number of cycles executed since a Reset. In Figure 1 the processor is in the reset state.

- Reset - This button resets the processor to the state seen in Figure 1 but does not affect the content of memory. The MC count is set to zero.

- Load Memory - The Load Memory button loads the memory with the Machine Code generated by the Assembler sheet, described in section 2.3.

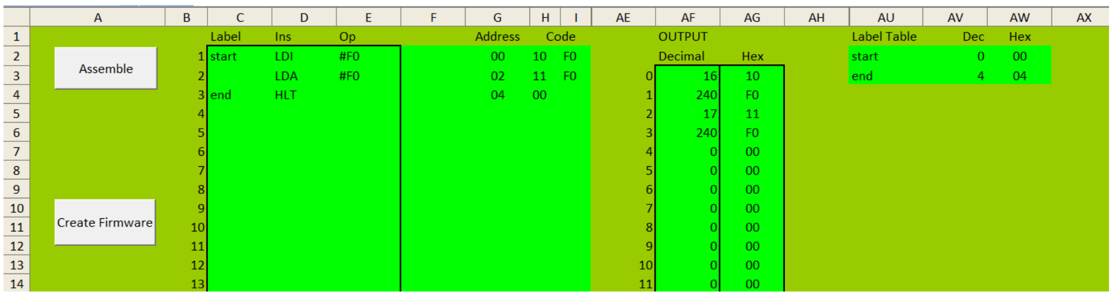
- Clear Memory - This button clears the memory to zeroes as shown in Figure 1.

- Hex - The memory content is displayed as hex characters and is indicated in the top row of Memory.

- Decimal - The memory content is displayed as decimal characters and is indicated in the top row of Memory.

- Disassemble - The memory content is displayed disassembled into instruction mnemonics and operand data over the length of the program and is indicated in the top row of Memory. Data are shown in hex.

2.3 Assembler



	A	B	C	D	E	F	G	H	I	AE	AF	AG	AH	AU	AV	AW	AX
1			Label	Ins	Op		Address	Code			OUTPUT			Label Table	Dec	Hex	
2		1	start	LDI	#F0		00	10	F0		Decimal	Hex		start	0	00	
3		2		LDA	#F0		02	11	F0	0	16	10		end	4	04	
4		3	end	HLT			04	00		1	240	F0					
5		4								2	17	11					
6		5								3	240	F0					
7		6								4	0	00					
8		7								5	0	00					
9		8								6	0	00					
10		9								7	0	00					
11		10								8	0	00					
12		11								9	0	00					
13		12								10	0	00					
14		13								11	0	00					

Figure 2 Assembler sheet

The Assembler sheet Figure 2 consists of input instructions and output Machine Code values for the instructions. The sheet is only updated by the button “Assemble”. All cells are locked against direct update.

The Assemble button copies the content of the codes written in the Source sheet (section 2.4) into the corresponding locations in the Assembler sheet where they are assembled into the output Machine Code values (decimal and hex are shown). The Assembler creates all the Machine Code values from the instructions and labels on the Source sheet. In Figure 2 the instructions are those given by the first example shown in Book 1 (A Basic Operation).

The Output values are copied into the processor memory by the Load Program button on the Processor sheet.

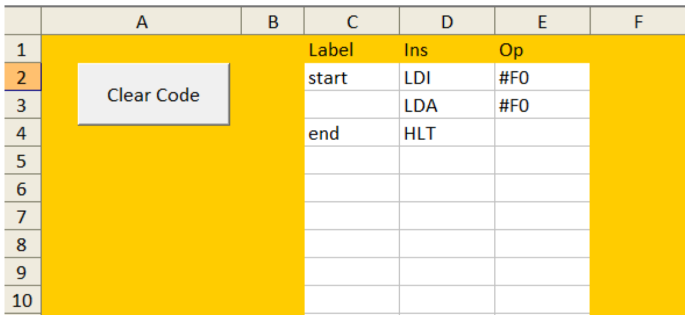
The assembled Machine Code values are also shown in hex in the two columns headed “Code”. The address of the first (and possibly only) byte is shown in the column headed “Address”. This part of the display can be useful when navigating around the stored values and observing the register displays in the simulator Book 3.

The values assigned to the labels are shown in decimal and hex in the “Label Table” to the right of the sheet. The syntax of the labels, instructions and operands is described in section 5.

The button “Create Firmware” creates a binary file of 256 bytes containing the assembled Machine Code values. The file is always named “Firmware.bin” and can be renamed directly (e.g. through Explorer) as required. An existing Firmware.bin file in the folder is replaced. The file is used by the simulator (Book 3) to load Machine Code values into the simulated hardware electronic circuit. The file is created in one of the following folders:

- For Excel: the “Default file location” set in the “General” tab of the spreadsheet options
- For Calc: in the same folder as the emulator.

2.4 Source



	A	B	C	D	E	F
1			Label	Ins	Op	
2			start	LDI	#F0	
3				LDA	#F0	
4			end	HLT		
5						
6						
7						
8						
9						
10						

Figure 3 Source sheet

Instruction Codes can be written directly into the open (white) cells of the Source sheet Figure 3. Full instructions on the rules governing the formatting and writing of codes are given in section 5.

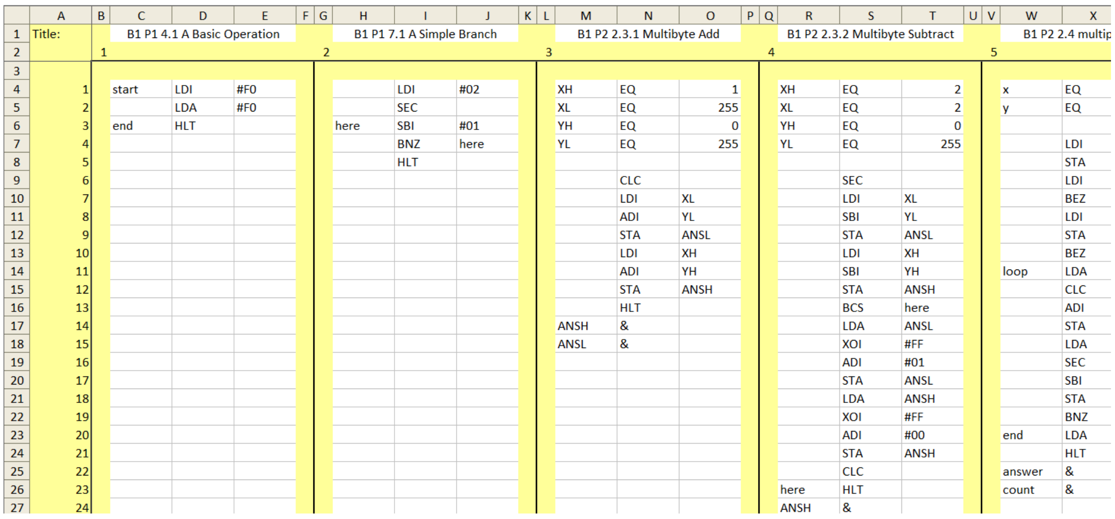
The button "Clear Code" is a short-cut for clearing all the cells to blank.

2.5 ALU Acc

The ALU and Accumulator processing features described in Book 1 are all calculated on this sheet. The sheet is divided into two parts: ALU (upper) and Accumulator (lower). The relevant part is highlighted only when a corresponding instruction is processed.

Further detail on the sheet is given in section 6.

2.6 Program Store



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	Title:		B1 P1 4.1 A Basic Operation					B1 P1 7.1 A Simple Branch					B1 P2 2.3.1 Multibyte Add					B1 P2 2.3.2 Multibyte Subtract					B1 P2 2.4 multip	
2		1					2					3					4					5		
3																								
4	1		start	LDI	#F0				LDI	#02			XH	EQ	1			XH	EQ	2			x	EQ
5	2			LDA	#F0				SEC				XL	EQ	255			XL	EQ	2			y	EQ
6	3		end	HLT				here	SBI	#01			YH	EQ	0			YH	EQ	0				
7	4								BNZ	here			YL	EQ	255			YL	EQ	255				LDI
8	5								HLT															STA
9	6													CLC					SEC					LDI
10	7													LDI	XL				LDI	XL				BEZ
11	8													ADI	YL				SBI	YL				LDI
12	9													STA	ANSL				STA	ANSL				STA
13	10													LDI	XH				LDI	XH				BEZ
14	11													ADI	YH				SBI	YH			loop	LDA
15	12													STA	ANSH				STA	ANSH				CLC
16	13													HLT					BCS	here				ADI
17	14												ANSH	&					LDA	ANSL				STA
18	15												ANSL	&					XOI	#FF				LDA
19	16																		ADI	#01				SEC
20	17																		STA	ANSL				SBI
21	18																		LDA	ANSH				STA
22	19																		XOI	#FF				BNZ
23	20																		ADI	#00			end	LDA
24	21																		STA	ANSH				HLT
25	22																		CLC				answer	&
26	23																	here	HLT				count	&
27	24																	ANSH	&					

Figure 4 Program Store sheet

The Program Store sheet Figure 4 is simply a repository for saved programs. A title may be added as shown. The titles shown in the figure correspond to the section numbers in Book (B) and Part (P) for the examples shown. When a saved program is to be assembled and loaded into the processor, it is copied (copy and paste) from the store into the corresponding location in Source.

2.7 Other Sheets

- Instruction Set - This sheet defines the Instruction Set for the simple processor.
- Micro Code - This sheet contains the documented micro codes for the simple processor.
- Gates - This sheet defines the Gate operation for the simple processor emulator.

Each of these sheets is further described in Part 2 of this Book.

3 User Guide

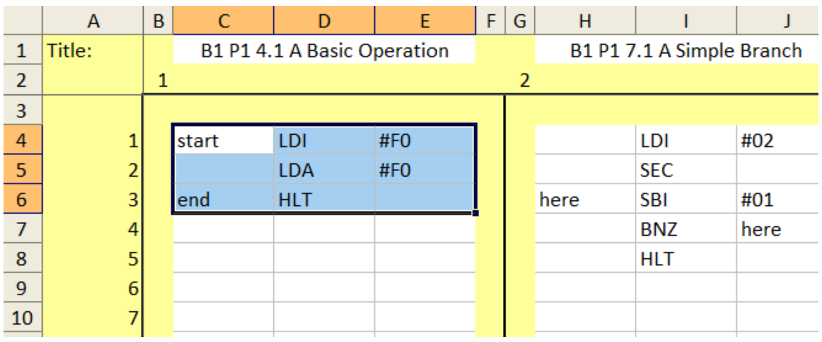
This section provides sufficient guidance to allow operation of the emulator as a demonstration of the simple process flows described in Book 1 Part 1.

In Book 1 Part 1 section 4.1 the simple idea of loading the accumulator with data is introduced. The process flow can be demonstrated on the emulator as follows.

- With the emulator open go to the Source sheet and ensure the cells for user programs are empty. If they are not click the “Clear Code” button.

- If at any time the spreadsheet provides a warning that cells containing data are being overwritten, select the check-box to block the warning since this is normal operation and continue.

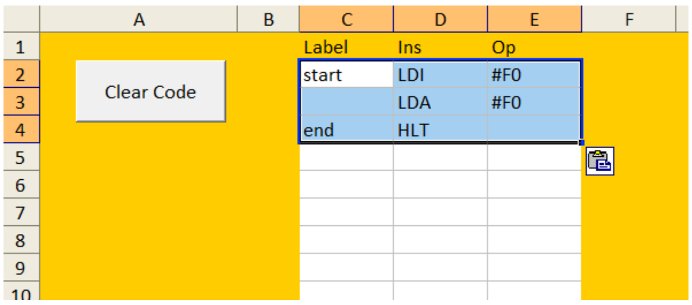
- Go to the Program Store sheet. Under “4.1 A Basic Operation” select all the cells containing content and copy them. Figure 5.



	A	B	C	D	E	F	G	H	I	J
1	Title:		B1 P1 4.1 A Basic Operation					B1 P1 7.1 A Simple Branch		
2		1					2			
3										
4	1		start	LDI	#F0				LDI	#02
5	2			LDA	#F0				SEC	
6	3		end	HLT				here	SBI	#01
7	4								BNZ	here
8	5								HLT	
9	6									
10	7									

Figure 5 Cells selected in the Program Store

- Go to the Source sheet and paste the cells in the corresponding positions. Figure 6. It is important that the same columns (i.e. Labels, Ins, Op) are used as the Program Store. (Alternatively, the text may be typed directly, but note the column position of the text).



	A	B	C	D	E	F
1			Label	Ins	Op	
2			start	LDI	#F0	
3				LDA	#F0	
4			end	HLT		
5						
6						
7						
8						
9						
10						

Figure 6 Cells pasted into Source sheet

- Go to the Assembler sheet. Click the “Assemble” button. The code will appear on the Assembler sheet.

- Go to the Processor sheet. Click the “Load Memory” button. The assembled code is loaded into the processor memory. Use any of the “Hex”, “Decimal” and “Disassemble” buttons to view the code in memory. These buttons can be used at any time.

- Click the “Reset” button. The processor is now ready to run the code.

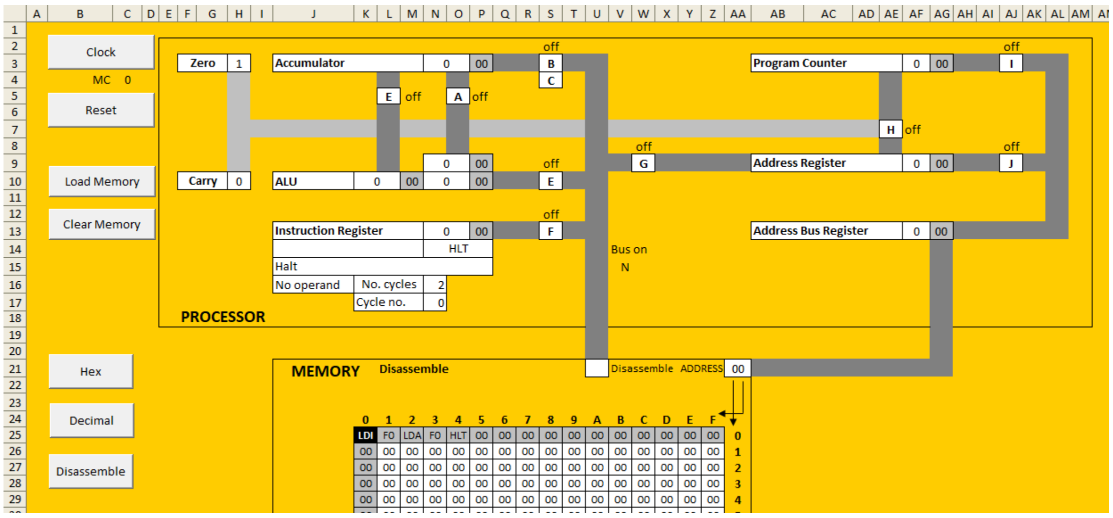
Each click of the “Clock” button runs another machine cycle. The processor is updated in the manner described in Book 1 except the emulator increments the Program Counter (where required) at the beginning of the next cycle so that the Program Counter can be fully observed in the current cycle.

Any of the programs in the Program Store can be copied into Source, assembled and run in the same way.

Instructions for writing programs are given in section 5.

4 Processor Sheet Description

The Processor sheet in the reset state is shown in Figure 7.



0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
LDI	F0	LDA	F0	HLT	00	00	00	00	00	00	00	00	00	00	00	0
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4

Figure 7 Reset state showing demonstration program disassembled

Notable features shown in Figure 7 are as follows.

- The count "MC" beneath the Clock button states the number of Machine Cycles performed i.e. the number of times Clock has been clicked since the reset state.

- Each gate A to H has the text "off" associated. This indicates the Gate is closed. When any Gate except B or C is opened during Fetch or Execute cycles the text changes to "on". For Gate B the text changes to "in" and for Gate C to "out".

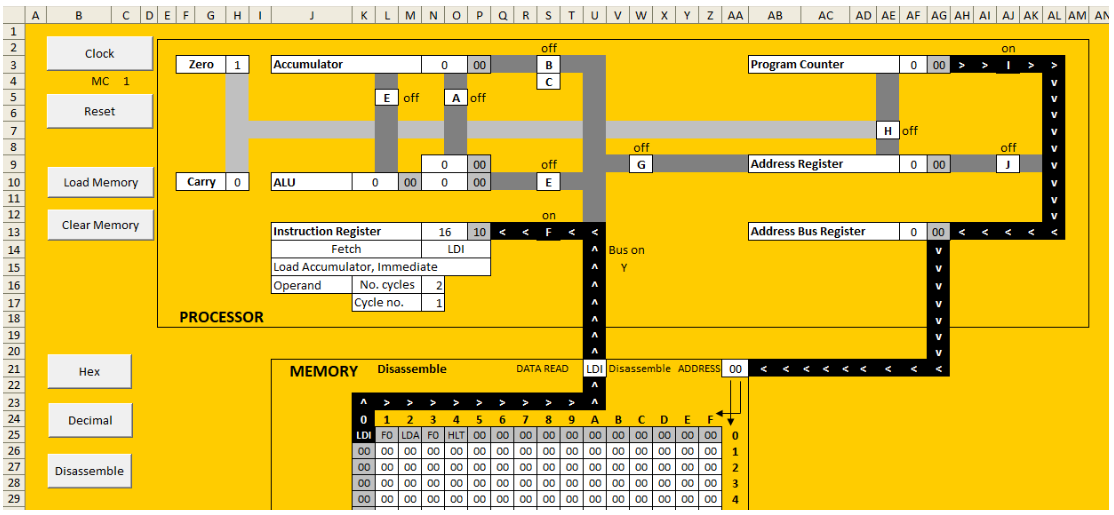
- The darker grey bars between registers and memory represent the paths by which byte information passes between them. The grey shading means no information is passing over the path.

- The path from the Address Bus Register to Memory is the address bus. The path from Gates B, C, E, F, G to Memory is the data bus. When there is data on the data bus the indicator "Bus on" changes to "Y".

- The hexadecimal content of the Address Bus Register is reflected in Memory in the byte "ADDRESS" when the address bus is on. The value decodes the row and column addresses in Memory (in grey shading) and hence the addressed byte (in black shading).

- The data bus content into Memory is shown. The content is in the same form as memory as selected by the Hex, Decimal and Disassemble buttons. The direction of the memory byte is also indicated by the indicator "DATA READ" for a memory read (Figure 8), "DATA WRITE" for a memory write, or blank (Figure 7) if memory is not accessed (i.e. the data bus is not on).

Figure 8 shows the paths for a typical Fetch cycle. Memory is shown in the "Disassemble" view.



0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
LDI	F0	LDA	F0	HLT	00	00	00	00	00	00	00	00	00	00	00	0
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4

Figure 8 Fetch LDI instruction

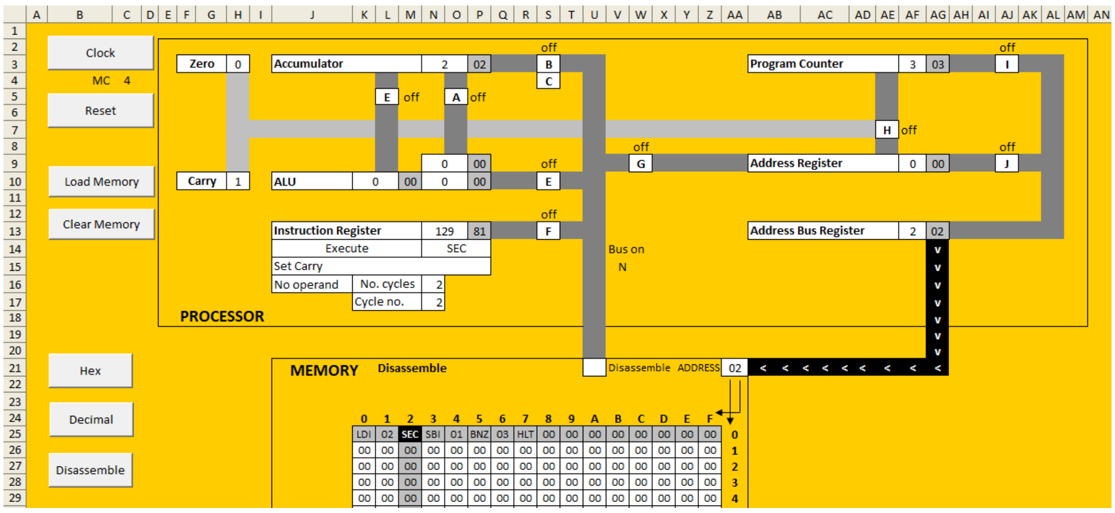
Notable features are as follows.

- Figure 8 shows the emulator demonstrating the Fetch cycle for an instruction. The Program Counter contains the address of the instruction loaded into the Instruction Register. The simple processor described in Book 1 and simulated in Book 3 increments the Program Counter at the end of the Fetch cycle. For clarity, when using the Program Counter the emulator always shows the incremented Program Counter at the beginning of the next cycle.

- Gates I and F are "on" and black shading with white chevrons indicates the direction of flow of information. The source data is copied into the destination.

- The instruction is copied into the Instruction Register. The emulator shows the instruction in its byte form (grey shading in hexadecimal) and the decimal equivalent to the left. The cells below indicate the cycle type (fetch or execute), the instruction mnemonic (LDI in the figure), the instruction name, whether the instruction includes an operand, the number of machine cycles required to complete the instruction and the count of machine cycles completed (Cycle no.).

Figure 9 shows the emulator executing an internal operation. The instruction sets the Carry flag.



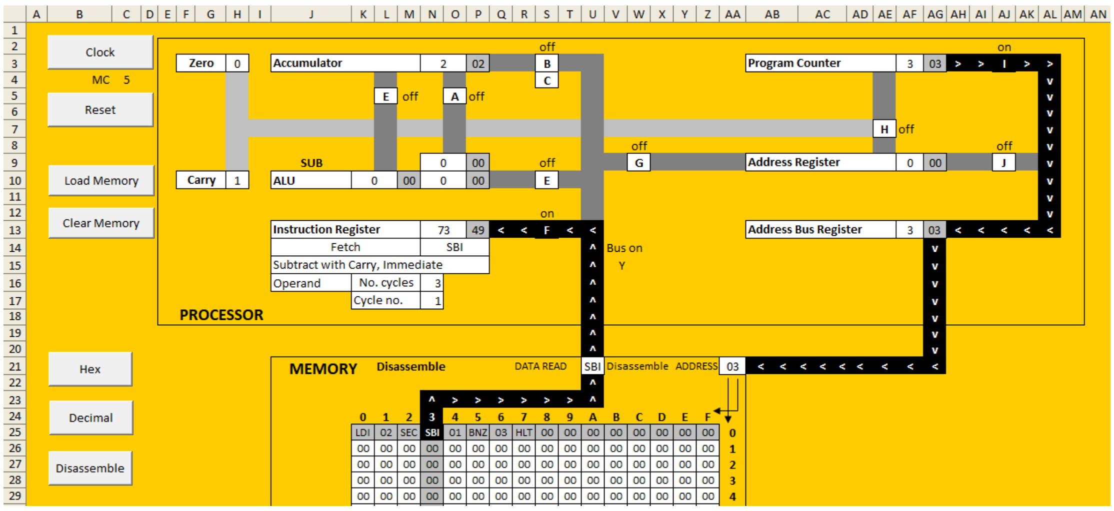
Zero 0
Carry 1
Accumulator 2 02
ALU 0 00 0 00
0 00
Program Counter 3 03
Address Register 0 00
Address Bus Register 2 02
Instruction Register 129 81
Execute SEC
Set Carry
No operand | No. cycles 2
Cycle no. 2
Disassemble ADDRESS 02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LDI	02	SEC	SBI	01	BNZ	03	HLT	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 9 Execute SEC Instruction

- The memory remains addressed by the Address Bus Register but the data bus is not on because the instruction execution is an internal operation.

Figure 10 shows an ALU instruction loaded into the Instruction Register.



Zero 0
Carry 1
Accumulator 2 02
ALU 0 00 0 00
0 00
Program Counter 3 03
Address Register 0 00
Address Bus Register 3 03
Instruction Register 73 49
Fetch SBI
Subtract with Carry, Immediate
Operand | No. cycles 3
Cycle no. 1
DATA READ SBI
Disassemble ADDRESS 03

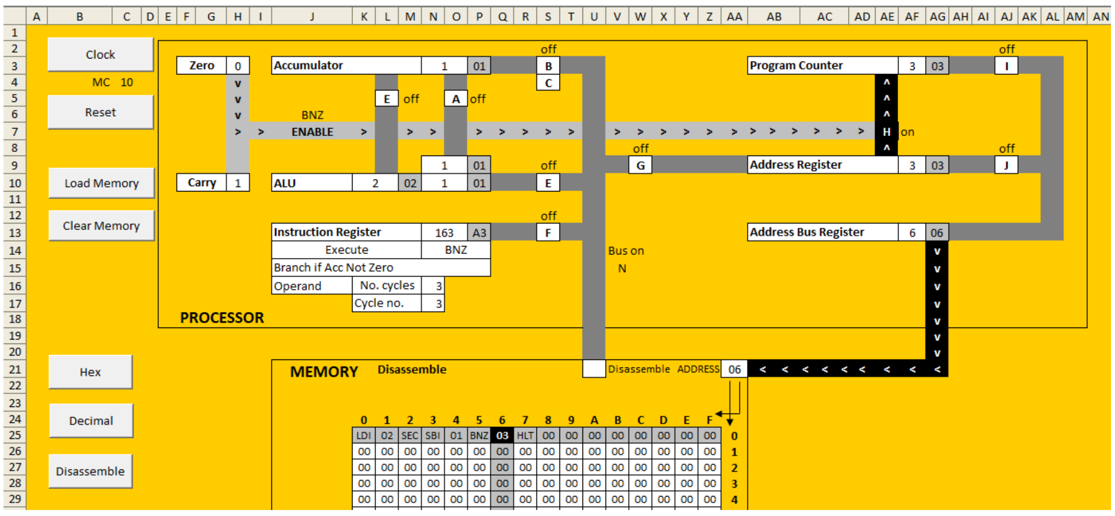
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LDI	02	SEC	SBI	01	BNZ	03	HLT	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 10 Fetch SBI

- The instruction concerns the ALU and the type is indicated by text appearing above the ALU as ADD, SUB, AND, OR or XOR as appropriate.

- The ALU register contains two inputs from the Accumulator and Memory (through Gate E) and one output linked to the Accumulator through Gate A. Each can be identified by the connecting data path.

Figure 11 shows the execution of a branch instruction where the condition enables Gate H to open.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	LDI	02	SEC	SBI	01	BNZ	03	HLT	00	00	00	00	00	00	00	00	0
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4

Figure 11 Execute BNZ branch condition met

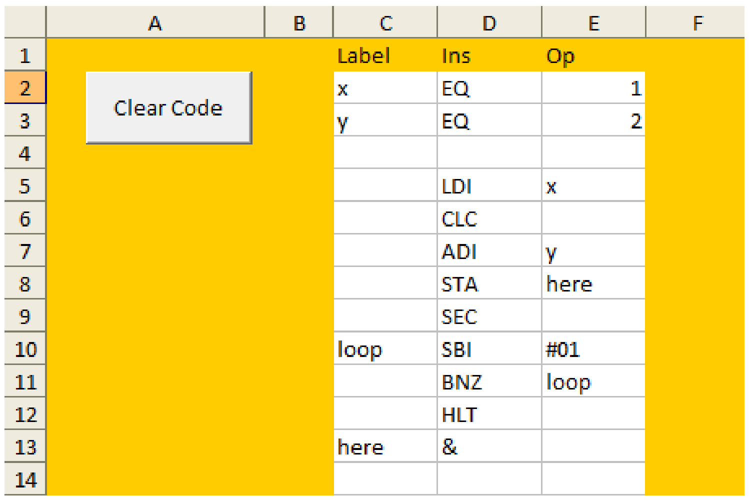
- Gate H control is shown by the light grey path connecting the flags to the Gate.
- The processing is an internal operation and the data bus is not on.
- When a branch instruction is loaded into the Instruction Register the instruction mnemonic is copied into the cell above the path between “Accumulator” and “ALU” as shown in Figure 11.
- The result of the condition is given in the cell below the instruction within the grey path. Gate H is “on” if the condition is “ENABLE” (Figure 11) and is “off” if “DISABLE” (not shown).
- The black chevrons indicate the flag involved and only progress to Gate H if the path is through “ENABLE”.

5 Writing Instruction Codes

This section contains a guide to writing simple processor programs. It is important to follow the guidance or the program may not build as expected, although it is possible to make program mistakes, of course.

Programs are best written on the Source sheet. When the program performs the task as expected then it can be copied into a blank repository in the Program Store if it is to be kept. To keep the program the emulator needs to be saved before it is closed. See section 1.4.

The Source sheet consists of three columns: Label (Label), Instruction (Ins), Operand (Op) as shown in Figure 12, which includes an example program.



	A	B	C	D	E	F
1			Label	Ins	Op	
2	Clear Code		x	EQ	1	
3			y	EQ	2	
4						
5				LDI	x	
6				CLC		
7				ADI	y	
8				STA	here	
9				SEC		
10			loop	SBI	#01	
11				BNZ	loop	
12				HLT		
13			here	&		
14						

Figure 12 Source sheet column headings and example program

5.1 *Labels*

The left column is reserved for labels. Only labels are expected in this column.

A label is a reference to one of three things:

- A constant defining the value of an operand
- A specific Instruction Code address
- A memory byte or byte list address

If the Instruction column contains the characters "EQ" then the label is a constant. The value is given in the Operand column as a decimal number. In general, wherever a constant appears in the Operand column in a program, a label should be used rather than using the number value directly. This makes it easier to change the value of a constant between program runs or if the need arises because only the value assigned to the constant by "EQ" needs to be changed. Figure 12 contains examples.

Sometimes the operand value is intrinsically fixed, such as a decrement. The value can be fixed by using a “#” sign preceding the hex value of the constant. Figure 12 contains an example on the subtract instruction. The “#” may be used during constant definition if the constant is provided in hex. Note that a decimal number in the Operand column is a label not a number, except where “EQ” is used in Instruction.

The program may contain branch instructions. In Machine Code the operand of the branch instruction requires the actual value of the memory location to which the program branches. The programmer does not know in advance the actual value. However, the instruction to branch to is known and its position in memory can be assigned a label. The written branch instruction operand uses the label and the assembler calculates the values when assembling the Machine Code. Therefore, the actual values are abstracted and the programmer does not need to assign the value directly. Generally, it is not important to know the actual byte address of the memory, just a reference that uniquely locates it in memory. Figure 12 contains an example (“loop”) of a branch label.

The program may make use of memory locations to load and store data. If the programmer requires a specific memory address to be used, the memory byte value can be assigned to a label as a constant and the constant used in the program. Alternatively, the address can be specified explicitly in the operand through the use of a “#” sign preceding the hex value of the address.

If the programmer does not require specific memory locations, then at the end of the program the remaining memory can be assigned using the “&” character in the Instruction column for each byte. Figure 12 contains an example on the store instruction (“here”).

Duplicate labels are not detected by the Assembler and no error is reported. The resulting operands will be incorrect for all instances. Labels must not be duplicated.

5.2 Instructions

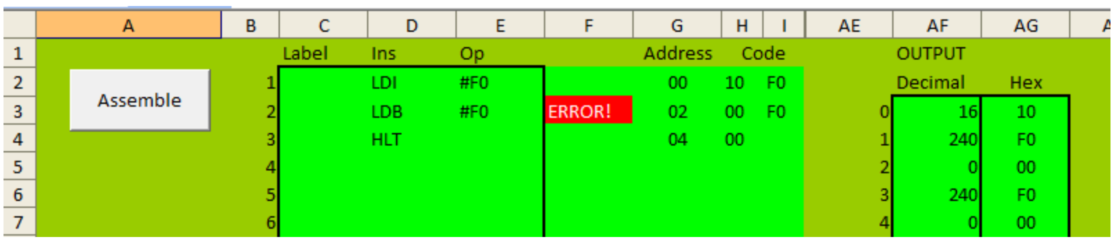
The centre column is reserved for instructions. Only instructions are expected in this column.

An instruction is one of three things:

- An instruction mnemonic
- An equate for fixing a constant value to a label
- A reference reserving a memory byte for data

The Instruction column must be formatted in one of the ways described here.

The column cell contains an Instruction mnemonic as defined for the simple processor. Instruction mnemonics are usually written using upper case characters but the case is not important to the Assembler. If the instruction is not recognised the Assembler shows an error at the line where the problem occurs. For example see Figure 13. The unknown instruction is assigned the value zero (i.e. HLT).

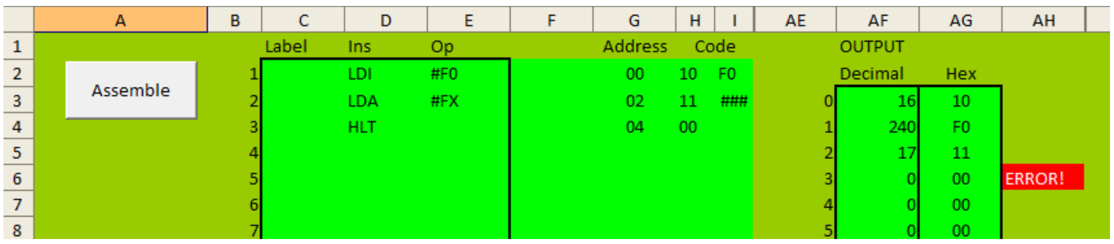


	A	B	C	D	E	F	G	H	I	AE	AF	AG
1			Label	Ins	Op		Address	Code			OUTPUT	
2	Assemble	1		LDI	#F0		00	10	F0		Decimal	Hex
3		2		LDB	#F0	ERROR!	02	00	F0	0	16	10
4		3		HLT			04	00		1	240	F0
5		4								2	0	00
6		5								3	240	F0
7		6								4	0	00

Figure 13 Example where Assembler does not recognise mnemonic

The special characters "EQ" which equates a label to a constant value. Using the label in an instruction operand assigns the value to that operand. The rules for EQ are as follows:

- A missing label will result in the assembler ignoring the line.
- A missing or non-numeric operand (except preceded with "#") will result in the value zero being assigned. The Label Table may show an error.
- A number is treated as a decimal number.
- Placing all the EQ statements at the beginning is generally convenient for the programmer but is not required by the Assembler.
- An entry preceded with "#" is treated as a hex number input. The hex number should consist of two hexadecimal characters. If the assembler detects that the hex number is incorrectly formatted then the value zero is assigned and the Assembler sheet indicates an error next to the operand in the Output columns. The entry under "Code" consists of a series of "#" characters. See example in Figure 14.



	A	B	C	D	E	F	G	H	I	AE	AF	AG	AH
1			Label	Ins	Op		Address	Code			OUTPUT		
2	Assemble	1		LDI	#F0		00	10	F0		Decimal	Hex	
3		2		LDA	#FX		02	11	###	0	16	10	
4		3		HLT			04	00		1	240	F0	
5		4								2	17	11	
6		5								3	0	00	ERROR!
7		6								4	0	00	
8		7								5	0	00	

Figure 14 Example of incorrectly formatted hex number

The special character "&" which reserves the next byte of memory for data storage. A label value or hex data entry in the Operand column is assigned to the byte. No entry in Operand assigns the value zero. An entry in the Label column may uniquely identify the byte or a list of bytes consisting of the labelled "&" and each following line entry "&" with no entry in the corresponding Label column. The "&" characters should only be used following all instruction mnemonics (i.e. program completion) otherwise the "Disassemble" view on the processor does not function properly.

5.3 Operands

The right column is reserved for operands. Only operands are expected in this column.

The Operand column entry is either a label, number, blank, or consists of a “#” character followed by a two-digit hex value. Decimal numbers appearing in the Operand column are regarded as labels and not numbers, except when “EQ” is used during constant definition. The usage of “EQ” and “#” is as defined previously.

Instructions which require an operand should have an entry in the Operand column, otherwise an error is shown. Conversely, if the instruction has no operand the Operand entry should be left blank otherwise an error is shown. (In all cases, the Assembler may assign a value to an entry).

The label refers to a defined memory space, constant or address of a referenced instruction, as appropriate to the context in which the label is defined. If the label is not recognised the value zero is assigned without a warning except the label will be missing from the Label Table.

An entry in Operand with no entry in Instruction is ignored by the Assembler.

The Assembler always produces some “Output” regardless of any errors. It is left to the programmer to identify and correct errors.

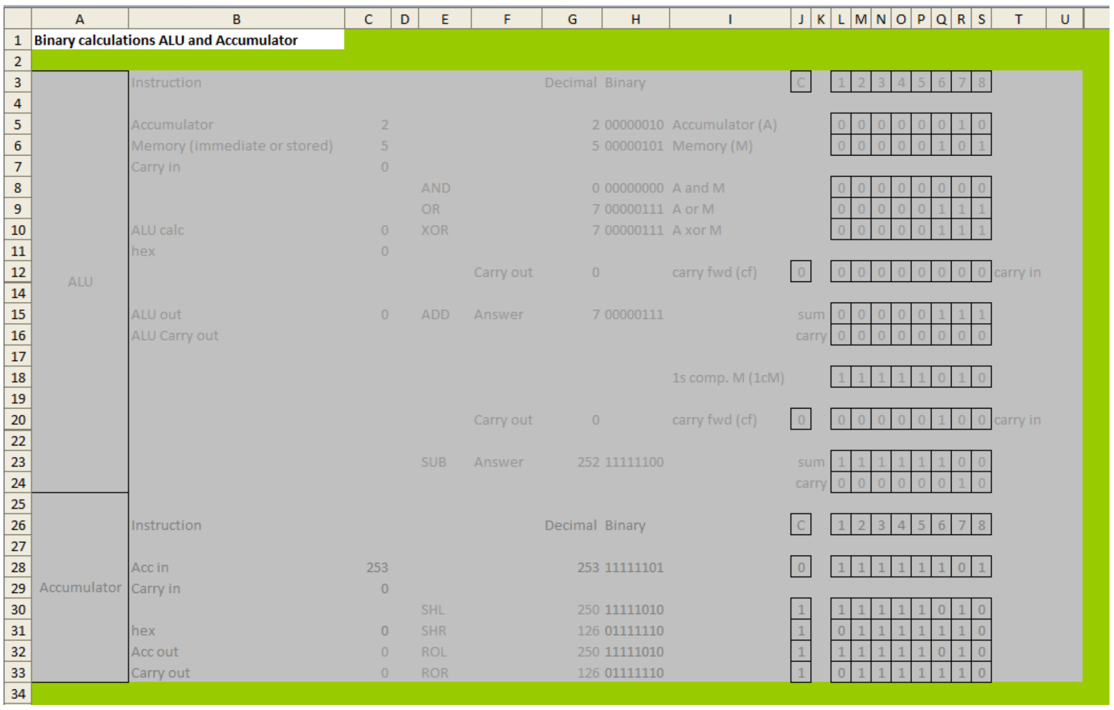
Blank lines are ignored by the Assembler and may be used to separate code lines if desired. An example occurs in Figure 12.

Writing programs using the mnemonics of the processor and a symbolic convention (as demonstrated here) is often referred to as programming in an “Assembler Language” or simply “Assembler”. Further examples are included in Book 3.

6 ALU Acc Sheet

The processor manipulates data through the ALU and Accumulator functions described in Book 1 Part 1. The emulator performs the calculations through the ALU Acc sheet and the computation can be viewed directly by selecting the sheet during the processing of a corresponding instruction. Note that this section includes some descriptions covered by Book 1 Part 2.

The ALU and Accumulator processing instructions are the arithmetic, logic, shift and rotate instructions. The ALU Acc sheet supports the instructions by continually updating its calculations for all the ALU and Accumulator processing functions. If the Instruction Register is not processing one of the ALU Acc instructions, then the ALU Acc sheet does not supply any results of its calculations to the Processor. This is indicated by shading the screen as shown in Figure 15.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	**Binary calculations ALU and Accumulator**																				
2																					
3		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	2				2	00000010	Accumulator (A)			0	0	0	0	0	0	1	0		
6		Memory (immediate or stored)	5				5	00000101	Memory (M)			0	0	0	0	0	1	0	1		
7		Carry in	0																		
8					AND		0	00000000	A and M			0	0	0	0	0	0	0	0		
9					OR		7	00000111	A or M			0	0	0	0	0	1	1	1		
10		ALU calc	0		XOR		7	00000111	A xor M			0	0	0	0	0	1	1	1		
11		hex	0																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	0	carry in	
14																					
15		ALU out	0		ADD	Answer	7	00000111			sum	0	0	0	0	0	1	1	1		
16		ALU Carry out									carry	0	0	0	0	0	0	0	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	1	0	1	0		
19																					
20						Carry out	0		carry fwd (cf)	0		0	0	0	0	0	1	0	0	carry in	
22																					
23					SUB	Answer	252	11111100			sum	1	1	1	1	1	1	0	0		
24											carry	0	0	0	0	0	0	1	0		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	253				253	11111101		0		1	1	1	1	1	1	0	1		
29	Accumulator	Carry in	0																		
30					SHL		250	11111010		1		1	1	1	1	1	0	1	0		
31		hex	0		SHR		126	01111110		1		0	1	1	1	1	1	1	0		
32		Acc out	0		ROL		250	11111010		1		1	1	1	1	1	0	1	0		
33		Carry out	0		ROR		126	01111110		1		0	1	1	1	1	1	1	0		
34																					

Figure 15 ALU Acc sheet not active

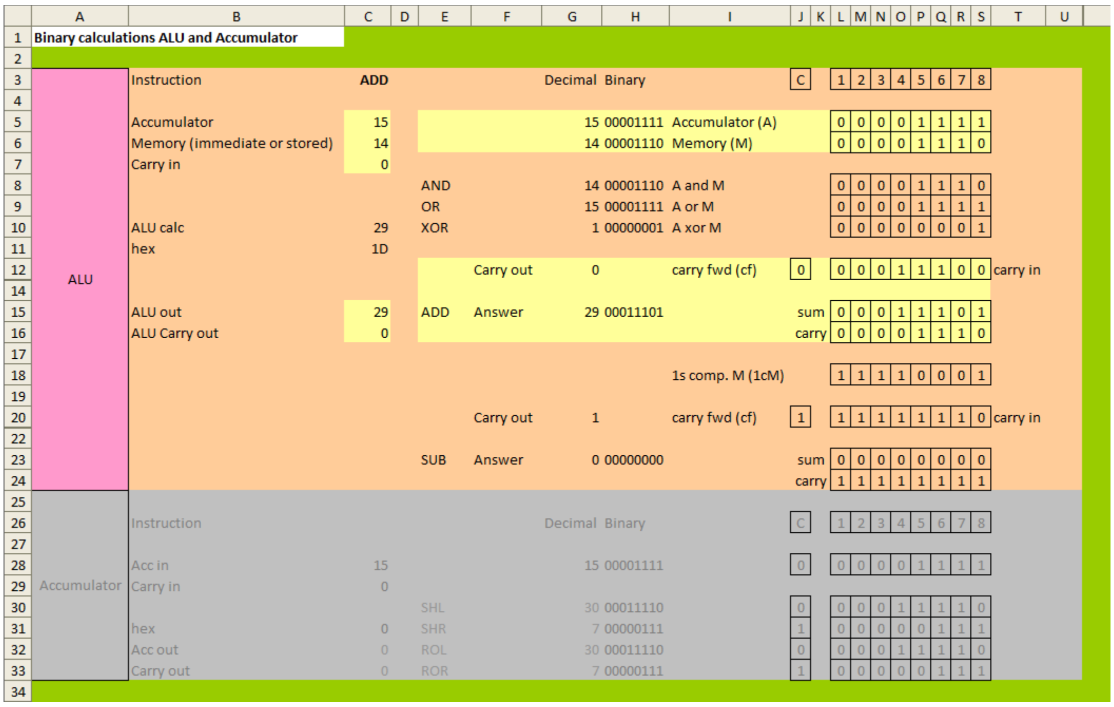
The sheet is divided into two sections: ALU calculations and Accumulator calculations. When the Instruction Register contains an instruction of either type the relevant part of the sheet shows the calculations performed for all instructions of the type. Within the section shown a particular set of calculations are highlighted which show the calculations for the actual instruction. It is the result of the highlighted calculations which appears as the sheet output which feed back to the Processor.

The sheet is highlighted when the instruction is fetched into the Instruction Register. The relevant data for the required calculations is only available when the instruction is executed. Therefore, the ALU Acc output is only relevant during the execute cycle of the instruction.

Example screens are as follows.

6.1 ALU

6.1.1 Addition



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3	ALU	Instruction	**ADD**				Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	15				15	00001111	Accumulator (A)			0	0	0	0	1	1	1	1		
6		Memory (immediate or stored)	14				14	00001110	Memory (M)			0	0	0	0	1	1	1	0		
7		Carry in	0																		
8					AND		14	00001110	A and M			0	0	0	0	1	1	1	0		
9					OR		15	00001111	A or M			0	0	0	0	1	1	1	1		
10		ALU calc	29		XOR		1	00000001	A xor M			0	0	0	0	0	0	0	1		
11		hex	1D																		
12						Carry out	0		carry fwd (cf)	0		0	0	0	1	1	1	0	0	carry in	
14																					
15		ALU out	29		ADD	Answer	29	00011101	sum			0	0	0	1	1	1	0	1		
16		ALU Carry out	0						carry			0	0	0	0	1	1	1	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	0	0	0	1		
19																					
20						Carry out	1		carry fwd (cf)	1		1	1	1	1	1	1	1	0	carry in	
22																					
23					SUB	Answer	0	00000000	sum			0	0	0	0	0	0	0	0		
24									carry			1	1	1	1	1	1	1	1		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	15				15	00001111		0		0	0	0	0	1	1	1	1		
29	Accumulator	Carry in	0																		
30					SHL		30	00011110		0		0	0	0	1	1	1	1	0		
31		hex	0		SHR		7	00000111		1		0	0	0	0	0	1	1	1		
32		Acc out	0		ROL		30	00011110		0		0	0	0	1	1	1	1	0		
33		Carry out	0		ROR		7	00000111		1		0	0	0	0	0	1	1	1		
34																					

Figure 16 ALU Acc sheet ALU active highlighting ADD

The ALU part of the sheet is highlighted. Within this part, all calculations relating to ALU functions are performed. The instruction is identified at the top of the sheet as “ADD” and only these calculations are relevant for addition and are further highlighted.

The “Accumulator (A)” and “Memory (M)” bytes are broken into the binary bits shown in the grids under the identifiers “1” to “8”. “1” is the most-significant bit on the sheet.

The “carry fwd (cf)” grid shows any Carry forwarded during the bit additions. Bit 8 of the grid is marked as “carry in” and is set to the value forwarded from the Processor sheet at the start of the calculation.

The grids “sum” and “carry” are derived by adding the A and M bits along with “carry fwd (cf)” bit as shown in Figure 17.

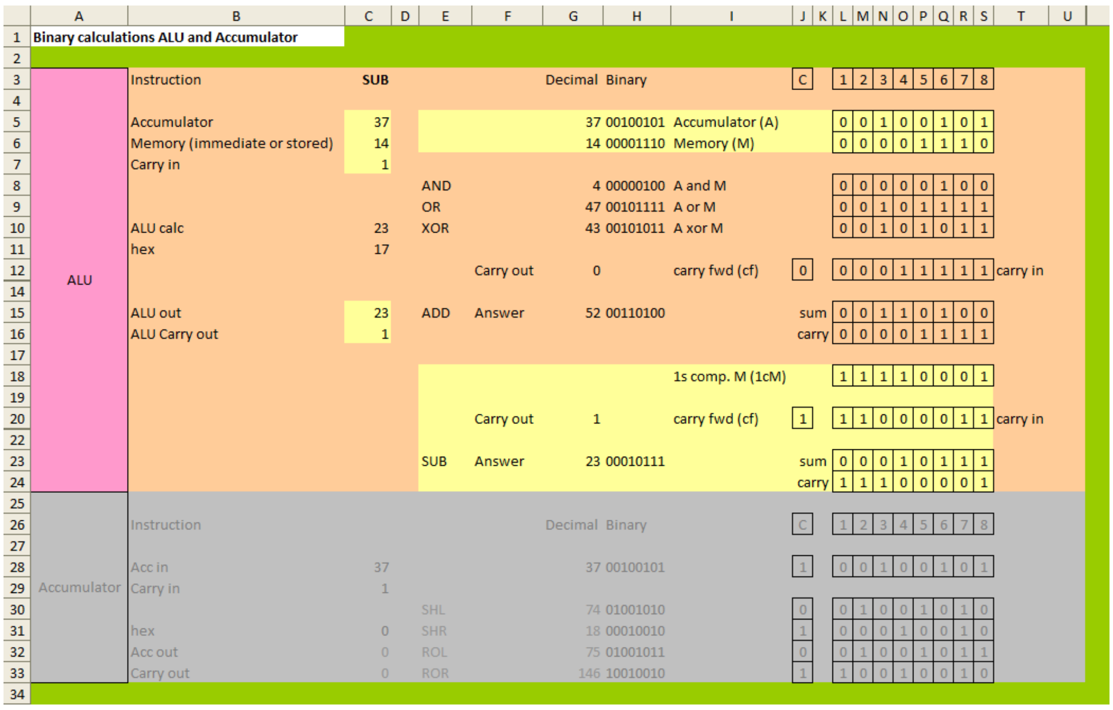
Inputs			Outputs	
A	M	cf	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 17 Addition of bits of byte

Processing proceeds by placing the Output Carry bit of position 8 into position 7 of the “Carry fwd (cf)” grid. The sequence is repeated for each bit down to position 1 with each individual carry rippling through the calculation. If the final addition (bit position 1) has the Carry out bit set, then the ALU Carry out bit is set. The Carry flag in the Processor is set to the value of ALU Carry out.

Note that the Figure 17 is the “Full Adder” described in Book 1 Part 2.

6.1.2 Subtraction



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	**Binary calculations ALU and Accumulator**																				
2																					
3		Instruction	**SUB**				Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	37				37	00100101	Accumulator (A)			0	0	1	0	0	1	0	1		
6		Memory (immediate or stored)	14				14	00001110	Memory (M)			0	0	0	0	1	1	1	0		
7		Carry in	1																		
8					AND		4	00000100	A and M			0	0	0	0	0	1	0	0		
9					OR		47	00101111	A or M			0	0	1	0	1	1	1	1		
10		ALU calc	23		XOR		43	00101011	A xor M			0	0	1	0	1	0	1	1		
11		hex	17																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	1	1	1	1	1	carry in	
14																					
15		ALU out	23		ADD	Answer	52	00110100			sum	0	0	1	1	0	1	0	0		
16		ALU Carry out	1								carry	0	0	0	0	1	1	1	1		
17																					
18									1s comp. M (1cM)			1	1	1	1	0	0	0	1		
19																					
20						Carry out	1		carry fwd (cf)	1		1	1	0	0	0	0	1	1	carry in	
22																					
23					SUB	Answer	23	00010111			sum	0	0	0	1	0	1	1	1		
24											carry	1	1	1	0	0	0	0	1		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	37				37	00100101		1		0	0	1	0	0	1	0	1		
29	Accumulator	Carry in	1																		
30					SHL		74	01001010		0		0	1	0	0	1	0	1	0		
31		hex	0		SHR		18	00010010		1		0	0	0	1	0	0	1	0		
32		Acc out	0		ROL		75	01001011		0		0	1	0	0	1	0	1	1		
33		Carry out	0		ROR		146	10010010		1		1	0	0	1	0	0	1	0		
34																					

Figure 18 ALU Acc sheet ALU active highlighting SUB

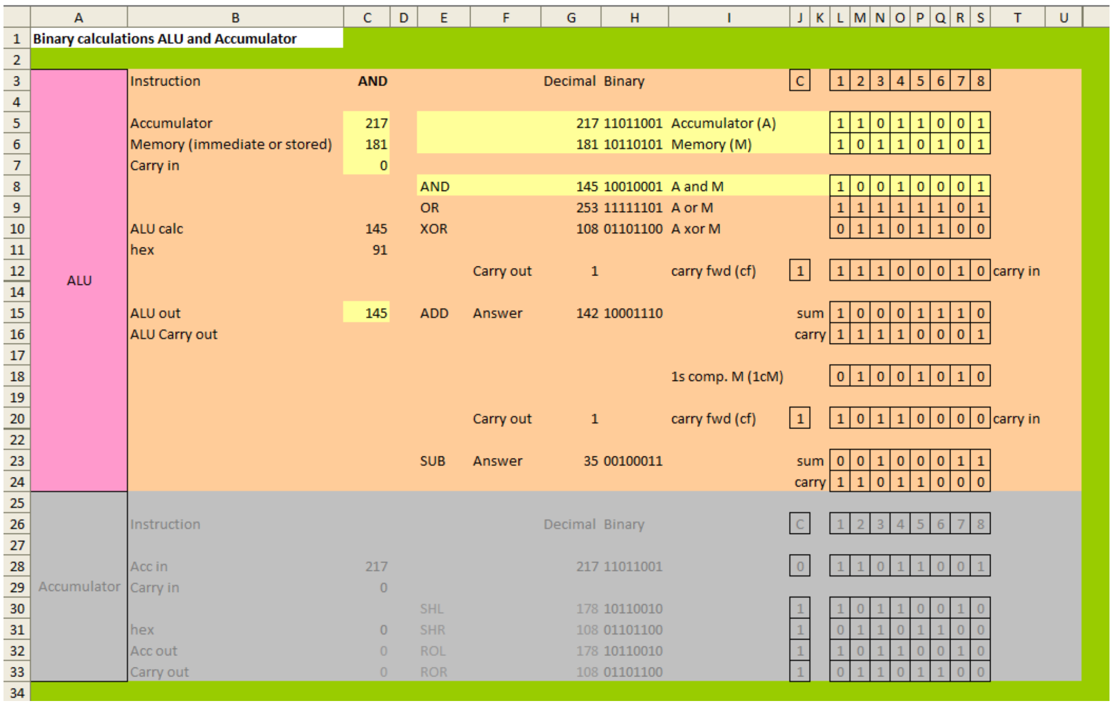
Figure 18 shows the highlight for subtraction. “Accumulator (A)” and “Memory (M)” bytes are shown along with calculations in the highlighted area for “SUB”.

The “Memory byte (M)” is ones-complemented in the grid “1s comp. M (1cM)” i.e. each bit is inverted. The grids “sum” and “carry” show the result of adding the bits in “A”, “1cM” and “cf” in the same procedure as for addition.

A full description of subtraction is provided in Book 1 Part 2.

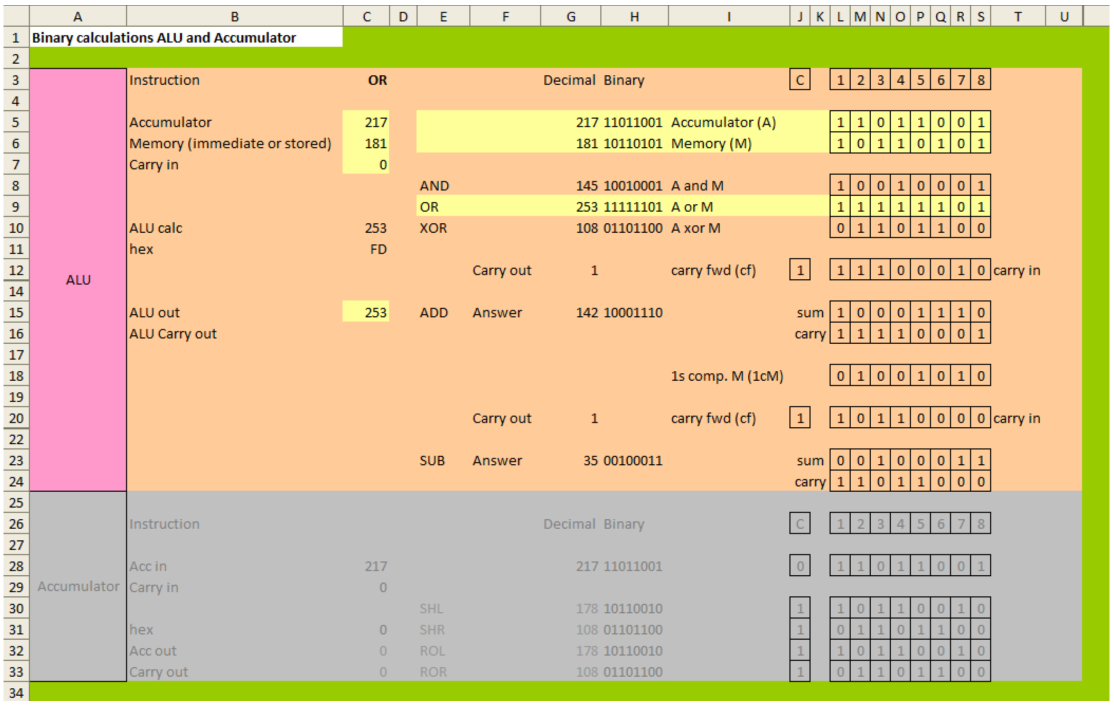
6.1.3 Logic

The screens highlighting each of the logical operators AND, OR and XOR are shown in the following figures.



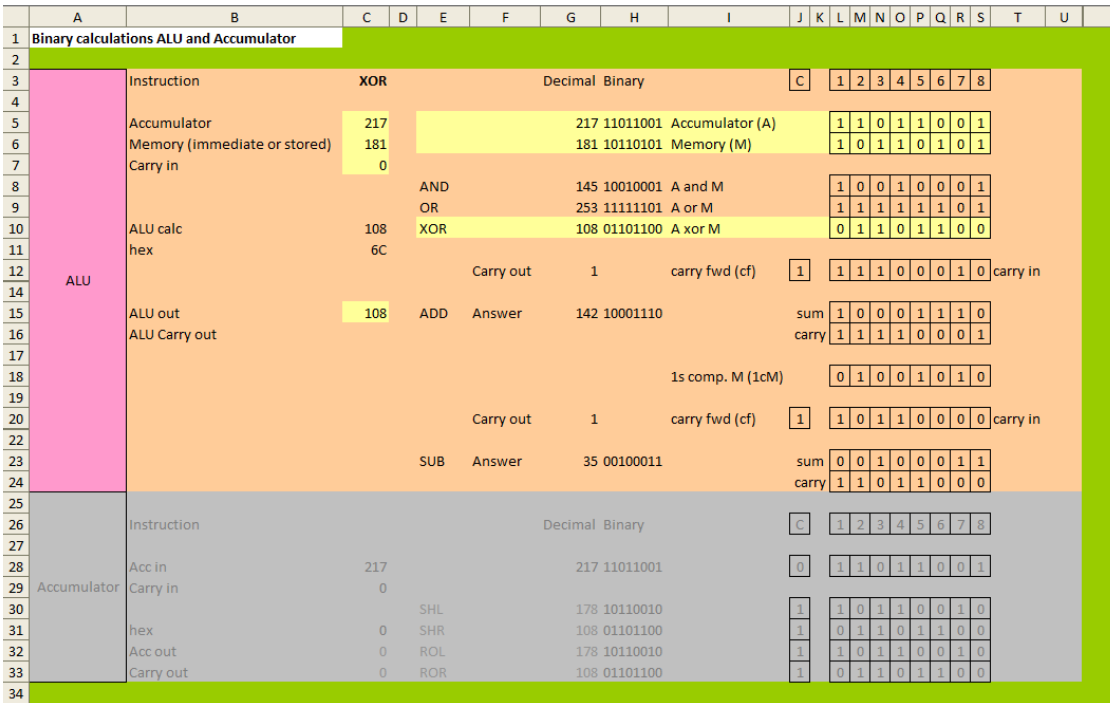
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction	AND				Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	217				217	11011001	Accumulator (A)			1	1	0	1	1	0	0	1		
6		Memory (immediate or stored)	181				181	10110101	Memory (M)			1	0	1	1	0	1	0	1		
7		Carry in	0																		
8					AND		145	10010001	A and M			1	0	0	1	0	0	0	1		
9					OR		253	11111101	A or M			1	1	1	1	1	1	0	1		
10		ALU calc	145		XOR		108	01101100	A xor M			0	1	1	0	1	1	0	0		
11		hex	91																		
12	ALU					Carry out	1		carry fwd (cf)	1		1	1	1	0	0	0	1	0	carry in	
14																					
15		ALU out	145		ADD	Answer	142	10001110			sum	1	0	0	0	1	1	1	0		
16		ALU Carry out									carry	1	1	1	1	0	0	0	1		
17																					
18									1s comp. M (1cM)			0	1	0	0	1	0	1	0		
19																					
20						Carry out	1		carry fwd (cf)	1		1	0	1	1	0	0	0	0	carry in	
22																					
23					SUB	Answer	35	00100011			sum	0	0	1	0	0	0	1	1		
24											carry	1	1	0	1	1	0	0	0		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	217				217	11011001		0		1	1	0	1	1	0	0	1		
29	Accumulator	Carry in	0																		
30					SHL		178	10110010		1		1	0	1	1	0	0	1	0		
31		hex	0		SHR		108	01101100		1		0	1	1	0	1	1	0	0		
32		Acc out	0		ROL		178	10110010		1		1	0	1	1	0	0	1	0		
33		Carry out	0		ROR		108	01101100		1		0	1	1	0	1	1	0	0		
34																					

Figure 19 ALU Acc sheet for AND



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction	OR				Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	217				217	11011001	Accumulator (A)			1	1	0	1	1	0	0	1		
6		Memory (immediate or stored)	181				181	10110101	Memory (M)			1	0	1	1	0	1	0	1		
7		Carry in	0																		
8					AND		145	10010001	A and M			1	0	0	1	0	0	0	1		
9					OR		253	11111101	A or M			1	1	1	1	1	1	0	1		
10		ALU calc	253		XOR		108	01101100	A xor M			0	1	1	0	1	1	0	0		
11		hex	FD																		
12	ALU					Carry out	1		carry fwd (cf)	1		1	1	1	0	0	0	1	0	carry in	
14																					
15		ALU out	253		ADD	Answer	142	10001110	sum			1	0	0	0	1	1	1	0		
16		ALU Carry out							carry			1	1	1	1	0	0	0	1		
17																					
18									1s comp. M (1cM)			0	1	0	0	1	0	1	0		
19																					
20						Carry out	1		carry fwd (cf)	1		1	0	1	1	0	0	0	0	carry in	
22																					
23					SUB	Answer	35	00100011	sum			0	0	1	0	0	0	1	1		
24									carry			1	1	0	1	1	0	0	0		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	217				217	11011001		0		1	1	0	1	1	0	0	1		
29	Accumulator	Carry in	0																		
30					SHL		178	10110010		1		1	0	1	1	0	0	1	0		
31		hex	0		SHR		108	01101100		1		0	1	1	0	1	1	0	0		
32		Acc out	0		ROL		178	10110010		1		1	0	1	1	0	0	1	0		
33		Carry out	0		ROR		108	01101100		1		0	1	1	0	1	1	0	0		
34																					

Figure 20 ALU Acc sheet for OR

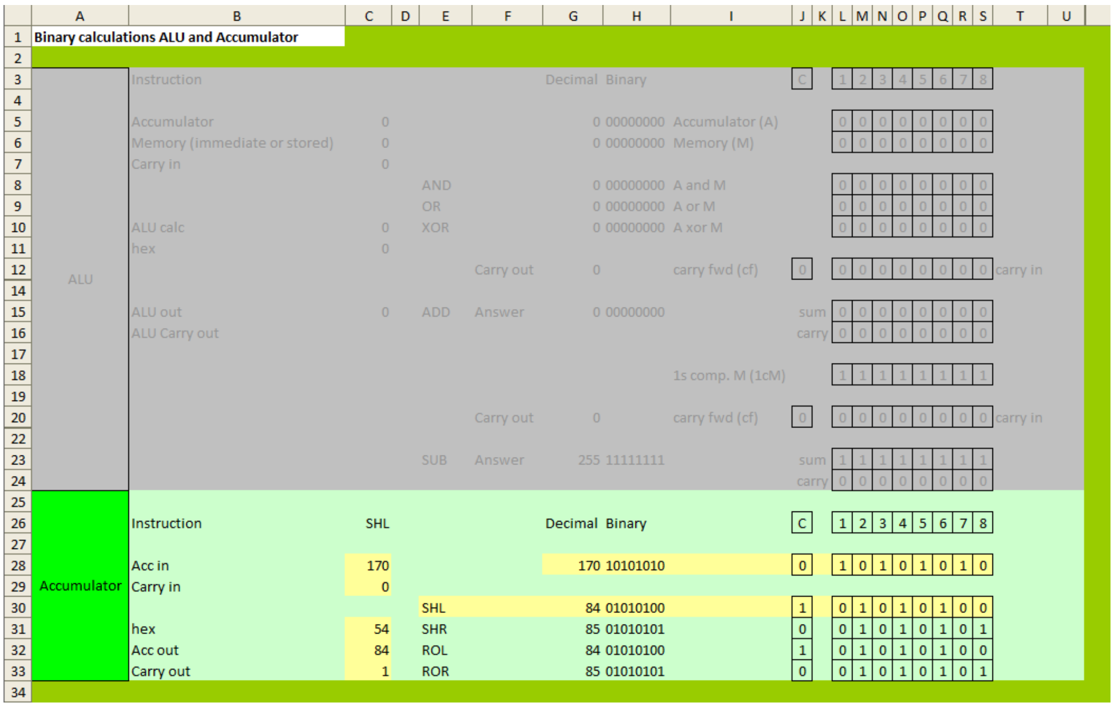


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction	XOR				Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	217				217	11011001	Accumulator (A)			1	1	0	1	1	0	0	1		
6		Memory (immediate or stored)	181				181	10110101	Memory (M)			1	0	1	1	0	1	0	1		
7		Carry in	0																		
8					AND		145	10010001	A and M			1	0	0	1	0	0	0	1		
9					OR		253	11111101	A or M			1	1	1	1	1	1	0	1		
10		ALU calc	108		XOR		108	01101100	A xor M			0	1	1	0	1	1	0	0		
11		hex	6C																		
12	ALU					Carry out	1		carry fwd (cf)	1		1	1	1	0	0	0	1	0	carry in	
14																					
15		ALU out	108		ADD	Answer	142	10001110	sum			1	0	0	0	1	1	1	0		
16		ALU Carry out							carry			1	1	1	1	0	0	0	1		
17																					
18									1s comp. M (1cM)			0	1	0	0	1	0	1	0		
19																					
20						Carry out	1		carry fwd (cf)	1		1	0	1	1	0	0	0	0	carry in	
22																					
23					SUB	Answer	35	00100011	sum			0	0	1	0	0	0	1	1		
24									carry			1	1	0	1	1	0	0	0		
25																					
26		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	217				217	11011001		0		1	1	0	1	1	0	0	1		
29	Accumulator	Carry in	0																		
30					SHL		178	10110010		1		1	0	1	1	0	0	1	0		
31		hex	0		SHR		108	01101100		1		0	1	1	0	1	1	0	0		
32		Acc out	0		ROL		178	10110010		1		1	0	1	1	0	0	1	0		
33		Carry out	0		ROR		108	01101100		1		0	1	1	0	1	1	0	0		
34																					

Figure 21 ALU Acc sheet for XOR

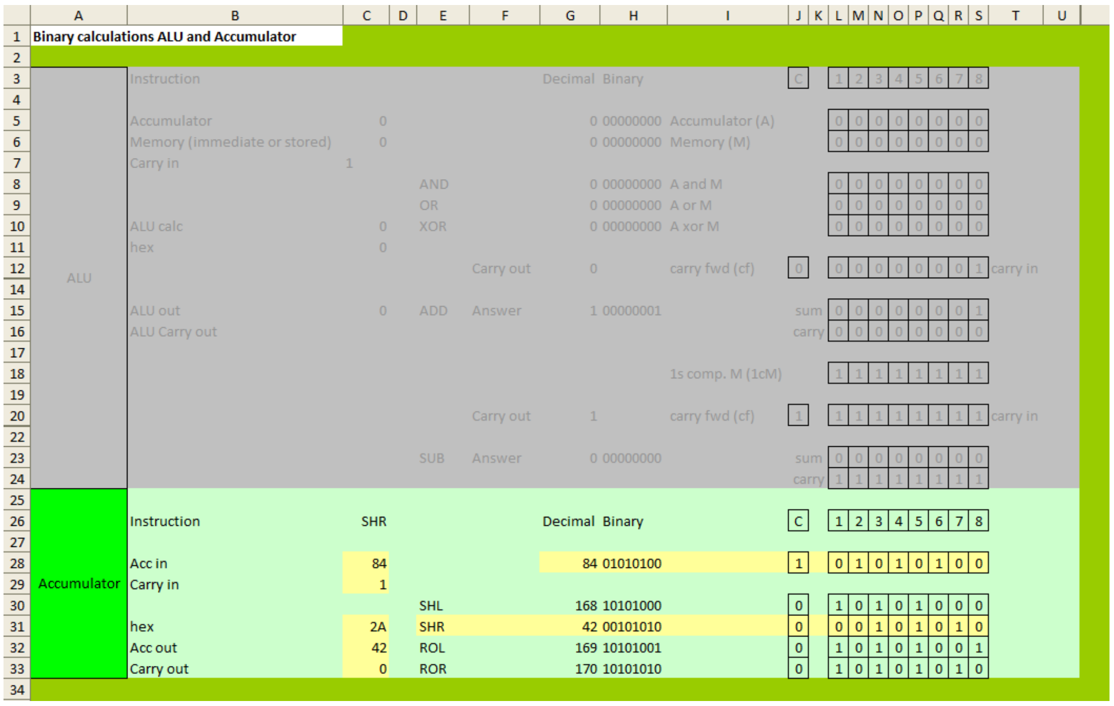
6.2 Accumulator

When the Instruction Register contains any of the Accumulator (shift/rotate) instructions the Accumulator section of the sheet shows the calculations for each of the four instructions. The specific instruction is highlighted. The shift and rotate sheets for each instruction appear as follows.



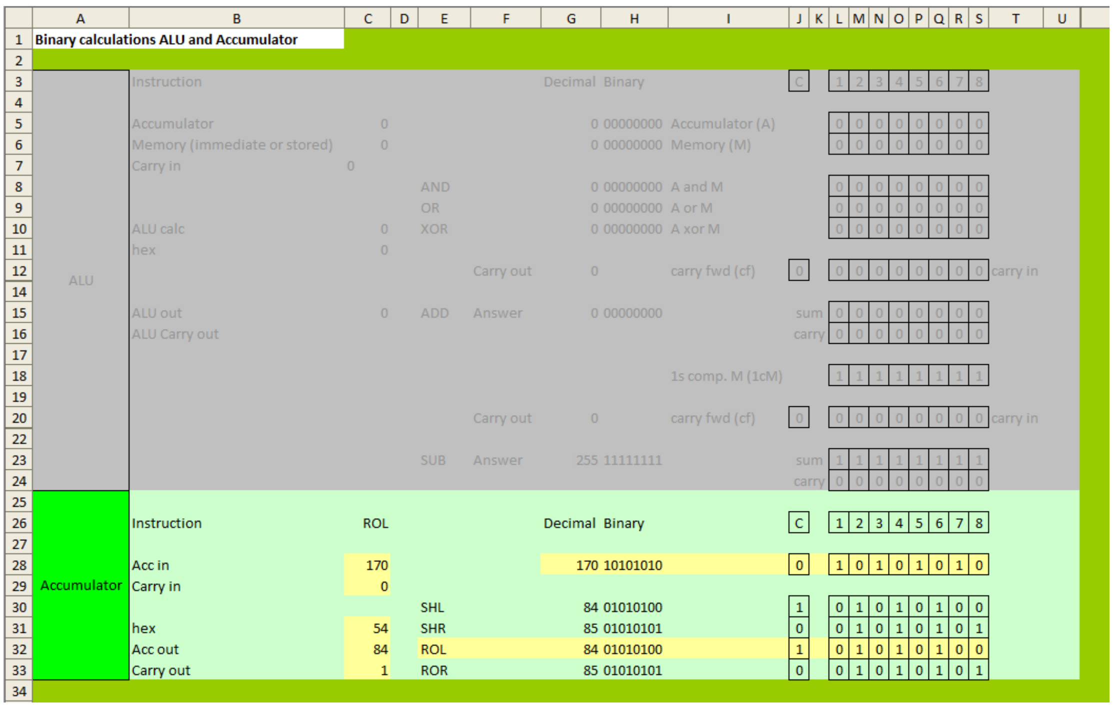
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	0				0	00000000	Accumulator (A)			0	0	0	0	0	0	0	0		
6		Memory (immediate or stored)	0				0	00000000	Memory (M)			0	0	0	0	0	0	0	0		
7		Carry in	0																		
8					AND		0	00000000	A and M			0	0	0	0	0	0	0	0		
9					OR		0	00000000	A or M			0	0	0	0	0	0	0	0		
10		ALU calc	0		XOR		0	00000000	A xor M			0	0	0	0	0	0	0	0		
11		hex	0																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	0	carry in	
14																					
15		ALU out	0		ADD	Answer	0	00000000		sum		0	0	0	0	0	0	0	0		
16		ALU Carry out								carry		0	0	0	0	0	0	0	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	1	1	1	1		
19																					
20						Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	0	carry in	
22																					
23					SUB	Answer	255	11111111		sum		1	1	1	1	1	1	1	1		
24										carry		0	0	0	0	0	0	0	0		
25																					
26		Instruction	SHL				Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	170				170	10101010		0		1	0	1	0	1	0	1	0		
29	Accumulator	Carry in	0																		
30					SHL		84	01010100		1		0	1	0	1	0	1	0	0		
31		hex	54		SHR		85	01010101		0		0	1	0	1	0	1	0	1		
32		Acc out	84		ROL		84	01010100		1		0	1	0	1	0	1	0	0		
33		Carry out	1		ROR		85	01010101		0		0	1	0	1	0	1	0	1		
34																					

Figure 22 ALU Acc sheet for SHL



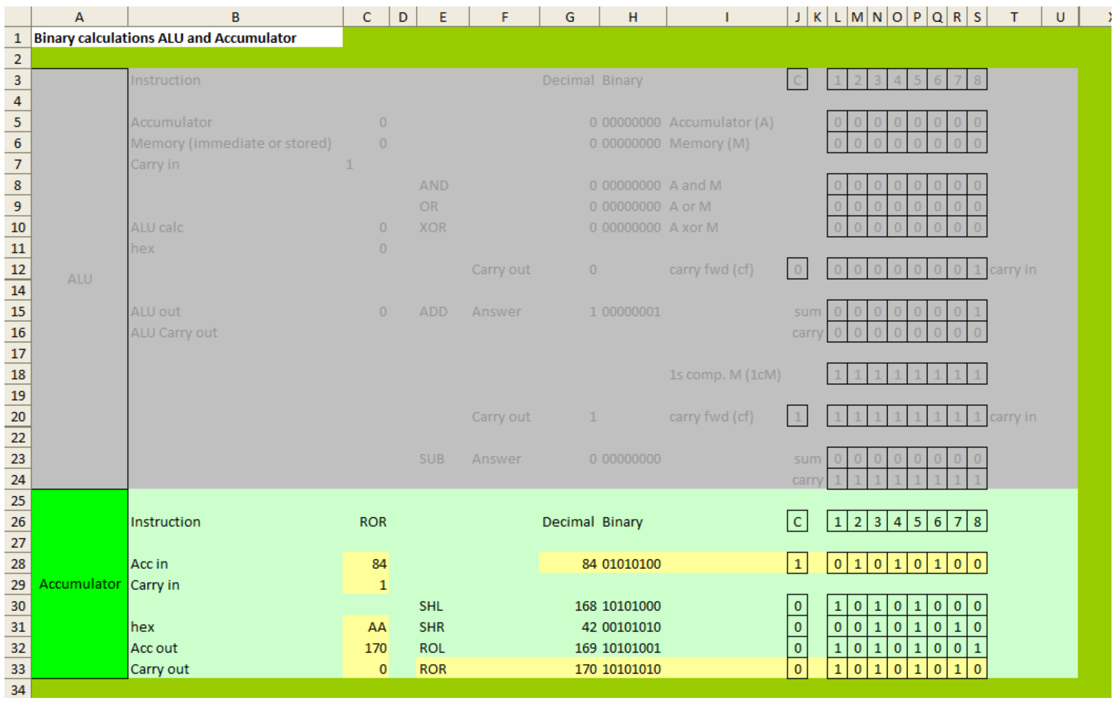
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	0				0	00000000	Accumulator (A)			0	0	0	0	0	0	0	0		
6		Memory (immediate or stored)	0				0	00000000	Memory (M)			0	0	0	0	0	0	0	0		
7		Carry in	1																		
8					AND		0	00000000	A and M			0	0	0	0	0	0	0	0		
9					OR		0	00000000	A or M			0	0	0	0	0	0	0	0		
10		ALU calc	0		XOR		0	00000000	A xor M			0	0	0	0	0	0	0	0		
11		hex	0																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	1	carry in	
14																					
15		ALU out	0		ADD	Answer	1	00000001		sum		0	0	0	0	0	0	0	1		
16		ALU Carry out								carry		0	0	0	0	0	0	0	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	1	1	1	1		
19																					
20						Carry out	1		carry fwd (cf)	1		1	1	1	1	1	1	1	1	carry in	
22																					
23					SUB	Answer	0	00000000		sum		0	0	0	0	0	0	0	0		
24										carry		1	1	1	1	1	1	1	1		
25																					
26		Instruction	SHR				Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	84				84	01010100		1		0	1	0	1	0	1	0	0		
29	Accumulator	Carry in	1																		
30					SHL		168	10101000		0		1	0	1	0	1	0	0	0		
31		hex	2A		SHR		42	00101010		0		0	0	1	0	1	0	1	0		
32		Acc out	42		ROL		169	10101001		0		1	0	1	0	1	0	0	1		
33		Carry out	0		ROR		170	10101010		0		1	0	1	0	1	0	1	0		
34																					

Figure 23 ALU Acc sheet for SHR



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	0				0	00000000	Accumulator (A)			0	0	0	0	0	0	0	0		
6		Memory (immediate or stored)	0				0	00000000	Memory (M)			0	0	0	0	0	0	0	0		
7		Carry in	0																		
8					AND		0	00000000	A and M			0	0	0	0	0	0	0	0		
9					OR		0	00000000	A or M			0	0	0	0	0	0	0	0		
10		ALU calc	0		XOR		0	00000000	A xor M			0	0	0	0	0	0	0	0		
11		hex	0																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	0	carry in	
14																					
15		ALU out	0		ADD	Answer	0	00000000			sum	0	0	0	0	0	0	0	0		
16		ALU Carry out									carry	0	0	0	0	0	0	0	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	1	1	1	1		
19																					
20						Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	0	carry in	
22																					
23					SUB	Answer	255	11111111			sum	1	1	1	1	1	1	1	1		
24											carry	0	0	0	0	0	0	0	0		
25																					
26		Instruction	ROL				Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	170				170	10101010		0		1	0	1	0	1	0	1	0		
29	Accumulator	Carry in	0																		
30					SHL		84	01010100		1		0	1	0	1	0	1	0	0		
31		hex	54		SHR		85	01010101		0		0	1	0	1	0	1	0	1		
32		Acc out	84		ROL		84	01010100		1		0	1	0	1	0	1	0	0		
33		Carry out	1		ROR		85	01010101		0		0	1	0	1	0	1	0	1		
34																					

Figure 24 ALU Acc sheet for ROL



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Binary calculations ALU and Accumulator																				
2																					
3		Instruction					Decimal	Binary		C		1	2	3	4	5	6	7	8		
4																					
5		Accumulator	0				0	00000000	Accumulator (A)			0	0	0	0	0	0	0	0		
6		Memory (immediate or stored)	0				0	00000000	Memory (M)			0	0	0	0	0	0	0	0		
7		Carry in	1																		
8					AND		0	00000000	A and M			0	0	0	0	0	0	0	0		
9					OR		0	00000000	A or M			0	0	0	0	0	0	0	0		
10		ALU calc	0		XOR		0	00000000	A xor M			0	0	0	0	0	0	0	0		
11		hex	0																		
12	ALU					Carry out	0		carry fwd (cf)	0		0	0	0	0	0	0	0	1	carry in	
14																					
15		ALU out	0		ADD	Answer	1	00000001			sum	0	0	0	0	0	0	0	1		
16		ALU Carry out									carry	0	0	0	0	0	0	0	0		
17																					
18									1s comp. M (1cM)			1	1	1	1	1	1	1	1		
19																					
20						Carry out	1		carry fwd (cf)	1		1	1	1	1	1	1	1	1	carry in	
22																					
23					SUB	Answer	0	00000000			sum	0	0	0	0	0	0	0	0		
24											carry	1	1	1	1	1	1	1	1		
25																					
26		Instruction	ROR				Decimal	Binary		C		1	2	3	4	5	6	7	8		
27																					
28		Acc in	84				84	01010100		1		0	1	0	1	0	1	0	0		
29	Accumulator	Carry in	1																		
30					SHL		168	10101000		0		1	0	1	0	1	0	0	0		
31		hex	AA		SHR		42	00101010		0		0	0	1	0	1	0	1	0		
32		Acc out	170		ROL		169	10101001		0		1	0	1	0	1	0	0	1		
33		Carry out	0		ROR		170	10101010		0		1	0	1	0	1	0	1	0		
34																					

Figure 25 ALU Acc sheet for ROR