# Tower of Eightness Reference Manual.

Author: Jennifer Gunn

Revised 27 May 2023.

# Table of Contents

# Chapter 0. Introduction to the Tower of Eightness.

## Welcome

You are now the proud owner of one of the most trailing edge pieces of technology. This sophisticated, versatile, and very capable engineering tool/toy is built for those with a need to play around, make and learn unimpeded by modern technologies closed-off and locked down architectures.

The Tower of Eightness has despite its trailing edge tech, enough capability to run an office, teach the basics of robotics, play games etc. The power is within you to make it do great things. Everything is documented where possible in order to make it the tinkerers heaven. There is just one thing I ask of you. When you make something beautiful, *share it!*

## A Guided tour

First, let's learn the pieces. The main unit consists of a 3D printed modular case which houses the backplane, of which there are versions with and without the W65C02S6-TPG14 Central Processing Unit and system clock generator. The W65C02S6-TPG14 is the beating heart of the system, and it can be considered to be the brain.

Then there is the memory card. On this card is 96 Kibibytes of RAM in three banks of 16 Kibibytes by 2 pages wide. At the top of memory space are two pages of ROM each also of 16 Kibibytes. This is where TowerOS, TowerBASIC and TowerTAPE FS live. This card is an essential without which the computer will not operate. It is recommended that this sits in the top slot if one intends to make frequent changes to the ROM.

Next up is the GPIO card. You need at least one of these to use TowerTAPE FS, SPI or $I^2C$ facilities. The first one should always be at $C020 base address.

Either an ACIA card or a dual ACIA card (recommended) is needed to provide one or more serial ports. All keyboard input is initially routed through the first serial port and on systems' that don't have some form of VDU output, this is how one sees the computers output.

In most cases you'll want a video output with some form of graphics, for this there is the ANSI video output card. With this, you'll be able to enjoy 80 columns output, a full PCDOS character set and some limited graphics capability. This card outputs either PAL or NTSC timed monochrome composite synch video.

Do you want to make some noise!!? Good, there's a soundcard based on the AY-3-8912A sound chip for that. It has a single stereo 3.5mm jack with line level output on it.

How about printing? For that there is the Centronics and Tower Peripheral Bus card. The printer port is especially useful. The Tower Peripheral Bus hardware is all there, but the firmware is a work in progress.

There is an up-and-coming much faster solid state storage card, this will operate with far greater reliability and speed than tape.

## Getting Up and Running

Firstly, we need power. On the back is a 2.1mm power jack that accepts anything from 8 to 12VDC positive tip with at least 1A. Whilst the comuter itself doesn't use all that, peripherals and future upgrades *might*. Plug that in but don't switch the computer on just yet.

Now, connect the ACIA port 1 to either a terminal/terminal emulator *or* the PS2 keyboard interface with the appropriate lead. For connecting to the keyboard interface, use the supplied SC1-K1 serial cable. It only fits one way round. For use with the terminal/terminal emulator you will need to make up a custom cable using the data in one of the ACIA adapter chapters. If using the PS2 to Keyboard adapter, you will need to supply power to that also. See the PS2 to Keyboard Adapter User Guide for further guidance there. We now have a way to type commands into the computer and if using it with a terminal, also see the output.

It is strongly recommended that you use the ANSI card with a composite capable display device. This wat you can enjoy its extra benefits. The output on that card is a single yellow RCA jack. This outputs either 50 or 60Hz monochrome video compatible with many modern television sets that support PAL or NTSC.

Should you wish to use TowerTAPE FS then you should now use a short and I do mean *short* 20-way ribbon cable to connect port B of the first GPIO card to the tape and joystick interface. This also supplies power to the tape and joystick interface, so you won't need yet another power supply.

These are considered the base options for the Tower of Eightness. Some might say sound is a base option, but that very much depends on your point of view. Now we can flip the power switch!


Figure 1 ToE Boot-screen

When starting from power-up you must always choose [C]old by pressing C. You will then be prompted to enter an amount of memory. For now just press [ENTER] and let the computer decide. This will drop you into TowerBASIC.

*Let the adventure begin!*

Let's get stuck in shall we? Good, TowerBASIC is a close fork of EhBASIC, it is therefore derived from EhBASIC, and due respect should be paid to its original and *late* author Lee Davidson. It is free to use but not copyright free. He would hate to see people abusing the spirit and intention of his work, so please don't. Now with that out of the way, BASICs of this type have a few things in common.

First, there are keywords such as **PRINT**, **IF** and **LET** which will appear throughout this manual as they do here. Take the following example for instance: -

```
10 PRINT "Hello world!"¶
20 GOTO 10¶
```

Typing this and then **RUN¶** tells the computer to print Hello world! repeatedly down the screen. To stop it doing so press escape or [Control]C, from now on shown ^C.

The numbers at the start of each line tell the computer where to put them in relation to each other, and each line is followed in numerical order. If you type a new line in with the same number, it replaces any existing one of the same line number with the new line. Try changing the message in the quotes on line 10 and see what happens!

Where you see **PRINT** that command prints whatever is between the quotes.

Line 20 does something different. It is an instruction to start running commands from the number following the **GOTO** command. In this simple form it must be followed by a line number that *must* exist or the computer will stop running your program with an **Undefined statement Error.**

Congratulations, you've just run your first program!

It would be a shame to have to type in an entire program consisting of many many lines of carefully crafted code, so connect your Tower of Eightness up to a tape recorder and whenever you have created something you'd rather not loose, just find a good spot on a tape, press record and type **SAVE** "<filename>"¶

When the computer has done saving you'll be greeted by the **Ready** prompt and be able to continue on with your day. Getting your program back is as simple as finding the right spot and typing **LOAD** "<optional filename>"¶, and playing the tape back. It may take a while to find the sweet spot on your tape recorders volume dial to get reliable loading of saved programs. Please refer to the chapter on the Tape and Joystick Interface for a fuller understanding.

## Variables and Math

So, computers are often insulted as being overgrown calculators, and whilst this is not true, they definitely owe their roots to the efforts of mathematicians and engineers.

The Tower of Eightness is quite capable of performing calculations with great speed and ease. Just to give you a taste, let's explore this a little.

Let us consider the following program: -

```
10 LET a=4
20 LET b=3
30 PRINT a*b
```

Upon being RUN this program produces the result 12. Nothing remarkable you might say, but the devil is in the detail for computers don't work in human numbers (decimal), they work in binary (noughts and ones, also known as base 2).

Line 10 assigns the variable 'a' with the value 4, whenever the computer then needs to use that number, you can use 'a' instead. In fact, you can change the contents of a whenever you like and even assign it the results of a previous calculation. The computer is taking an equation, in this case a times b and **PRINT**ing the result.

Line 20 is doing the same for the variable b, but not b=3.

Line 30 both performs the calculation and prints the result for the user to see.

For the Tower of Eightness, this was a trivial calculation that took such a short time I bet you didn't even perceive it. The computer has a good selection of operators.

It can add (+), subtract (-), multiply (*), divide (/), raise to a power (^) and much, much more. The TowerBASIC Reference Manual gives a full list of operators and functions to assist the programmer in making use of the computers calculating capability.

There are also string variables, and *arrays* of variables can exist as well.

Any variable name that ends in s '$' symbol is a *string* variable. Strings are any number of characters one after another and can have some limited operations after them. We shall just briefly touch upon them here. To know more, refer to the TowerBASIC Reference Manual.

```
N$="<Yourname>"
PRINT "Hello ";N$;". It's jolly nice to meet you!"
```

## Using the PRINT Command

You've seen PRINT in use already, but did you know, it can do *more?* In fact, when properly used, it can do most of what you might want to do with the Tower of Eightness display. Its purpose is more fully realised when you know the PC-DOS character set and special *control codes* that give it the ability to draw pixels, double height, double width, bold or otherwise characters sets and even change the number of columns on the screen.

PRINT can exist on its own to move down the screen by one line:-

```
10 PRINT "MENU"
20 PRINT
30 PRINT "Spam and eggs"
40 PRINT "Spam and chips"
50 PRINT "Spam, spam, spam, wonderful Spam!!"
```

**`PRINT`** can print multiple things together!

```
10 LET a=9
20 PRINT "The square root of";a;" is";SQR(a)
```

which yields the result of '**`The square root of 9 is 3`**' when run. We've also just encountered a function **`SQR()`** which *returns* the square root of whatever is in the parentheses. There are many such functions.

One such function especially useful with **`PRINT`** is **`CHR$()`**. It takes a number from 0 to 255 and passes the result as a single character. If you look at the PC-DOS character set, listed at the back of this manual, you will see that each and every character has a unique number associated with it, and better yet, there's also a whole bunch of fancy symbols to play with as well! A word of caution though, it is quite possible to get the screen in a terrible mess whilst exploring this, so don't forget that if you press the reset button and choose [W]arm start, you'll have a nice readable display to call home again.

PRINT can also produce columnated figures. Try the following: -

PRINT 1,2,3,4


Figure 2 Tabulated printing in TowerBASIC.

As you can see from the above image, there are fixed tabs where the numbers appear. This is useful for making your information sit more readably and presentably.

There are a couple of useful functions for moving whatever you print along too. Following the print statement or after a ',' or ';' within the **`PRINT`** statement, you can include either a **`SPC(x)`** or **`TAB(x)`** function where x is a positive numeric value.

**`SPC(x)`** works by moving the print position x characters along, but **`TAB(x)`** attempts to get to column x so long as that column hasn't already been passed.

Finally, if you end your print statement with a semicolon, a carriage return is not issued. That is to say that the next thing starts *immediately* after the last print.

# Chapter 1. The Zero Page.

## Tower of Eightness Zero Page Usage

| | | |
|---|---|---|
| LAB_WARM | $00 | BASIC warm start entry point |
| Wrmjpl | $01 | BASIC warm start vector jump low byte |
| Wrmjph | $02 | BASIC warm start vector jump high byte |
| | | |
| Usrjmp | $0A | USR function JMP address |
| Usrjpl | $0B | USR function JMP vector low byte |
| Usrjph | $0C | USR function JMP vector high byte |
| Nullct | $0D | nulls output after each line |
| TPos | $0E | BASIC terminal position byte |
| TWidth | $0F | BASIC terminal width byte |
| Iclim | $10 | Input column limit |
| Itempl | $11 | Temporary integer low byte |
| Itemph | $12 | Temporary integer high byte |
| nums_1 | $11 | Number to bin/hex string convert MSB |
| nums_2 | $12 | Number to bin/hex string convert |
| nums_3 | $13 | Number to bin/hex string convert LSB |
| | | |
| Srchc | $5B | Search character |
| Temp3 | $5B | Temp byte used in number routines |
| Scnquo | $5C | Scan-between-quotes flag |
| Asrch | $5C | Alt search character |
| | | |
| XOAw_l | $5B | eXclusive OR, OR and word low byte |
| XOAw_h | $5C | eXclusive OR, OR and AND word high byte |
| | | |
| Ibptr | $5D | Input buffer pointer |
| Dimcnt | $5D | # of dimensions |
| Tindx | $5D | Token index |
| | | |
| Defdim | $5E | Default DIM flag |
| Dtypef | $5F | Data type flag, $FF=string, $00=numeric |
| Oquote | $60 | Open quote flag (b7) (Flag: DATA scan; LIST quote; memory) |
| Gclctd | $60 | Garbage collected flag |
| Sufnxf | $61 | Subscript/FNX flag, 1xxx xxx = FN(0xxx xxx) |
| Imode | $62 | Input mode flag, $00=INPUT, $80=READ |
| | | |
| Cflag | $63 | Comparison evaluation flag |
| | | |
| TabSiz | $64 | TAB step size (was input flag) |
| | | |
| next_s | $65 | Next descriptor stack address |

These two bytes form a word pointer to the item currently on top of the descriptor stack.

| | | |
|---|---|---|
| last_s | $66 | Last descriptor stack address low byte |
| last_sh | $67 | Last descriptor stack address high byte (always $00) |
| des_sk | $68 | Descriptor stack start address (temp strings) |

| | | |
|---|---|---|
| | $70 | End of descriptor stack |
| | | |
| ut1_pl | $71 | Utility pointer 1 low byte |
| ut1_ph | $72 | Utility pointer 1 high byte |
| ut2_pl | $73 | Utility pointer 2 low byte |
| ut2_ph | $74 | Utility pointer 2 high byte |
| | | |
| Temp_2 | $71 | Temp byte for block move |
| | | |
| FACt_1 | $75 | FAC temp mantissa1 |
| FACt_2 | $76 | FAC temp mantissa2 |
| FACt_3 | $77 | FAC temp mantissa3 |
| | | |
| dims_l | $76 | Array dimension size low byte |
| dims_h | $77 | Array dimension size high byte |
| | | |
| TempB | $78 | Temp page 0 byte |
| | | |
| Smeml | $79 | Start of mem low byte (Start-of-Basic) |
| Smemh | $80 | Start of mem high byte (Start-of-Basic) |
| Svarl | $7B | Start of vars low byte (Start-of-Variables) |
| Svarh | $7C | Start of vars high byte (Start-of-Variables) |
| Sarryl | $7D | Var mem end low byte (Start-of-Arrays) |
| Sarryh | $7E | Var mem end high byte (Start-of-Arrays) |
| Earryl | $7F | Array mem end low byte (End-of-Arrays) |
| Earryh | $80 | Array mem end high byte (End-of-Arrays) |
| Sstorl | $81 | String storage low byte (String storage (moving down)) |
| Sstorh | $82 | String storage high byte (String storage (moving down)) |
| Sutill | $83 | String utility ptr low byte |
| Sutilh | $84 | String utility ptr high byte |
| Ememl | $85 | End of mem low byte (Limit-of-memory) |
| Ememh | $86 | End of mem high byte (Limit-of-memory) |
| Clinel | $87 | Current line low byte (Basic line number) |
| Clineh | $88 | Current line high byte (Basic line number) |
| Blinel | $89 | Break line low byte (Previous Basic line number) |
| Blineh | $8A | Break line high byte (Previous Basic line number) |
| | | |
| Cpntrl | $8B | Continue pointer low byte |
| Cpntrh | $8C | Continue pointer high byte |
| | | |
| Dlinel | $8D | Current DATA line low byte |
| Dlineh | $8E | Current DATA line high byte |
| | | |
| Dptrl | $8F | DATA pointer low byte |
| Dptrh | $90 | DATA pointer high byte |
| | | |
| Rdptrl | $91 | Read pointer low byte |
| Rdptrh | $92 | Read pointer high byte |
| | | |
| Varnm1 | $93 | Current var name 1st byte |

| | | |
|---|---|---|
| Varnm2 | $94 | Current var name 2nd byte |
| Cvaral | $95 | Current var address low byte |
| Cvarah | $96 | Current var address high byte |
| Frnxtl | $97 | Var pointer for FOR/NEXT low byte |
| Frnxth | $98 | Var pointer for FOR/NEXT high byte |
| Tidx1 | $97 | Temp line index |
| Lvarpl | $97 | Let var pointer low byte |
| Lvarph | $98 | Let var pointer high byte |
| Prstk | $99 | Precedence stacked flag |
| comp_f | $9B | compare function flag, bits 0,1 and 2 used<br>Bit 2 set if ><br>Bit 1 set if =<br>Bit 0 set if < |
| func_l | $9C | Function pointer low byte |
| func_h | $9D | Function pointer high byte |
| garb_l | $9C | Garbage collection working pointer low byte |
| garb_h | $9D | Garbage collection working pointer high byte |
| des_2l | $9E | String descriptor_2 pointer low byte |
| des_2h | $9F | String descriptor_2 pointer high byte |
| g_step | $A0 | Garbage collect step size |
| Fnxjmp | $A1 | Jump vector for functions |
| Fnxjpl | $A2 | Functions jump vector low byte |
| Fnxjph | $A3 | Functions jump vector high byte |
| g_indx | $A2 | Garbage collect temp index |
| FAC2_r | $A3 | FAC2 rounding byte |
| Adatal | $A4 | Array data pointer low byte |
| Adatah | $A5 | Array data pointer high byte |
| Nbendl | $A4 | New block end pointer low byte |
| Nbendh | $A5 | New block end pointer high byte |
| Obendl | $A6 | Old block end pointer low byte |
| Obendh | $A7 | Old block end pointer high byte |
| Numexp | $A8 | String to float number exponent count |
| Expcnt | $A9 | String to float exponent count |

| | | |
|---|---|---|
| Numbit | $A8 | Bit count for array element calculations |
| | | |
| Numdpf | $AA | String to float decimal point flag |
| Expneg | $AB | String to float eval exponent -ve flag |
| | | |
| Astrtl | $AA | Array start pointer low byte |
| Astrth | $AB | Array start pointer high byte |
| | | |
| Histrl | $AA | Highest string low byte |
| Histrh | $AB | Highest string high byte |
| | | |
| Baslnl | $AA | BASIC search line pointer low byte |
| Baslnh | $AB | BASIC search line pointer high byte |
| | | |
| Fvar_l | $AA | Find/found variable pointer low byte |
| Fvar_h | $AB | Find/found variable pointer high byte |
| | | |
| Ostrtl | $AA | Old block start pointer low byte |
| Ostrth | $AB | Old block start pointer high byte |
| | | |
| Vrschl | $AA | Variable search pointer low byte |
| Vrschh | $AB | Variable search pointer high byte |
| | | |
| FAC1_e | $AC | FAC1 exponent |
| FAC1_1 | $AD | FAC1 mantissa1 |
| FAC1_2 | $AE | FAC1 mantissa2 |
| FAC1_3 | $AF | FAC1 mantissa3 |
| FAC1_s | $B0 | FAC1 sign (b7) |
| | | |
| str_ln | $AC | String length |
| str_pl | $AD | String pointer low byte |
| str_ph | $AE | String pointer high byte |
| | | |
| des_pl | $AE | String descriptor pointer low byte |
| des_ph | $AF | String descriptor pointer high byte |
| | | |
| mids_l | $AF | MID$ string temp length byte |
| | | |
| negnum | $B1 | String to float eval -ve flag |
| numcon | $B1 | Series evaluation constant count |
| | | |
| FAC1_o | $B2 | FAC1 overflow byte |
| | | |
| FAC2_e | $B3 | FAC2 exponent |
| FAC2_1 | $B4 | FAC2 mantissa1 |
| FAC2_2 | $B5 | FAC2 mantissa2 |
| FAC2_3 | $B6 | FAC2 mantissa3 |
| FAC2_s | $B7 | FAC2 sign (b7) |
| | | |
| FAC_sc | $B8 | FAC sign comparison, Acc#1 vs #2 |
| FAC1_r | $B9 | FAC1 rounding byte |

| | | |
|---|---|---|
| ssptr_l | $B8 | String start pointer low byte |
| ssptr_h | $B9 | String start pointer high byte |
| | | |
| sdescr | $B8 | String descriptor pointer |
| | | |
| csidx | $BA | Line crunch save index |
| Asptl | $BA | Array size/pointer low byte |
| Aspth | $BB | Array size/pointer high byte |
| | | |
| Btmpl | $BA | BASIC pointer temp low byte |
| Btmph | $BB | BASIC pointer temp low byte |
| | | |
| Cptrl | $BA | BASIC pointer temp low byte |
| Cptrh | $BB | BASIC pointer temp low byte |
| | | |
| Sendl | $BA | BASIC pointer temp low byte |
| Sendh | $BB | BASIC pointer temp low byte |
| LAB_IGBY | $BC | Get next BASIC byte subroutine |
| | | |
| LAB_GBYT | $C2 | Get current BASIC byte subroutine |
| Bpntrl | $C3 | BASIC execute (get byte) pointer low byte |
| Bpntrh | $C4 | BASIC execute (get byte) pointer high byte |
| | | |
| | $D7 | end of get BASIC char subroutine |
| | | |
| Rbyte4 | $D8 | Extra PRNG byte |
| Rbyte1 | $D9 | Most significant PRNG byte |
| Rbyte2 | $DA | Middle PRNG byte |
| Rbyte3 | $DB | Least significant PRNG byte |
| | | |
| NmiBase | $DC | NMI handler enabled/setup/triggered flags |
| | | |
| | | bit function |
| | | 7 interrupt enabled |
| | | 6 interrupt setup |
| | | 5 interrupt happened |
| | | |
| | $DD | NMI handler addr low byte |
| | $DE | NMI handler addr high byte |
| IrqBase | $DF | IRQ handler enabled/setup/triggered flags |
| | $E0 | IRQ handler addr low byte |
| | $E1 | IRQ handler addr high byte |
| | | |
| | $E2 | TPB card temporary location |
| | $E3 | TPB card temporary location |
| | $E4 | TAPE temporary location. |
| | $E5 | TAPE BlockLo |
| | $E6 | TAPE BlockHi |
| | $E7 | TOE_MemptrLo low byte general purpose pointer |
| | $E8 | TOE_MemptrHi high byte general purpose pointer. |

| | | |
|---|---|---|
| | $E9 | unused |
| | $EA | unused |
| | $EB | unused |
| | $EC | unused |
| | $ED | unused |
| | $EE | unused |
| Decss | $EF | Number to decimal string start |
| Decssp1 | $F0 | Number to decimal string start |
| | $FF | Decimal string end |

# Chapter 2. OS Memory Map.

## TowerBASIC Non-Zero Page Memory

Operating systems memory is necessary, or the OS would not work. Some benefit to the user is afforded by the clear documentation of this memory below.

| | | |
|---|---|---|
| $200 | ccflag | Control-C flag. 0=Enabled, 1=Disabled. |
| $201 | ccbyte | Control-C character. One may define whatever one chooses, but with great power comes great RESETs! |
| $202 | ccnull | Timeout time for the Control-C character. |
| $203-$204 | VEC_CC | Control-C check vector. |
| $205-$206 | VEC_IN | TowerBASIC Input vector. |
| $207-$208 | VEC_OUT | TowerBASIC output vector. |
| $209-$20A | VEC_LD | TowerBASIC LOAD vector. |
| $20B-$20C | VEC_SV | TowerBASIC SAVE vector. |
| $20D-£20E | VEC_VERIFY | TowerBASIC VERIFY vector. |

THE Next two vectors (Greyed out) are no longer in use.

| | | |
|---|---|---|
| $F9F3-$F9F4 | IRQ_vec | Interrupt ReQuest vector. (Currently patched out, see Chapter 12, The IRQ Sub-System.) |
| $20F-$210 | NMI_vec | Non Maskable Interrupt vector. |

## ToE Monitor Top Level Component Memory

| | | |
|---|---|---|
| $5E0 | os_outsel | Output stream selection bitfield. See table at end of chapter. |
| $5E1 | os_infilt | Input stream filter selection bitfield. See table at end of chapter. |
| $5E2 | os_insel | This variable contains an input stream selection bit. The bitfield allows you to select which source you want to use, with invalid selections causing a return to ACIA1. |

## ToE ACIA Configuration Variables

| | | |
|---|---|---|
| $5E3 | ACIA1_cfg_cmd | The load values for configuring ACIA1's command register upon initialisation. |
| $5E4 | ACIA1_cfg_ctrl | The load values for configuring the ACIA1's control register upon initialisation. |
| $5E5 | ACIA2_cfg_cmd | The load values for configuring ACIA2's command register upon initialisation. |
| $5E6 | ACIA2_cfg_ctrl | The load values for configuring ACIA2's command register upon initialisation. |

## ToE I2C Engine Variables

| | | |
|---|---|---|
| $5D0 | I2C Status | Stores the status of the I2C engine including the ACK/NAK bit. |
| $5D1 | I2C_Byte | This is the data register for the I2C engine. |

## Tower Peripheral Bus Memory Locations

| | | |
|---|---|---|
| $5F2 | TPB_curr_dev | TPB Currently selected device ID. |
| $5F3 | TPB_dev_type | TPB device type. |
| $5F4 | TPB_last_read | Last read byte from the TPB bus. |
| $5F5 | TPB_BUS_status | Status word for the TPB bus. Subject to change. |
| $5F6 | TPB_BUS_tries | Bus device counter, this ensures fewer hangs. |
| $5F7 | TPB_BUS_lim | Bus countdown timer limit. (Reload value). |
| $5F8 | TPB_BUS_lenlo | Low byte of the length of the block in or out. |
| $5F9 | TPB_BUS_lenhi | High byte of the length of the block in or out. |
| $5FA | TPB_BUS_stlo | Low byte of the start of the block in or out. |
| $5FB | TPB_BUS_sthi | High byte of the start of the block in or out. |
| $5FC | TPB_BUS_blk_type | Type of block transfer. See Table below. |
| $600 | TPB_Dev_table | Device descriptor table. |
| $610-$6FF | TPB_BUS_IO_buff | Buffer for IO operations on the TPB bus. |
| $700-$7FF | TPB_BUFFER | Buffer for TPB block operations. |

## TowerTAPE Filing System Memory.

| | | |
|---|---|---|
| $900-$901 | V_TAPE_BlockSize | Size of block to transfer to or from tape. |
| $902 | TAPE_Temp2 | Temporary memory location for TFS internals. |
| $903 | TAPE_Temp3 | Temporary memory location for TFS internals. |
| $904 | TAPE_Temp4 | Temporary memory location for TFS internals. |
| $905 | TAPE_LineUptime | Number of passes the tape system superloop has made since the tape line rose. |
| $906 | TAPE_Demod_Status | Demodulated bit status. |
| $907 | TAPE_Demod_Last | Previous demodulated bit status. |
| $908 | TAPE_StartDet | Start bit detection status. |
| $909 | TAPE_RX_Status | Receive engine status bitfield. |
| $90A | TAPE_BitsToDecode | Down counter of remaining bits to decode. |
| $90B | TAPE_ByteReceived | Last byte received by the TFS. |
| $90C | TAPE_Sample_Position | Countdown timer for bit engine sample synchronization. |

| | | |
|---|---|---|
| $90D | TAPE_BlockIn_Status | Status register for the F_TAPE_BlockIn function. |
| $90E-929 | TAPE_Header_Buffer | This is where the tape header information is stored for use when **SAVE**ing and **LOAD**ing. |
| $92A | TAPE_CS_AccLo | Tape checksum accumulator low byte. |
| $92B | TAPE_CS_AccHi | Tape checksum accumulator high byte. |
| $92C | V_TAPE_Phasetime | Tape phase time variable. |

The following variables are allocated for future upgrades to the tape system and ignored for now.

| | | |
|---|---|---|
| $92D | V_TAPE_Sample_Offset | Sample offset variable. This determines how far into a bit the sample is taken. |
| $92E | V_TAPE_Bitlength | How many bits in a word stored on tape. |
| $92F | V_TAPE_bitcycles | Number of cycles of the super-loop to a bit. |
| $930 | V_TAPE_Verify_Status | Stores the verify status bits used by the TAPE_VERIFY_vec ($FF7E) |
| $931-$942 | V_TAPE_Fname_Buffer | Working file name buffer. Stores the null terminated file name specified in **LOAD**, **SAVE** and **VERIFY** commands. |
| $943 | V_TAPE_LOADSAVE_Type | Temporary store of what file type is being worked on by the TAPE file system. |
| $944-$945 | V_TAPE_Address_Buff | Temporary store of the starting address being worked on by the TAPE file system. |
| $946-$947 | V_TAPE_Size_Buff | Temporary store of how big the file being worked on is. |
| $948 | V_TAPE_Config | TowerTAPE Filing system configuration word. |

## AY Soundcard V2 Memory Locations

| | | |
|---|---|---|
| $A00 | AY_Reg | Register to write to |
| $A01-$A02 | AY_Data | Contents to be transferred between the system and the AY-3-8912A when calling AY_Userwrite_vec ($FFD5), AY_Userwrite_16_vec ($FFCF), AY_Userread_vec ($FFD8) or AY_Userread_16_vec ($FFD2). |
| $A03 | AY_Mask | Contains the shadow copy of the enable bits for the AY-3-8912A sound channels. |
| $A04 | AY_Channel | Contains a shadow copy of the channel specified in the BASIC command **SOUND**. |

| | | |
|---|---|---|
| $A05-$A06 | AY_Period | Contains a shadow copy of the period specified in the BASIC command **SOUND**. |
| $A07 | AY_Volume | Contains a shadow copy of the volume specified in the BASIC command **SOUND**. |
| $A08-$A09 | AY_Envelope_Period | Contains a shadow copy of the envelope period specified in the BASIC command **ENVELOPE**. |
| $A0C | AY_Envelope_Mode | Contains a shadow copy of the envelope mode specified in the TowerBASIC command **ENVELOPE**. |

## IRQ Handler Subsystem Locations

| | | |
|---|---|---|
| $A20-$A2F | IRQH_CallList | Table of 8 addresses for the IRQ Handlers |
| $A30-$A31 | IRQH_CallReg | Address being worked on by set or clear calls. |
| $A32 | IRQH_ClaimsList | bitfield showing which IRQ's claimed an interrupt. LSb is IRQ0, MSb is IRQ7 |
| $A33 | IRQH_MaskByte | Selection switch for interrupts. 1 means on and the order is LSb for IRQ0 through MSb for IRQ7. |
| $A34 | IRQH_WorkingMask | Used internally when enumerating IRQs. |
| $A35 | IRQH_CurrentEntry | Convenience variable informing IRQs etc which IRQ is currently being handled. |
| $A36-$A46 | IRQH_Command_Table | 16-byte table consisting of a parameter followed by the command code. Ordered by ascending IRQ number. |

## Countdown Timer IRQ Locations

| | | |
|---|---|---|
| $A46 | CTR_V | Counter. This is the countdown timer's present value, decremented each GPIO card Timer 1 IRQ. |
| $A48 | CTR_LOAD_VAL_V | Counter Load Value. Reload value for the GPIO card Timer 1 IRQ. Decrements once per PHI2 clock. Refer to your CPU jumper setting. |

## System Vector Locations

| | | |
|---|---|---|
| $FF60 | TOE_PrintStr_vec | Prints a null terminated string to the currently selected outputs. |
| $FF63 | TAPE_Leader_vec | Generates a tape leader signal. |
| $FF66 | TAPE_BlockOut_vec | Transmits a block of bytes. |

| | | |
|---|---|---|
| $FF69 | TAPE_ByteOut_vec | Transmits a byte. |
| $FF6C | TAPE_BlockIn_vec | Reads a block of bytes from tape. |
| $FF6F | TAPE_ByteIn_vec | Reads a byte from tape. |
| $FF72 | TAPE_init_vec | Initialises the tape system. |
| $FF75 | TAPE_CAT_vec | Continually scans the tape, outputting the filename and type of any found files. |
| $FF78 | TAPE_SAVE_BASIC_vec | **SAVE**s a program to tape. |
| $FF7B | TAPE_LOAD_BASIC_vec | **LOAD**s a program from tape. |
| $FF7E | TAPE_F_TAPE_VERIFY_BASIC | **VERIFY**s a program in memory against what is stored on tape. Can be accessed from TowerBASIC using the **VERIFY** command. |
| $FF90 | ANSI_init_vec | Initialises the ANSI card. |
| $FF93 | ANSI_write_vec | Writes whatever is in the accumulator to the ANSI card. |
| $FF96 | TPB_init_vec | Initialises the Tower Peripheral Bus card. |
| $FF99 | TPB_LPT_write_vec | Writes the contents of A to the Centronics port. |
| $FF9C | TPB_tx_byte_vec | Writes a byte to the tower peripheral bus. |
| $FF9F | TPB_block_vec | Writes a block of bytes to the tower peripheral bus. |
| $FFA2 | TPB_ATN_handler_vec | Processes ATN signals generated by TPB peripherals. |
| $FFA5 | TPB_rx_byte_vec | Reads a byte from the TPB bus. |
| $FFA8 | TPB_rx_block_vec | Reads a block from the TPB bus. |
| $FFAB | TPB_Dev_Presence_vec | Checks for the presence of a given device on the bus. |
| $FFAE | TPB_Req_Dev_Type_vec | Requests the device report its device type. |
| $FFB1 | TPB_dev_select_vec | Selects a device on the TPB. A device must not respond unless selected. |
| $FFB4 | TPB_Ctrl_Block_Wr_vec | Writes to a devices control block. |
| $FFB7 | TPB_Ctrl_Block_Rd_vec | Reads a devices control block. |
| $FFCF | AY_Userwrite_16_vec | Writes to the specified AY register and its consecutive register. |
| $FFD2 | AY_Userread_16_vec | Reads from the specified AY register and its consecutive register. |
| $FFD5 | AY_Userwrite_vec | Writes to the specified AY register. |
| $FFD8 | AY_Userread_vec | Reads from the specified AY register. |
| $FFDB | IRQH_Handler_Init_vec | Initialises the IRQ Handler sub-system. |
| $FFDE | IRQH_SetIRQ_vec | Atomically sets an IRQ address in the handler table. |
| $FFE1 | IRQH_ClrIRQ_vec | Atomically clears an IRQ to the null IRQ handler in the specified handler table. |
| $FFE4 | IRQH_SystemReport_vec | Returns the IRQ Handler subsystem version and base address of the handler data structure. |
| $FFE7 | INIT_COUNTDOWN_IRQ | Initialises the countdown timer IRQ. |

## CPU Vectors (CPU hard-wired)

| | | |
|---|---|---|
| $FFFA | NMI_vec | Non-Maskable-Interrupt vector. |
| $FFFC | RES_vec | Reset vector. This is the reset vector. The CPU takes the address here as it's start point. |
| $FFFE | IRQ_vec | Interrupt request vector. IRQ's jump from here. |

## System Soft-Switches

System soft switches are bits stored within system variables that control characteristics of the system such as output streams and input filtering. This section is likely to grow with revision changes.

| *os_outsel ($5E0)* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Function | Reserved for future uses. | | | TAPE | ACIA2 | TPB LPT | ANSI | ACIA1 |

| *os_infilt ($5E1)* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Function | Reserved for future uses. | | | | | | ACIA2 LF Filter | ACIA1 LF Filter. |

| *os_insel ($5E2)* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Function | Reserved for future uses. | | | TAPE | ACIA2 | Reserved | | ACIA1 |

# Chapter 3. The ANSI Card.

## ANSI Card Description

ANSI stands for American National Standards Institute, and in this case, we're dealing with ANSI terminal output.

Each character is put into the card's internal memory by means of addressing the W65C22S-TPG14 versatile interface adapter on the ANSI card. This card must be set to base address C000-C00F by setting the selector wheel to 0 if one wants to access it via the TowerOS, a second ANSI card could be installed but you must drive it yourself as no OS support is given. To read or set the cards address, just use the number on the wheel as the most significant digit of a two-digit hexadecimal number and add the nIOSEL base address to it.

Refer to the Western Design Center datasheet for operation of the VIA, something which cannot be understated in its importance on the ToE as it is extensively utilised on several add-on cards.

The Avail line is connected to PA6 and indicates the cards availability for use. PA7 is the acknowledge line and flips upon receipt of any byte placed upon port B.

Therefore, the relevant addresses are as per the below table.

| Address | ANSI Function |
|---|---|
| 0 | Output byte (Output Register B) |
| 1 | PA7 = Ack, PA6 = Avail, PA5-0 not used. (Output Register A) |
| 2 | Data Direction Register B. Should be set to $FF |
| 3 | Data Direction Register A. Should be set to $40 |

To write directly to the ANSI processor, one checks the Ack bit and current Avail bit are matching, only changing the content of the data byte when they do. Then, one flips the Avail line. When the ANSI processor has accepted it, both the Ack and Avail lines will match. One can go about one's merry way but may not change the contents without first checking for agreement.

The nIRQ line of the VIA is *not* connected to the system bus. If one wishes to use the VIA internal hardware to generate an interrupt, one must use a Schottky diode to wire-or to the appropriate interrupt line. There is nothing stopping one from using the internal timers however, you would just need to poll the registers in software.

Access to the ANSI card through the TowerOS is the recommended method and the following functions below are how this is achieved.

ANSI_init_vec Call Address: $FF90

Calling this address initialises the ANSI card registers. Normally you won't need this as TowerOS does this on start-up.

ANSI_write_vec Call Address: $FF93

The contents of the Accumulator is written to the ANSI card. Note that if the ANSI card is still busy, this will function will block until it has completed. There is no timeout, error condition, or filters.

## System Soft switch Control

The system soft switch for this card is in os_outsel ($5E0) and is bit 1. Setting this bit causes the system to send output to the ANSI card and is on by default unless you use the ACIA build of the ToE ROM.

## Controlling the Display

To control the display, one sends control codes which cause the video processor to update the display contents. There is a full set of ASCII characters including extended PC DOS characters and some limited graphics.

Other effects include doubling the width, height of the characters, and whether it is bold.

To control the character width, height and whether it is bold, one sends a character control attribute $18 (24) followed by a byte containing the following bitfield:

| *Character Control Attribute $18 (24) and bitfield of following byte* | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Graphics | Spare | Spare | Spare | Spare | Double Height | Bold | 80 Columns |

## Positioning the Cursor

Issuing control code $0E (14) (set column) or $0F (15) (set row) followed by the desired position achieves this goal. TowerBASIC Also has the **`LOCATE`** command which sets both x and y position.

## Redefining the Cursor

Control code $02 (2) followed by the ASCII character desired defines the cursor. If the cursor is set to 0 then it is turned off. Leaving the cursor over a graphical area causes the undesirable effect of making graphical area to flash.

## Accessing the PC DOS Characters ( 0 (0) – $1F (31) )

To use these extra characters, send $1A (26) first, this causes the next character to be displayed as the specified PC DOS character.

## Simple Graphics

The pixel resolution is 160 across by 100, the origin of which is top left at 0,0 up to 159,99 in the bottom right.

One may issue a SetPixel command $5 (5) or ClearPixel $6 (6) command followed by x then y. TowerBASIC also has the **`PLOT`** command to set or clear a pixel.

Please note that there is currently no way to read back what you have on the screen so you will need to keep track of the salient details.

## Writing a Pixel Pattern Directly

Internally, each character cell can be loaded with a 2 by 4 pattern by first sending $80 (128), then the bit pattern is defined by the bit position of the character cell.

## TowerBASIC commands for graphics

There are three commands currently implemented that assist in the use of the ANSI card.

**`CLS`**

This clears the screen to a useable state.

**`LOCATE x,y`**

Sets the print cursor to position x, y.

**`PLOT m,x,y`**

If m is 1 then this sets a pixel at x,y pixel position, but if m is 0 then it clears that pixel.

## *Video Control Codes*

| | | |
|---|---|---|
| $01 (01) | Cursor home | *(Standard ASCII)* |
| $02 (02) | Define cursor character (2nd byte is the cursor character, or 0 to turn off) | |
| $03 (03) | Cursor blinking | |
| $04 (04) | Cursor solid | |
| $05 (05) | Set graphics pixel (next two bytes = x,y) | |
| $06 (06) | Reset graphics pixel (next two bytes = x,y) | |
| $08 (08) | Backspace | *(Standard ASCII)* |
| $09 (09) | Tab | *(Standard ASCII)* |
| $0A (10) | Linefeed | *(Standard ASCII)* |
| $0C (12) | Clear screen | *(Standard ASCII)* |
| $0D (13) | Carriage return | *(Standard ASCII)* |
| $0E (14) | Set column 0 to 79 (2nd byte is the column number) or 0 to 39 for a 40 char line | |
| $0F (15) | Set row 0 to 24 (2nd byte is the row number) | |
| $10 (16) | Delete start of line | |
| $11 (17) | Delete to end of line | |
| $12 (18) | Delete to start of screen | |
| $13 (19) | Delete to end of screen | |
| $14 (20) | Scroll up | |
| $15 (21) | Scroll down | |
| $16 (22) | Scroll left | |
| $17 (23) | Scroll right | |
| $18 (24) | Set font attribute for the current line | |
| $1A (26) | Treat next byte as a character (to allow PC DOS char codes 1 to 31 to be displayed on screen) | |
| $1B (27) | ESC - reserved for ANSI sequences | |
| $1C (28) | Cursor right | |
| $1D (29) | Cursor Left | |
| $1E (30) | Cursor up | |
| $1F (31) | Cursor down | |
| $20 (32) to… $7E (126) | Standard ASCII codes | |
| $7F (127) | Delete | |
| $80 (128) to… $FF (255) | PC (DOS) extended characters | |

# Chapter 4. The Single ACIA Card.

## Single ACIA Card Description

Serial communications are provided to permit headless usage, for serial terminals, modems and many other devices to be connected to the Tower of Eightness. Most notably is keyboard input, the PS2 to Serial interface being the easiest way to control the ToE. Without some way to communicate with the ToE it would be useless and so this is one of the essential interfaces. The default port setting is 9600 baud, 1 start bit, 1 stop bit, no parity and RTS/CTS hardware handshaking.

The 65C51-4P or equivalent Asynchronous Communications Interface Adapter which ACIA stands for, bridges the gap between the system bus and RS232 serial. This card provides a 9 pin RS232 port at the correct signalling levels.

Mark is -Ve and Space is +Ve. Serial data always starts with a start bit, 5 to 8 data bits depending on the setting and at least one stop bits at a regular rate known as the bitrate. The bit rate is not the baud rate as can be realised by considering that the start bit and stop bits take up time also, therefore the baud rate is less than the bitrate by necessity.

Provided below is the pinout of the serial IO DB9 connector which is wired as Data Terminal Equipment. The only handshaking lines provided however are Request-To-Send and Clear-To-Send.

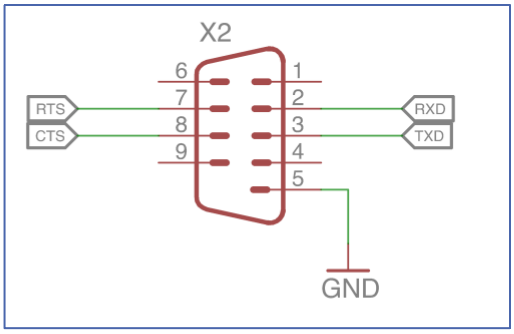



*Figure 1*

## Setting the IO Address

To set the IO address one moves jumpers on JP3 to either closed or open positions. Open is a 1 and closed is a 0. There are six of them and they form the top six bits of the IO address offset from nIOSEL which is at $C000. This card occupies four locations in IO address space. The default base address and the one the Tower OS will use is at $C010 meaning that all the jumpers should be set to CCOCCC from back edge to front.

The least significant bit is nearest the bus connector and the most significant furthest away. Whatever binary value is jumpered, just multiply it by four and add the nIOSEL base address.

It is possible to add several ACIA cards, and even mix with dual ACIA cards. but they must not share IO addresses with anything else and at least *one* must be at $C010 as it receives the keyboard input.

## System Soft switch Control

The system soft switch for output to this card is in os_outsel ($5E0) and is bit 0. Setting this bit causes the system to send output to the ACIA card and is off by default unless you use the ACIA build of the ToE ROM.

There is also a system soft switch for input filtering, which is os_infilt ($5E1). Bit 0 when set to 1 (default is set) strips out linefeed characters ($A).

## ToE ACIA Configuration Variables

| | | |
|---|---|---|
| $5E3 | ACIA1_cfg_cmd | The load values for configuring ACIA1's command register upon initialisation. |
| $5E4 | ACIA1_cfg_ctrl | The load values for configuring the ACIA1's control register upon initialisation. |

**The following variables are only applicable if a second ACIA is present at $C014.**

| | | |
|---|---|---|
| $5E5 | ACIA2_cfg_cmd | The load values for configuring ACIA2's command register upon initialisation. |
| $5E6 | ACIA2_cfg_ctrl | The load values for configuring ACIA2's command register upon initialisation. |

## ACIA Vectors

TowerOS provides the following vectors for direct access to the ACIAs but the ACIA2 vectors are only relevant if you have a second ACIA at $C014.

| | |
|---|---|
| ACIA_INI_SYS_vec ($FF42) | Initialises the ACIA system to default values. 8N9600 for both ports. |
| ACIA1_init_vec ($FF45) | Initialises ACIA1 to the values in ACIA1_cfg_ctrl ($5E4) and ACIA1_cfg_cmd ($5E3) system variables. Changing these variables and re-initialising the ACIA can changes the settings. |
| ACIA2_init_vec ($FF48) | Initialises ACIA2 to the values in ACIA2_cfg_ctrl ($5E6) and ACIA2_cfg_cmd ($5E5) system variables. Changing these variables and re-initialising the ACIA can changes the settings. |
| ACIA1out_vec ($FF4B) | Waits until ACIA1's transmit buffer is empty then puts the contents of the accumulator into the transmit buffer. |
| ACIA2out_vec ($FF4E) | Waits until ACIA2's transmit buffer is empty then puts the contents of the accumulator into the transmit buffer. |

# Chapter 5. The Dual ACIA Card.

## Dual ACIA Card Description

Serial communications are provided to permit headless usage, for serial terminals, modems, and many other devices to be connected to the Tower of Eightness. Most notably is keyboard input, the PS2 to Serial interface being the easiest way to control the ToE. Without some way to communicate with the ToE it would be useless and so this is one of the essential interfaces. The default port setting is 9600 baud, 1 start bit, 1 stop bit, no parity and RTS/CTS hardware handshaking.

The 65C51-4P or equivalent Asynchronous Communications Interface Adapter which ACIA stands for, bridges the gap between the system bus and RS232 serial. This card provides a 9 pin RS232 port at the correct signalling levels.

Mark is -Ve and Space is +Ve. Serial data always starts with a start bit, 5 to 8 data bits depending on the setting and at least one stop bits at a regular rate known as the bitrate. The bit rate is not the baud rate as can be realised by considering that the start bit and stop bits take up time also, therefore the baud rate is less than the bitrate by necessity.

Provided below is the pinout of the serial IO DB9 connectors which are wired as Data Terminal Equipment. The only handshaking lines provided however are Request-To-Send and Clear-To-Send.

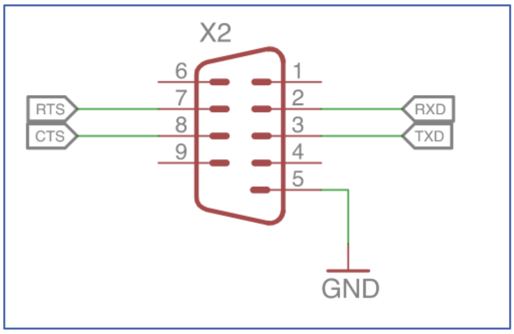



*Figure 1*

## Setting the IO Address

To set the IO address one sets the DIP switches to the appropriate binary address. There are six of them and they form the top six bits of the IO address offset from nIOSEL which is at $C000. This card occupies two sets of four locations in IO address space. The default base address and the one the Tower OS will use for primary input is at $C010 meaning that the appropriate DIP switch setting is 00100. The second ACIA port should be set to $C014 with the DIP switches set to 00101.

The least significant bit is nearest the bus connector and the most significant furthest away. Whatever binary value is jumpered, just multiply it by four and add the nIOSEL base address.

It is possible to add several ACIA cards, but they must not share IO addresses with anything else and at least *one* must be at $C010 as it receives keystrokes upon start up.

## System Soft switch Control

The system soft switches for output to this card are in os_outsel ($5E0). Bit 0 controls the primary ACIA and bit 3 controls the secondary ACIA. Setting these bits causes the system to send output to the selected ACIA and ACIA 1 is on off default unless you use the ACIA build of the ToE ROM.

There are also system soft switches for input filtering, which are in os_infilt ($5E1). Bit 0 when set to 1 (default is set) strips out linefeed characters ($A) on ACIA 1 and bit 1 (on by default) filters ACIA 2 in the same manner.

## ToE ACIA Configuration Variables

| | | |
|---|---|---|
| $5E3 | ACIA1_cfg_cmd | The load values for configuring ACIA1's command register upon initialisation. |
| $5E4 | ACIA1_cfg_ctrl | The load values for configuring the ACIA1's control register upon initialisation. |
| $5E5 | ACIA2_cfg_cmd | The load values for configuring ACIA2's command register upon initialisation. |
| $5E6 | ACIA2_cfg_ctrl | The load values for configuring ACIA2's command register upon initialisation. |

## ACIA Vectors

TowerOS provides the following vectors for direct access to the ACIAs.

| | |
|---|---|
| ACIA_INI_SYS_vec ($FF42) | Initialises the ACIA system to default values. 8N9600 for both ports. |
| ACIA1_init_vec ($FF45) | Initialises ACIA1 to the values in ACIA1_cfg_ctrl ($5E4) and ACIA1_cfg_cmd ($5E3) system variables. Changing these variables and re-initialising the ACIA can changes the settings. |
| ACIA2_init_vec ($FF48) | Initialises ACIA2 to the values in ACIA2_cfg_ctrl ($5E6) and ACIA2_cfg_cmd ($5E5) system variables. Changing these variables and re-initialising the ACIA can changes the settings. |
| ACIA1out_vec ($FF4B) | Waits until ACIA1's transmit buffer is empty then puts the contents of the accumulator into the transmit buffer. |
| ACIA2out_vec ($FF4E) | Waits until ACIA2's transmit buffer is empty then puts the contents of the accumulator into the transmit buffer. |
| ACIA1in_vec ($FF51) | Checks ACIA1's input buffer for received data and if present loads it into the accumulator. If |

| | |
|---|---|
| | no data is present, then the accumulator is set to 0. The carry flag is set if successful, otherwise it is cleared. |
| ACIA2in_vec ($FF54) | Checks ACIA2's input buffer for received data and if present loads it into the accumulator. If no data is present, then the accumulator is set to 0. The carry flag is set if successful, otherwise it is cleared. |

# Chapter 6. The GPIO Card.

## GPIO Card Description

This card is basically both ports, complete with CA1, CA2, BA1 and BA2 broken out into two identical sockets. The IRQ line is also connected to assist the programmer with its use.

For those wishing to attach an external peripheral, be it home-made or otherwise, the GPIO card implements not one, but two BBC Micro equivalent user ports. Port B is notably however, used for the cassette and joystick interface and if used for other things may clash with other things. This is true only of this card if it is mapped into a base address of $C040.

## General Configuration

To set its address, one set the jumpers in accordance with the upper nybble of the lower byte of its address and then adds on the nIOSEL offset which is $C000.

Placing the card on a flat surface with the bus connector to the left, moving a jumper to the left represents a binary 0 and to the right is binary 1. CID1 is the least significant bit and CID4 is the most significant bit so the default from CID1 to 4 is left, left, right, left.

## Usage

There are only calls to use port B with the cassette interface and port A with I²C and SPI. To use these, please read the chapter entitled Cassette and Joystick Interface for the cassette, or the chapters on I²C and SPI. The pins for I²C and SPI are not configured until their respective engines are initialised. To take full advantage of this card one needs to write directly to the registers of its VIA chip and so one is directed to the WDC65C22S6-TPG14 datasheet. This is especially important with regards to the VIAs electrical specification as exceeding those may lead to damage to the card and even in some cases the ToE itself.

The some of the pins of the VIA have special features specific to that pin and internal to the VIAs are timers, counters, handshake lines, pulse generators etc.

## Pinout

**Port A. Base address offset 1, Data Direction Register Offset 3.**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| 0V | 0V | 0V | 0V | 0V | 0V | 0V | 0V | +5V | +5V |
| 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 | CA2 | CA1 |

**Port B. Base address offset 0, Data Direction Register Offset 2.**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| 0V | 0V | 0V | 0V | 0V | 0V | 0V | 0V | +5V | +5V |
| 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 | CB2 | CB1 |

Be aware that neither of these ports is hot pluggable or protected by any kind of static, or out of specification protection except for the VIAs internal diodes. Also, there is no fuse or other kind of current limiting on the 5V lines. It is up to the user to ensure that the bus is not abused.

# Chapter 7. The AY-3-8912A Sound Card V2.

> *"Music speaks what cannot be expressed,*
> *sooths the mind and gives it rest,*
> *heals the heart and makes it whole,*
> *and flows from heaven to the soul."*

## AY Soundcard Description

From the above prose, you might gather that this is a *sound card* and so it is that you may, with minimal skill produce at least some beeps and squarks. With great skill the chip this sound card is based on has produced some very joyous tunes rejoiced by many a kid of the 80's.

The Tower of Eightness has a maximum clock speed primarily limited by its peripheral set. As the processor has a design limit of 14MHz and the backplane is good for perhaps 10MHz, only cards that can handle the system clock frequency and do not unduly load the bus may operate correctly as part of the system. One such device to limit the system speed is the AY-3-8912A sound chip which here is clocked locally at 1.842MHz. To that end, this card isolates the sound chip from the main system bus to permit maximum system operating speed. This isolation is achieved by the use of an on card W65C22S VIA chip.

This VIA is mapped into $C0E0 to $C0EF by convention, thereby allowing programs to directly drive it for maximum performance. This is necessary as sound is popular for games amongst other things.

The nIRQ line is connected so that one may use interrupts.

Port B (Offset 0) is connected to the data lines of the AY-3-8912A, and Port A (Offset 1) is used for control lines. PA0 is connected to BC1 whilst PA1 is connected to BDIR. BC2 is tied high therefore creating a simplified bus for the AY chip.

PA0 and PA1 should be set as outputs and their logic should be driven as listed below.

It is important to note that there is currently no logic protection to prevent bus contention between the AY and the VIA chip and the onus is on the programmer to maintain harmony and protect the logic from stressful conditions. More advanced logic will be brought forward at a later date but for now this is all that is required. This has been done with the Oric-1 and Oric Atmos computers and many of those have survived more than 30 years so this is a proven adequate solution.

When BDIR is low, the AY bus is readied for output of its internal registers, and when high it is ready to receive from the bus.

BC1 should remain low until Port A is configured as an input with BDIR low or an output with appropriate data on it and BDIR high. BC1 should be strobed to make the transfer.

The following table should clarify this: -

| *Table of Bus States* | | |
|---|---|---|
| BC1 | BDIR | State |
| 0 | 0 | Inactive. |
| 1 | 0 | Read from AY. The AY selected register is on the data lines. |
| 0 | 1 | Write to AY. The data lines are transferred to the AY |
| 1 | 1 | Latch Address. Write register address to AY from data lines. |

For the full AY-3-8912A hardware specification, refer to the Microchip datasheet.

## Configuration

The only configuration that can be done to this card is to set its base address offset by means of the provided jumpers. The four jumpers are arranged such that the least significant bit is nearest the VIA and the most significant nearest the edge of the card. These jumpers form a binary address that is the most significant four bits of the offset from nIOSEL. Leave the base address at $C0E0 unless this is an additional sound card since software is being targeted at the above agreed address and firmware is being written to use it too.

## Usage from TowerBASIC

There are two commands to make sounds and music on the AY, **SOUND** and **ENVELOPE**.

**SOUND** takes the form below.

**SOUND** channel, period, volume

where channel is 0 to 6. 0 through 2 are the sound channels A, B and C whilst 3 through 5 is using the same output channels but for noise.

For tones, period is any value from 0 to 4095 with the period being calculated as being 1.842MHz/16/period.

Volume is fixed between 0-15 or if 16 used, it is defined by the **ENVELOPE** period and mode.

**ENVELOPE** defines the period and modulation mode of any waveform set to volume 16. The envelope period is in the range of 0-65535 and is calculated as 1.842MHz/256/period. It is unfortunate that the AY contains only one envelope generator and that this affects all sound channels so set equally. If you need to control a sound channels envelope with a differing mode or something more exotic, wouldn't go far wrong with an interrupt in assembly language, though this negates the use of TowerBASIC whilst the interrupt is in use and consumes further resources.

**ENVELOPE** period, mode

Here is an example which plays a bell sound at about C5 (523.25Hz).

```
10 SOUND 1, 220,16
20 ENVELOPE 24000,0
```

## The Mode Parameter.

The mode parameter is a bitfield containing four bits that control the envelope generator of the AY.

| *Mode parameter* | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| | | | | Continue | Attack | Alternate | Hold |

Hold (bit 0) when set, limits the envelope generator to one cycle when set, the value at the end of the cycle being held at the end of the cycle.

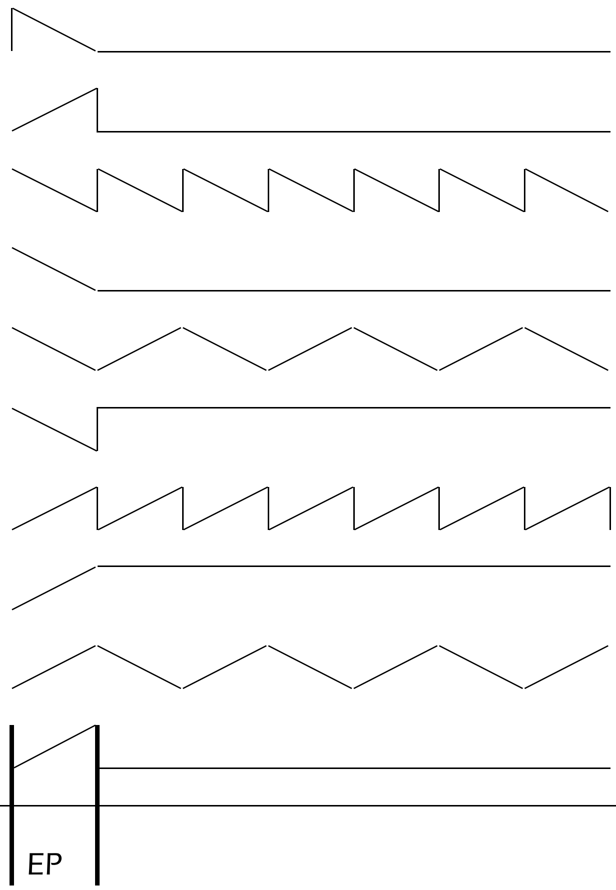
Alternate (bit 1) when set, causes the envelope generator to reverse the direction of its cycle when it reaches its end. It thus produces an up-down effect.

Attack (bit 2) when set, causes the envelope generator to count up, and when clear causes it to count down, producing a decay.

Continue (bit 3) when set, causes the cycle pattern to be defined by the hold bit, but when clear causes the counter to reset to 0 after one cycle.

## Envelope Table.

| *Envelope Shape/Cycle Operation* | | | | |
|---|---|---|---|---|
| Mode bits (AY reg 13) | | | | |
| B3 | B2 | B1 | B0 | |
| Continue | Attack | Alternate | hold | Graphic Representation of Envelope Generator Output. |
| 0 | 0 | X | X | |
| 0 | 1 | X | X | |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 1 | |




**EP** is the envelope period (duration of one cycle).

The envelope period is limited to 16 different logarithmic values and as such the envelope has significant jumps at the higher volume levels. Luckily, human hearing is also logarithmic.

## Register Writes to the AY

To write directly to an AY register one simply pokes the appropriate register address to AY_Reg ($A00) and data to AY_Data ($A01-$A02) and calls the appropriate function call. It is of course possible to drive the AY by directly controlling the W65C22 on the AY card this in fact necessary if you change its address or add a second card. It should be noted that it is far quicker to make these calls than it is to POKE the registers directly, and quicker (and easier) still for many use cases to use the provided TowerBASIC commands. Below is a list of currently implemented function call vectors.

## List of AY function call vectors

| | |
|---|---|
| AY_Userwrite_16_vec ($FFCF) | Takes a 16-bit word and puts it in the registers specified by AY_Reg ($A00) and the consecutive register. |
| AY_Userread_16_vec ($FFD2) | Fetches a 16-bit word from the registers specified by AY_Reg ($A00) and the consecutive register and places it in AY_Data ($A01-$A02). |
| AY_Userwrite_vec ($FFD5) | Takes a byte at AY_Data ($A01) and puts it in the register specified by AY_Reg ($A00). |
| AY_Userread_vec ($FFD8) | Fetches a byte from the register specified by AY_Reg ($A00) and places it in AY_Data ($A01, low byte). The high byte is not overwritten. |

## AY System Memory Locations

| | |
|---|---|
| AY_Reg ($A00) | Pointer to the AY register to be either written to or read from. |
| AY_Data ($A01-$A02) | Used for transferring data to the above pointed register(s). $A01 contains the low byte and $A02 the high one. Used by the AY user read and write functions provided to make the users' usage easier. |
| AY_Mask ($A03) | Holds a shadow copy of which channels are selected. |
| AY_Channel ($A04) | The retrieved copy of the channel specified by the TowerBASIC command **`SOUND`**. |
| AY_Period ($A05-$A06) | The retrieved copy of the period parameter of either of TowerBASICs **`SOUND`** or **`ENVELOPE`** commands. |
| AY_Volume ($A07) | The retrieved copy of the volume specified by the TowerBASIC command **`SOUND`**. |
| AY_Envelope_Period ($A08-$A09) | The retrieved copy of the period specified by the TowerBASIC command **`ENVELOPE`**. |

| | |
|---|---|
| AY_Envelope_Mode ($A0A) | The retrieved copy of the envelope mode specified by the TowerBASIC command **`ENVELOPE`**. |

# Chapter 8. The Tape & Joystick Interface.


Figure 3. Tape and Joystick Interface Version 2.

## Tape and Joystick Interface Description

Tape loading, saving and dual Atari style joystick interface are supplied using this interface. It is designed to be connected to a GPIO port and the TowerOS provides support for it through Port B of the GPIO card mapped to the base address of $C040.

One should keep bits 0 and 7 of DDRB $C042 set when using it as these are used to drive the tape output and select which joystick will be read.

The interface is driven by modulating bit 7 to generate a tape signal for **`SAVE`** operations. Conversely, bit 6 is used to monitor the state of the audio coming in for **`LOAD`** operations.

The tape signal coming in is cleaned up by a biased Schmitt trigger buffer amplifier and fed back to the VIA on pin 6.

The TowerOS provides several system vectors that can be called and has extensive memory locations associated with this interface which will need to be used to make the most of it and are documented later.

To use the joystick interface, one either clears bit 0, selecting joystick port 1 (left) or set it, selecting joystick port 2 (right). The state of the joystick can then be read on Port B bits 1 through 5.

An example fire button read would go something like this: -

```
10 F = NOT(BITTST($C040,5))
```

The observant amongst you will already have noticed the **`NOT`** in that little snippet of TowerBASIC. That is because the joystick ports supply negative logic. A 0 means that switch is pressed.

One should *never* make the joystick bits outputs as that would prevent access to *both* joysticks on the affected bits. It won't however, break anything.

Below is a table of bits associated with the joystick port: -

| *Tape and Joystick Bit Usage* | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Tape Out | Tape In | Fire | Right | Left | Down | Up | JS Select |

## Tape Support in TowerOS

TowerOS supports loading and saving of TowerBASIC from simple **LOAD** and **SAVE** commands, but for the advanced user there are system calls and memory locations associated. It should be noted that when you see **XXXX XXXX** that means that the file is at address **XXXX** hexadecimal and is **XXXX** hexadecimal long in that order. Leading zeros are left in to to keep the output of '**CAT**' neat and regular.

To load an TowerBASIC program simply type '**LOAD** "<optional filename>"¶' and play the cassette. Specifying the file name ensures that *only* that file will be loaded, and the quotes may not be omitted. If the first character of the filename is '**!**' then it will automatically run upon successful load. This feature can be disabled by clearing the TAPE_AutoRUN_En bit (b1) of V_TAPE_Config ($948)

Under normal circumstances, the display will tell you '**Searching**…' whilst it is looking for a header, '**Found BASIC: XXXX XXXX** "<Filename>"' when it finds the specified file and '**Loading**…' whilst it is loading and '**Ready**' when it has completed loading. Not giving a file name causes it to load the *first* file it finds. Should the load encounter an error, you will get the message '**Tape Loading Error.'**.

To save an TowerBASIC program one types '**SAVE** "<filename>"¶' whilst the cassette is recording. You must include a filename of between 1 and 16 characters in length when saving a file. No checks are made for what those characters are except that a null is used to terminate the string in TowerTAPE FS, so avoid this and other unfortunate characters where possible. *Remember, the name you use when saving is the name that will be used to load your program.* If it is too awkward, you may be making a rod for your own back, though clever tricks are also possible this way.

The system will tell you it is '**Saving XXXX XXXX** "<Filename>"' and when done will drop you to the **Ready** prompt. Adding a '**!**' to the start of your chosen filename prompts the system to run it when loaded back. This only applies to TowerBASIC programs. To make a binary execute, use a self-executing BASIC binary loader.

Given the nature of cassette storage, errors are a concern and to ensure the integrity of the saved file, the '**VERIFY** "<optional filename>"¶' command is included. This will operate very much like the **LOAD** "" command, but instead reports on the consistency of the **SAVE**d content with that in RAM, breaking with the **VERIFY Error** the moment it encounters a byte that either can't be read or does not match with the one in memory. This step whilst optional, should not be ignored. Many people have been reduced to tears over lost files.

When one gives a specific file name to **LOAD** or **VERIFY**, the system ignores any files that do not match the name given or of another file type.

If you should receive an error message, then remedial action will be required. The following error messages and their meanings are listed below.

- Header error. Retrying.
  - This means loading error has occurred with the header. TowerTAPE filing system lets you know so you can rewind to the start and try again or escape back to TowerBASIC if you give up.
- Tape loading error.
  - This means loading error has occurred with the file block. TowerTAPE filing system drops you back into the TowerBASIC Ready prompt letting you know of the failure.
- Verify Error.
  - Receiving this indicates that there is something wrong with the program stored on the cassette and that it will need to be re-done. Consider the quality of the equipment and media in use if this becomes too much of a nuisance.
- Filename Too Long
  - Filenames must be no longer than 16 characters and must be present when saving a file.

## Listing the Contents of Cassette.

'**CAT¶**'

Starts the catalogue system, printing each file and its type as it is found. This command needs to be escaped by pressing any key to return control to the system but does not cause a warm start or print upon return. This is so that it may be used from within a program without interrupting flow.

**CAT** displays '**Searching**…' followed by the line '**Found BASIC: XXXX XXXX** "<Filename>"' for each filename found.

## Binary File Handling.

To save binary data one does as per the following example: -

‘**SAVE** “<filename>” **$A000, $100¶**’

This would save $100 bytes of data starting at address $A000. To load in binary data, one must specify a load address as in the below example: -

Loading is accomplished as follows: -

‘**LOAD** “<optional filename>” **$A000¶**‘

And to verify the integrity of the data saved: -

‘**VERIFY** “<optional filename>” **$A000¶**‘

Note that there is no comma separating the address from the filename, it is not needed. One does have to specify the load address at present, as there is currently no other way to indicate that the binary file should be loaded at other than its original address. This feature may be subject to change.

## List of TowerTAPE Filing System Calls

| | |
|---|---|
| $FF63 | TAPE_Leader_vec |
| $FF66 | TAPE_BlockOut_vec |
| $FF69 | TAPE_ByteOut_vec |
| $FF6C | TAPE_BlockIn_vec |
| $FF6F | TAPE_ByteIn |
| $FF72 | TAPE_init_vec |
| $FF75 | TAPE_CAT_vec |
| $FF78 | TAPE_SAVE_BASIC_vec (to be used from TowerBASIC) |
| $FF7B | TAPE_LOAD_BASIC_vec (to be used from TowerBASIC) |

### $FF63 TAPE_Leader_vec

Calling this vector causes the generation of a leader tone.

### $FF66 TAPE_BlockOut_vec

This is the vector call address for F_TAPE_BlockOut. V_TAPE_BlockSize ($900) contains the number of bytes to write and TAPE_BlockLo ($E5) and TAPE_BlockHi($E6) contain the block pointer.

This function returns having modified TAPE_BlockLo ($E5) and TAPE_BlockHi ($E6).

### $FF69 TAPE_ByteOut_vec

Whatever byte is in the accumulator when this function is called is output.

### $FF6C TAPE_BlockIn_vec

Calls to this function require parameters to be loaded into specific memory locations. point to the start of the block to be read. V_TAPE_BlockSize ($900-$901) specifies how many bytes will be read and TAPE_BlockLo ($E5) and TAPE_BlockHi ($E6) contain the write pointer used by this function. No additional information is read.

This function drops through before completion if any character is received from the ACIA card or an overrun error occurs and the state of the engine upon exit is reported in **TAPE_BlockIn_Status** ($90D)

This function returns having modified TAPE_BlockLo ($E5) and TAPE_BlockHi ($E6) which it uses these incrementally point to the byte it is writing to memory at any given moment.

### $FF6F TAPE_ByteIn

Attempts to read a byte from the tape and return it in **TAPE_ByteReceived** ($90B). If a byte is received from the ACIA whilst it is in progress or if an overrun error occurs this function will exit reporting its status in **TAPE_RX_Status** ($909).

### $FF72 TAPE_init_vec

Initialises the TowerTAPE filing system. This is called by TowerOS on bootup and only needs to be called if the TowerTAPE filing system needs to be re-initialised.

### $FF75 TAPE_CAT_vec

Starts the tape system cataloguing routine. Can be called from Assembly language or by using the TowerBASIC keyword CAT.

### $FF78 TAPE_SAVE_BASIC_vec

Causes the TowerBASIC program or specified memory range (as a binary file) to be saved. This should not be called from anywhere except TowerBASIC as it is carefully designed to work as part of TowerBASIC.

### $FF7B TAPE_LOAD_BASIC_vec

Causes the attempted loading of an TowerBASIC program or binary file from tape. This should not be called from anywhere except TowerBASIC as it is carefully designed to work as part of TowerBASIC.

### $FF7E TAPE_VERIFY_BASIC_vec

Causes the attempted verification of an TowerBASIC program or binary stored on tape. This should not be called from anywhere except TowerBASIC as it is carefully designed to work as part of TowerBASIC.

## TowerTAPE Filing System, System Variables

Below is a list of system variables with their associated address and size. These were grabbed direct from a spreadsheet which is also available.

| *System Variable* | *Address* | *Number of Bytes* |
|---|---|---|
| TAPE_BlockLo | $E5 | 1 |
| TAPE_BlockHi | $E6 | 1 |
| V_TAPE_BlockSize | $900 | 2 |
| TAPE_temp2 | $902 | 1 |
| TAPE_temp3 | $903 | 1 |
| TAPE_temp4 | $904 | 1 |
| TAPE_LineUptime | $905 | 1 |
| TAPE_Demod_Status | $906 | 1 |
| TAPE_Demod_Last | $907 | 1 |
| TAPE_StartDet | $908 | 1 |
| TAPE_RX_Status | $909 | 1 |
| TAPE_BitsToDecode | $90A | 1 |
| TAPE_ByteReceived | $90B | 1 |
| TAPE_Sample_Position | $90C | 1 |
| TAPE_BlockIn_Status | $90D | 1 |

*TAPE_Header_Buffer $90E 28*

| | | |
|---|---|---|
| TAPE_HeaderID | $90E | 4 |
| TAPE_FileType | $912 | 1 |
| TAPE_FileSizeLo | $913 | 1 |
| TAPE_FileSizeHi | $914 | 1 |
| TAPE_LoadAddrLo | $915 | 1 |
| TAPE_LoadAddrHi | $916 | 1 |
| TAPE_FileName | $917-927 | 17 |
| TAPE_ChecksumLo | $928 | 1 |
| TAPE_ChecksumHi | $929 | 1 |

## *Remainder of TowerTAPE System Variables*

| | | |
|---|---|---|
| TAPE_CS_AccLo | $92A | 1 |
| TAPE_CS_AccHi | $92B | 1 |
| | | |
| V_TAPE_Phasetime | $92C | 1 |
| V_TAPE_Sample_Offset | $92D | 1 |
| V_TAPE_Bitlength | $92E | 1 |
| V_TAPE_bitcycles | $92F | 1 |
| V_TAPE_Verify_Status | $930 | 1 |
| V_TAPE_Fname_Buffer | $931-$942 | 17 |
| V_TAPE_LOADSAVE_Type | $943 | 1 |
| V_TAPE_Address_Buff | $944-$945 | 2 |
| V_TAPE_Size_Buff | $946-$947 | 2 |
| V_TAPE_Config | $948 | 1 |

V_TAPE_Config has at present two bits for controlling the file system, with only TAPE_AutoRUN_En being implemented. ParityOn is planned for implementation at time of modulation scheme amendment.

| *V_TAPE_Config Bit Usage ($948)* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Function | | | | | | | TAPE_AutoRUN_En | ParityOn |
| Default | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

## TowerTAPE File Modulation Scheme

Each byte is encoded a series of bursts of carrier. A presence of carrier signifies a 1 and an absence a 0. Each burst of carrier is 16 cycles at 4KHz.

Each byte is structured as follows: -

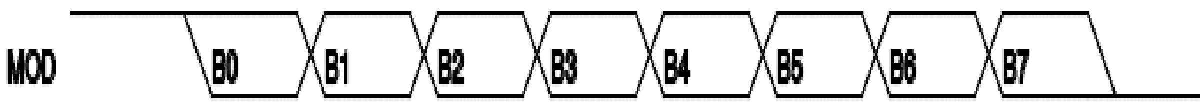



Figure 4 Bit Order for Tape Byte

# Chapter 9. The V2 Memory Board.

## Memory Bank Structure on the ToE

The ToE organises its memory into four groups of two banks of memory of 16 kibibytes each. Each bank group has a selection bit associated with it in a write only register located at address $C0FF. Writes to this register should be handled with great care as to not inadvertently page out something you are using at that time. Since many things you might need are located in the bottom and top banks, these are most likely to be of use to the machine code programmer and unlikely to be of use to anyone programming in TowerBASIC. The banks with the best outcome to the user of TowerBASIC are the middle ones as they can be excluded from use by TowerBASIC at boot time or by careful reconfiguration later.

This register is initialised to 0 at reset placing the lower half of each memory IC into view. For now, the upper four bits should be set to 0 whenever writing to this register as this is reserved for later expansion.

Banks 0 through 2 are RAM and bank 3 is the ToEs ROM containing TowerOS and TowerBASIC.

Although bank 3 consists of 32 kibibytes of space, the address range $C000 to $C0FF is put aside for memory mapped hardware and is inaccessible. This constitutes a loss of just half a kibibyte. In a system with 96 kibibytes of RAM and 32 kibibytes of ROM. A small price to pay for such a generous and fully decoded IO space!

## Jumper Selection

There are two jumpers on this card, the one nearest the bus connector (JP4), is to allow the use of 27 series 16K EPROMs and should be set 1-2 (back) for 27 series, or 2-3 (front) for 28C256 EEPROMs.

The other jumper (JP5) is a write protect. One cannot state enough that this should be left write protected under normal circumstances to avoid crashes and corruption of the ROM. Position 1-2 (back) is write-protected and 2-3 (front) allows writes to occur.

## Firmware Update Guidance

To write to the EEPROM, one must use a bootloader as write times are somewhat in excess of normal access times and the data lines contain write status information whilst a write is being attempted. The CPU would attempt to execute this status information as instructions if code was being executed from the ROM at this time!!

Inadvertent writes will not cause a crash when write protect is on.

# Chapter 10. The CPU Card (V2).

## CPU Card V2 Description

Central to the Tower of Eightness is its processor. The WDC65C02-TPG14 processor is a CMOS microprocessor with an eight-bit wide data bus and a sixteen-bit wide address bus. It executes single byte opcodes at typically two to three processor cycles per instruction and does this at up to 14MHz. The Tower bus being a backplane for various unknown addons has required the CPU to be limited to 4MHz and so the on-board oscillator produces 4, 2 or 1MHz as selected by a set of jumpers.

As can be seen from Figure 3, the board has four positions on a header labelled 1 through 4 alongside some text informing one of the available speeds. Connecting across jumper position 1 will give 1MHz operation, 2 will give 2MHz, and 3 will give 4MHz from the divider circuit. Position four is for connecting an external clock to the system bus and *must* not be shorted or the clock line will be tied to ground!

The recommended position is as fast as your hardware will allow. Most of the time this is 4MHz. There is also a header which provides LED output for the status of the IRQ and NMI lines to assist the programmer with interrupts. For the assembly language programmer or the hardware developer, the WDC65C02 datasheet is a must read and is included. For the TowerBASIC user, it is usually enough to know how fast this processor is running.

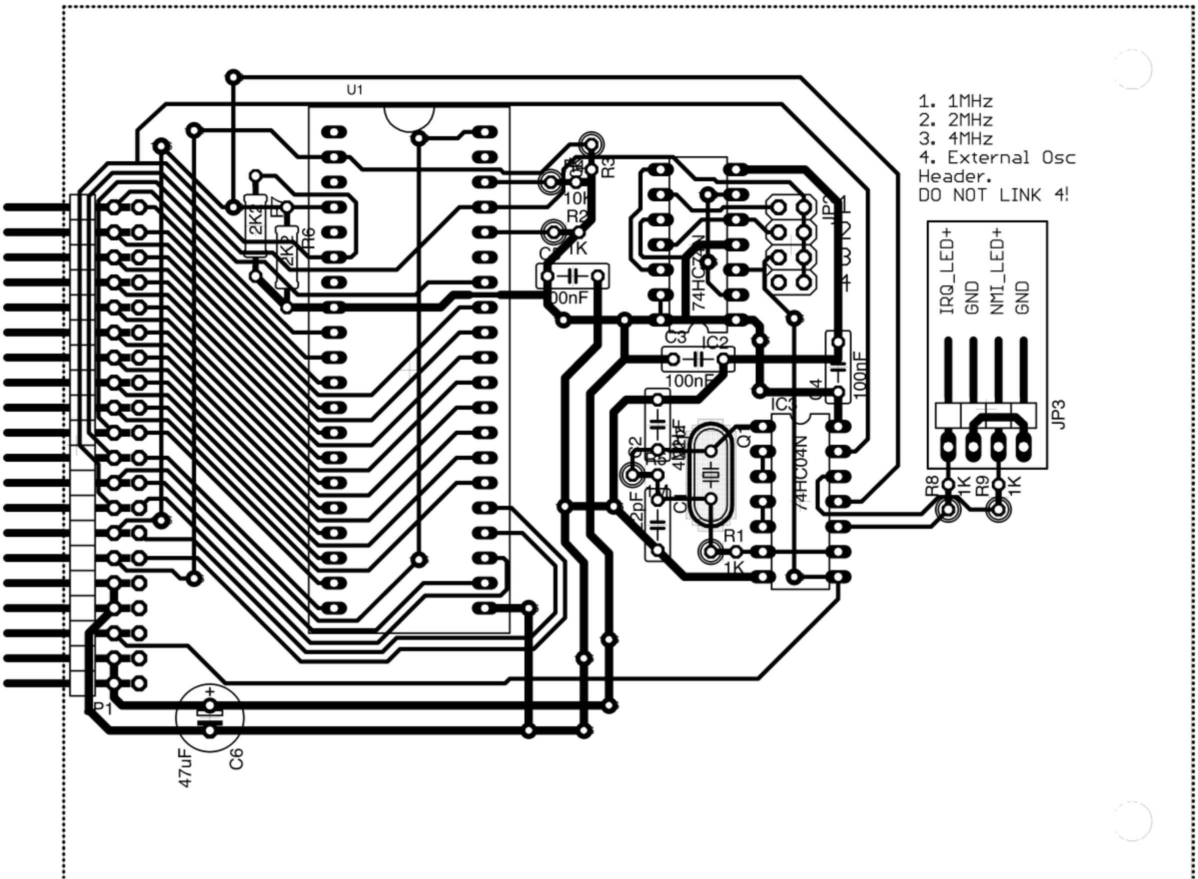



*Figure 3*

# Chapter 11. Tower 8 Card & CPU Backplane.

## CPU Backplane Description

As the system grew, the original backplanes became a little *constraining* and so was born the 8 slot integrated CPU backplane. On the backplane there are 8 TowerBUS slots, one W65C02S6-TPG14 CPU, one clock generator (which can be disabled), IRQ and NMI lights header, reset header, power input terminals and the logic to decode bank selection and IO page selection. This does not handle sideways bank selection, that is done by the memory card.

It should be noted that the 8 slot and CPU backplane needs either the extension piece with two extra joining bars *or* the larger modular case sections. Both the new motherboard and the older 7 slot one can use the same endcaps, joining bars and power supply module.

## TowerBUS Pinout and Specification

There are 40 pins in two rows of 20. Each pin is assigned a signal except for the 5V and ground pins which are assigned in pairs.

Each signal pin is compatible with 74HC series logic and there is no buffering so be mindful of fanout, taking note of the W65C22S6-TPG14 datasheet for the CPU bus pins and the Lattice GAL22V10D-7LP datasheet for the bank selection pins. The power supply is rated at 1.4A but the supplied fuse is an F1A M205. Do not exceed 1.4A even with a bigger fuse fitted.

| *TowerBUS Pinout* | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Top row LH Side | nIOSEL | nBANK3 | nBANJ2 | nBANK1 | nBANK0 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 |
| Pin No. | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 |
| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 |
| Bottom Row LH Side | VCC | VCC | OSC | GND | GND | RDY | IRQB | NMIB | RESB | PHI2 | RWB | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | A0 |

## *Table of Pins/Signals*

| | | |
|---|---|---|
| 1 | nIOSEL | Signals to hardware that the IO address range is being selected. |
| 2 | VCC | +5V power output. Tied to pin 4. |
| 3 | nBANK3 | Signals to the hardware that memory bank 3 is being selected. |
| 4 | VCC | +5V power output. Tied to pin 3. |
| 5 | nBANK2 | Signals to the hardware that memory bank 2 is being selected. |
| 6 | OSC | This is the 4MHz oscillator output for the system. |
| 7 | nBANK1 | Signals to the hardware that memory bank 1 is being selected. |
| 8 | GND | Return path to ground. Tied to pin 10. |
| 9 | nBANK0 | Signals to the hardware that memory bank 0 is being selected. |
| 10 | GND | Return path to ground. Tied to pin 8. |
| 11 | A15 | Address line 15 from the CPU. |
| 12 | RDY | Ready pin for use arbitrating the bus usage of the CPU. |
| 13 | A14 | Address line 14 from the CPU. |
| 14 | IRQB | Interrupt request signal used by hardware to interrupt the CPU. |
| 15 | A13 | Address line 13 from the CPU. |
| 16 | NMIB | Non-Maskable interrupt request signal used by the hardware to interrupt the CPU. |
| 17 | A12 | Address line 12 from the CPU. |
| 18 | RESB | Reset line. Briefly pulled low by the reset circuit on the PSU board and optionally by external hardware. |
| 19 | A11 | Address line 11 from the CPU. |
| 20 | PHI2 | Processor clock generator, used extensively around the system to provide synchronisation of various hardware connected to the TowerBUS. This is the rate at which the CPU operates and is set by jumpers. |
| 21 | A10 | Address line 10 from the CPU. |
| 22 | RWB | Signals that the CPU is reading the bus when high, writing when low. Must be used in conjunction with a means of ensuring processor timing requirements are met. |
| 23 | A9 | Address line 9 from the CPU. |
| 24 | D7 | Data bit 7 of the TowerBUS. |
| 25 | A8 | Address line 8 from the CPU. |
| 26 | D6 | Data bit 7 of the TowerBUS. |
| 27 | A7 | Address line 7 from the CPU. |
| 28 | D5 | Data bit 5 of the TowerBUS. |
| 29 | A6 | Address line 6 from the CPU. |
| 30 | D4 | Data bit 4 of the TowerBUS. |
| 31 | A5 | Address line 5 from the CPU. |
| 32 | D3 | Data bit 3 of the TowerBUS. |
| 33 | A4 | Address line 4 from the CPU. |
| 34 | D2 | Data bit 2 of the TowerBUS. |

| | | |
|---|---|---|
| 35 | A3 | Address line 3 from the CPU. |
| 36 | D1 | Data bit 1 of the TowerBUS. |
| 37 | A2 | Address line 2 from the CPU. |
| 38 | D0 | Data bit 0 of the TowerBUS. |
| 39 | A1 | Address line 1 from the CPU. |
| 40 | A0 | Address line 0 from the CPU. |

# Chapter 12. Tower Peripheral Bus & Centronics Interface.

## Tower Peripheral Bus and Centronics Card Description

This card provides both multi-drop serial communications (Tower Peripheral Bus) and Centronics printer support. The TPB bus *in particular*, has very complex behaviour and many function calls. This bus is a work-in-progress at present, but the hardware is functional. This card has a base address of $C020 and uses up-to $C02F.

## The Centronics Port

This is modelled after the BBC micro implementation and may serve to output any characters as the user wishes via the OUTP_V, or more directly via either the vectored system call TPB_LPT_write_vec ($FF99) or by direct hardware access, though this is discouraged. To use TPB_LPT_write_vec, place the character to be written into the accumulator and call TPB_LPT_write_vec. This presently a blocking call and if the printer hangs the Centronics bus in a way you can't clear printer-side, you will have to perform a warm start.

To direct your output stream to also go to the LPT port, one usually sets os_outsel ($5E0) bit 2 to 1, everything then also going to the Centronics port. Clearing bit 2 stops this.

| *Centronics Port Pinout* | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Top | NC | NC | NC | ACK | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 | STROBE |
| | 25 | 23 | 21 | 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| Bottom | 26 | 24 | 22 | 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| | NC | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND |

# Chapter 13. The IRQ Handler Sub-system.

## IRQ Handler Sub-System Description

Interrupts make it easier to handle outside events such as incoming serial data, timing, and software exceptions. The ToE's processor furnishes only rudimentary interrupt handling, and this sub-system is provided to flexibly manage multiple interrupts. The way this is achieved is by use of a table of eight vectors processed one by one, using a bitfield to select which one's are active. Initially, the table is populated with a null handler such that if one of its entries is inadvertently enabled, it is handled gracefully, rather than crashing the system.

The means by which one sets or clears an interrupt vector are by the provision of atomic calls which retain the interrupt handling state as much as possible so that other interrupts are delayed as little as possible.

## IRQ System Call Vectors

| | |
|---|---|
| $FFDB | IRQH_Handler_Init_vec |
| $FFDE | IRQH_SetIRQ_vec |
| $FFE1 | IRQH_ClrIRQ_vec |
| $FFE4 | IRQH_SystemReport_vec |

### $FFDB IRQH_Handler_Init_vec

Calling this vector clears the interrupt vector table and resets the IRQ Handler sub-system. It may be that individual interrupt vectors get a reset vector table too, but this is not a given yet.

### $FFDE IRQH_SetIRQ_vec

Atomically transfers an interrupt's vector from **IRQ_CallReg** to the table location specified in A. Locations available are 0 through 7 with 0 being handled first, subsequent locations being handled in sequential order. Note that this does not affect whether the interrupt vector is selected or not. It is the users' responsibility to manage this themselves.

### $FFE1 IRQH_ClrIRQ_vec

Atomically transfers the null vector to the table location specified in the accumulator. Locations available are 0 through 7 with 0 being handled first, subsequent locations being handled in sequential order. Note that this does not affect whether the interrupt vector is selected or not. It is the users' responsibility to manage this themselves.

**$FFE4 IRQH_SystemReport_vec**

Calling this returns the base address of the IRQ handler table, including all variables. This is done so that the table location does not need to be known in advance. Given that this data structure is very likely to change, this will prevent breakage of code written for the ToE.

Upon return, X will contain **IRQH_Table_Base** low byte, Y will contain its high byte, and A will contain the IRQ handler version.

## *IRQH_Table_Base Structure*

| Offset | Name | Purpose |
|---|---|---|
| $0-$15 | IRQH_CallList | Eight consecutive little-endian call addresses for the users' IRQ device handlers. |
| $16-$17 | IRQH_CallReg | Intermediate register for atomically transferring addresses to the call list above. |
| $18 | IRQH_ClaimsList | Bitfield showing which interrupts claimed and thus serviced an interrupt. |
| $19 | IRQH_MaskByte | Bitfield for selecting which interrupts are to be serviced in the event of the IRQ vector being invoked. |
| $20 | IRQH_WorkingMask | Internal variable, this walks from the LSb to the MSb and is used for various parts of the process. Do not write to this variable. |
| $21-$31 | IRQH_CMD_Table | Table of IRQ Commands and parameter values. |

## Command Codes

Each IRQ has an associated parameter and command code byte entry in the IRQH_CMD_Table. These are sorted as follows. First is the parameter, followed by the command code.

Each IRQ must check on entry and before claiming IRQ for the following commands:

| | |
|---|---|
| 0. IRQH_Service_CMD | Instructs the IRQ that service is required. |
| 1. IRQH_Shutdown_CMD | The IRQ must perform an orderly shutdown. |
| 2. IRQH_Reset_CMD | The IRQ must reset to an initial state. |

Further commands may be implemented by the user but values below 8 are presently reserved for future upgrades.

## Starting an IRQ Service

Starting and IRQ consists of writing it's address to the IRQH_CallReg ($A30-$A31), Loading the accumulator with our chosen IRQH_CallList location (0-7), calling IRQH_SetIRQ_vec to load it atomically and then calling our IRQ's initialisation routine.

The IRQ initialisation routine should handle starting of the IRQ as necessary. To return the IRQ vector to the system, there is a separate call IRQH_ClrIRQ_vec ($FFE1).

## Servicing an IRQ

IRQ's running through the IRQ Handler must check upon entry, their associated command entry, pointed to by IRQH_CMD_Table + the X index register. The IRQ must process at least the minimum system command codes listed above. Parameters for the minimum set are not required.

Each time the IRQ is called, it must check that the hardware associated with it has generated an interrupt, set the appropriate claim bit in IRQH_ClaimsList ($A32) by ORing IRQH_WorkingMask ($A34) into it and clear the hardware IRQ signal so as not to cause nuisance IRQ calls and system hangs. IRQ's not generated by the associated hardware must not cause undue resource wastage or set the claim bit.

Exit from the IRQ routine is by RTS not RTI. This is so that other IRQ routine's may check their associated hardware and commands before the handler hands control back to the system.

## Stopping an IRQ

Sending the IRQH_Shutdown_CMD (1) to the IRQ will cause the IRQ to stop. This does not remove it from the IRQH_CallList but does allow the IRQ the chance to stop in an appropriate manner. Alternatively, an atomic call may be created to do the same.

A stopped IRQ may be re-started at any time by either calling its initialisation routine or if appropriate, setting its associated bit in IRQH_MaskByte ($A33).

## Removing an IRQ Service from the IRQH_CallList ($A20-$A30)

One must have first stopped the IRQ, then the accumulator must be loaded with the correct table entry. This is followed by calling IRQH_ClrIRQ_vec, which handles this atomically. After clearing, the table contains a safe dummy entry that points to a function that does nothing but ensure the IRQH_ClaimsList is correct before returning control to the IRQ handler.

# Chapter 14. The Countdown IRQ Handler Sub-system.

## Countdown IRQ Subsystem Description

This provides a countdown timer with programmable count rate. A GPIO card or other hardware containing a W65C22 must be installed with its W65C22 base address at $C020 for this to work as it relies on the W65C22's Timer 1 to generate a regular stream of interrupts. This 6522 is chosen as it provides cassette storage signals and is normally expected to be present at this base address.

The IRQ is initialised at start-up in prime position at IRQH_CallList ($A20) position 0.

The default count interval is set at 39999 PHI2 ticks/count giving 10mS tick at 4MHz. It is the users' responsibility to ensure an appropriate reload value is set for proper operation.

There is only one vector associated with the countdown timer, INIT_COUNTDOWN_IRQ_vec ($FFE7). Calling this with its IRQ mask bit in the accumulator (1 unless the user moves the IRQ to a less prime location) initialises the countdown timer, which will update the countdown variable each count until it reaches 0, after which it shuts down.

It is possible to alter the IRQ reload value mid countdown, but one must either set the interrupt mask bit before updating or adverse effects may occur.

## System Variables for the Countdown IRQ

| | | |
|---|---|---|
| ($A46-$A47) | CTR_V | Counts down from its initial value until zero is reached. |
| ($A48-$A50) | CTR_LOAD_VAL_V | The T1 reload value used upon initialisation. Changing this alters the IRQ rate. Caution is advised that setting this at or close to zero will cause the system to become unresponsive since it will use up all available CPU time. |

# Chapter 15. The I2C-Bus Engine Sub-system.

## I2C Engine General Description

This I2C-bus engine is provided such that those who are sufficiently experienced in such things can have a multitude of I2C-bus devices operated from the computer with lower effort. Whilst it is not the fastest implementation by far, it still affords a huge range of possibilities, from ADCs and DACs to digital IO expanders, digital pots, sensors, programmable oscillators, tuners and memories both volatile and non-volatile just to name a few. It should therefore be a very useful addition.

The $I^2C$-bus is implemented by software alone and only requires a GPIO card mapped to $CO40 such that it can use port A. Pin PA0 is SDA (Serial Data) and pin PA1 is SCL (Serial CLock).

Data is transmitted at somewhat below the $I^2C$-bus standard speed specification of 100KHz and depends on both the system clock and interrupt load. Timings are thus slightly irregular but perfectly useable.

There are calls to Initialise the I2C-Subsystem, send a (re)start, send a stop, output a byte and input a byte at present. Soon to implemented are calls to send an I2C 7-bit address with the relevant read or write bit set.

Before using the $I^2C$ subsystem, one needs to initialise it by calling I2C_Init_vec ($FF81).

There are vectors for Start, Stop, Out and In. A start may be sent as a re-start.

The user circuit must provide suitable pull-ups as these lines are driven as open-drain to facilitate multiple $I^2C$ slave devices.

For a fuller understanding of the $I^2C$-bus and protocol, one should read documents such as NXP's UM10204 $I^2C$-bus specification and user manual.

## TowerBASIC Commands for I2C communications

**`I2C_INIT`** is provided to setup the $I^2C$-bus engine and pins. It should be used *before* any other $I^2C$ commands.

**`I2C_START`** sends the start (S) condition on the bus.

**`I2C_STOP`** sends the stop (P) condition on the bus, freeing it.

**`I2C_OUT()`** is used to transmit a byte to the $I^2C$-bus. The returned byte has ACK (0)/NAK (1) returned in bit 0 and if the bus times out then bit 1 will be set. To send a device address, it should be in the top seven bits with bit 0 being the read/not write bit.

**`I2C_IN()`** reads a byte from the $I^2C$-bus and sends either an ACK (0) or NAK (1) which the user passes to the function.

## List of $I^2C$-Bus Subsystem Variables

| | |
|---|---|
| I2C_Status ($5D0) | $I^2C$ Subsystem status register. There is an associated bitfield for this. |
| I2C_Byte ($5D1) | The byte to be transmitted or which has just been received. |
| I2C_Timeout_V ($5D1-$5D2) | Contains the timeout counter variable. This is used to reload an internal counter which the $I^2C$ subsystem uses to determine how long to wait before giving up. Used to prevent hangs in the event the slave device does not respond. |

## I2C_Status ($5D0) Bitfield

| | |
|---|---|
| Bits 7-4 | Not used. |
| Bit 3 | I2C_STA_Master. Currently, the $I^2C$ engine is always the master. This bit is presently ignored. |
| Bit 2 | I2C_STA_Rd_nWr. Determines whether the $I^2C$-bus is sending a Read or Write to the slave device. Currently not implemented. |
| Bit 1 | I2C_STA_Timeout. Indicates whether the $I^2C$-bus engine timed out in its last operation. |
| Bit 0 | I2C_STA_NAK. Used to signal to the slave either an ACKnowledge or negative ACKnowledge at the end of a byte transmission. |

## I2C-Subsystem Calls

| | |
|---|---|
| I2C_Init_vec ($FF81) | Calling this initialises the $I^2C$-bus engine including the IO pins SDA (PA0) and SCL (PA1). |
| I2C_Start_vec ($FF84) | Sends a start condition to the slave. |
| I2C_Stop_vec ($FF87) | Sends a stop condition to the slave. |
| I2C_Out_vec ($FF8A) | Transmits the byte placed in I2C_Byte ($5D1). The I2C_STA_NAK (bit 0) is copied from I2C_Status ($5D1) also and should the device have timed out, the I2C_STA_Timeout (bit 1) will also be set. |
| I2C_In_Vec ($FF8D) | Reads a byte from the slave device, placing it in I2C_Byte ($5D1). |

# Chapter 16. The SPI Sub-system.

## SPI Subsystem Description

Some peripherals one might attach require that they be interfaced using an interfacing standard called Serial Peripheral Interface. This is an interface that transfers data serially, transferring one bit in and on bit out, synchronised by a clock signal. There can be multiple devices sharing an SPI bus, as long as each device has its own select pin. In many cases, this is called slave select, shortened to SS, but it is also often called Chip Select. The assertion level of one's SPI devices can be both negative and positive logic on both the clock and selection pins, but not the data pins. Normally, SPI devices accept data starting with the MSb in words of 8 bits.

The SPI engine implemented here always transmits MSb first in 8-bit words but drives the SS (device select) pin and SCK (shift clock) in either positive or negative logic. The SPI engine can also support all three SPI modes.

The SPI engine operates on the following named pins: -

| | |
|---|---|
| MOSI (Master Out, Slave In). | Data is sent to the slave on this pin. |
| MISO (Master In Slave Out). | Data is received from the slave on this pin. |
| SCK (Shift ClocK). | This signal provided the clock edge for each bit. |
| SS (Slave Select). | Provides a means of selecting the appropriate device. |

It is possible to specify which pin to use for MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCK (Shift ClocK) and SS (Slave Select). By changing the settings, one can use multiple devices on the same SPI bus. The SPI bus only operates on Port A of the primary GPIO card at a base address of $C040 for now, however.

Because all these pins are being driven directly from GPIO port A, which also provides the I2C port on pins PA0 and PA1, if you intend on using both, you will have to take care. Also, these pins are 5V CMOS IO pins and not all SPI devices will be logic level compatible. It is beyond the scope of this manual to explain the intricacies of bridging differing logic families.

SPI speeds achieved on this bus are software and processor clock limited and if you require faster speeds, hardware assistance will be required.

## Using the SPI engine

There is a data structure called SPI_Struct. SPI Struct starts at $400 and its fields are listed in the table below.

| SPI_Struct ($400-$408) | | | |
|---|---|---|---|
| Field name | Function | Address | Default values. |
| SPI_In | Byte received | $400 | 0 |
| SPI_Out | Byte to be sent | $401 | 0 |
| SPI Mode | See table below | $402 | 0 |
| SPI_SS_Pin | The bit is set is SS | $403 | 0b00100000 |
| SPI_SS_Act | 1=Active Low, 0=High | $404 | 1 |
| SPI_MOSI_Pin | The bit set is MOSI | $405 | 0b00001000 |
| SPI_MISO_Pin | The bit set is MISO | $406 | 0b00010000 |
| SPI_SCK_Pin | The bit set is SCK | $407 | 0b00000100 |
| SPI_Temp | Internally used | $408 | xx. Not for user |

Calling SPI_Struct_Init_vec ($FF57) prefills SPI_Struct ($400-$408) with default values.

To transfer over SPI, configure the engine by loading sane values into SPI_Struct ($400-$408) and call SPI_Init_vec ($FF5A) to get everything ready. Then load SPI_Out ($401) before each transfer and call SPI_Xfer_vec ($FF5D) to make the transfer. Upon return, SPI_In ($400) should contain any returned byte. One can transfer as many bytes as one desires without reconfiguring the engine, or even change the SS pin to use a different SPI device. It is not recommended to change the SS pin active level between devices.

## List of SPI Function Calls

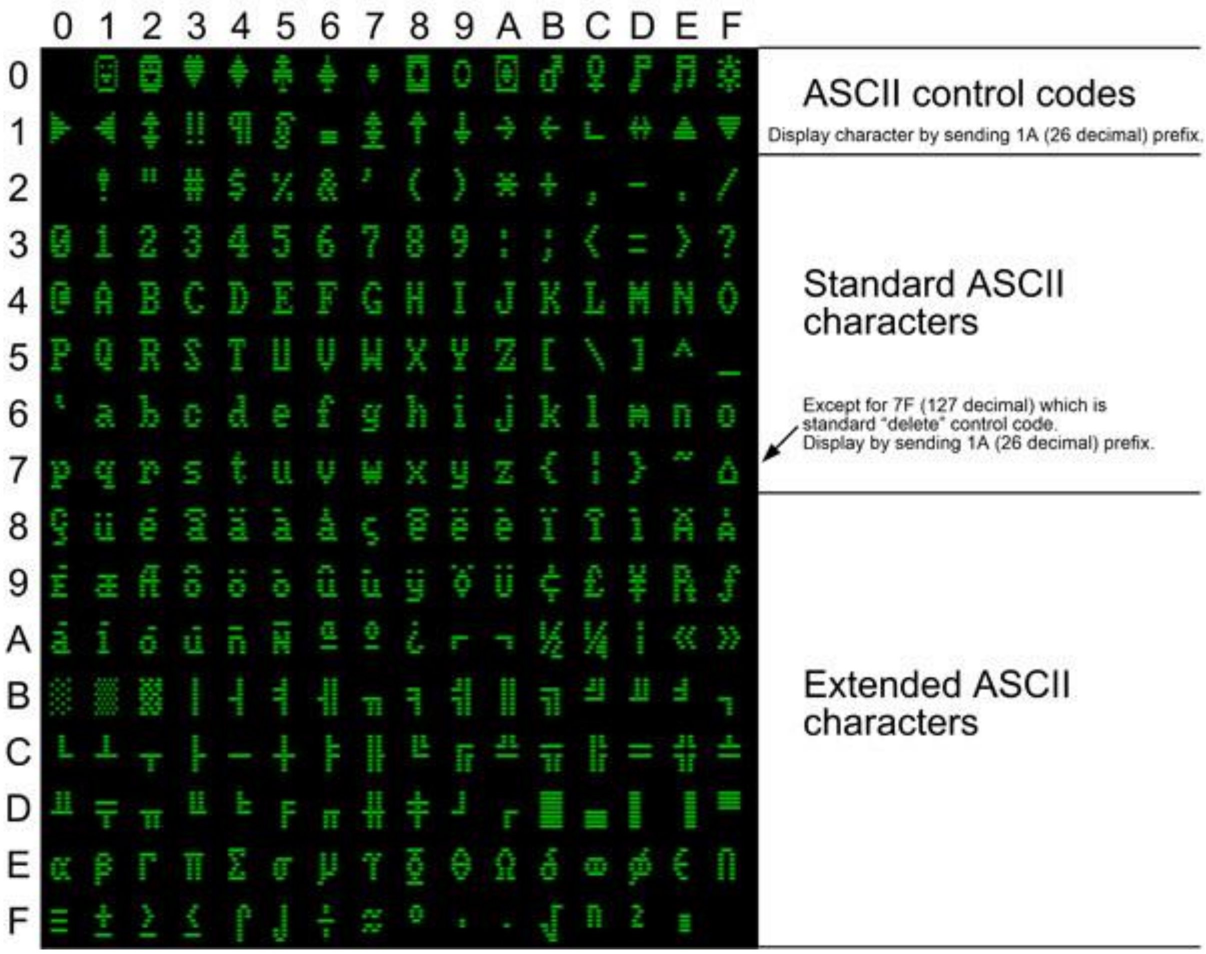
| | |
|---|---|
| SPI_Struct_Init_vec ($FF57) | Initialises SPI_Struct ($400-$408) with sensible safe values useful to a wide range of SPI slaves. |
| SPI_Init_vec ($FF5A) | Initialises the SPI engine and pins ready for transmissions. |
| SPI_Xfer_vec ($FF5D) | Transfers one byte into SPI_In ($400) and one byte out from SPI_Out ($401). |

| Table of SPI Modes | | |
|---|---|---|
| Mode 0 | CPHA0, CPOL0 | Positive clock, transfer on leading edge of clock. |
| Mode 1 | CPHA1, CPOL0 | Positive clock, transfer on trailing edge of clock. |
| Mode 2 | CPHA0, CPOL1 | Negative clock, transfer on leading edge of clock. |
| Mode 3 | CPHA1, CPOL1 | Negative clock, transfer on trailing edge of clock. |

# Appendices

## Appendix A. PC-DOS Character Set

*The row gives the most significant digit and the column the least. This chart is in hexadecimal.*



# Appendix B. Display Layout

| 80 Column Mode (Text) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Row 0, Col 0 | ... | ... | ... | ... | ... | ... | ...Row 0, Col 79 |
| ... | ... | ... | ... | ... | ... | ... | |
| ... | ... | ... | ... | ... | ... | ... | |
| Row 24,0 | ... | ... | ... | ... | ... | ... | ...Row 24, Col 79 |

| 40 Column Mode (Text) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Row 0, Col 0 | ... | ... | ... | ... | ... | ... | ...Row 0, Col 39 |
| ... | ... | ... | ... | ... | ... | ... | |
| ... | ... | ... | ... | ... | ... | ... | |
| Row 24,0 | ... | ... | ... | ... | ... | ... | ...Row 24, Col 39 |

| Graphics (Bloxel Graphics) | | | | | | | |
|---|---|---|---|---|---|---|---|
| X0,Y0 | ... | ... | ... | ... | ... | ... | ...X159,Y0 |
| ... | ... | ... | ... | ... | ... | ... | |
| ... | ... | ... | ... | ... | ... | ... | |
| X0,Y99 | ... | ... | ... | ... | ... | ... | ...X159,Y99 |

## Appendix C. Video Control Codes

| | | |
|---|---|---|
| $01 (01) | Cursor home | *(Standard ASCII)* |
| $02 (02) | Define cursor character (2nd byte is the cursor character, or 0 to turn off) | |
| $03 (03) | Cursor blinking | |
| $04 (04) | Cursor solid | |
| $05 (05) | Set graphics pixel (next two bytes = x,y) | |
| $06 (06) | Reset graphics pixel (next two bytes = x,y) | |
| $08 (08) | Backspace | *(Standard ASCII)* |
| $09 (09) | Tab | *(Standard ASCII)* |
| $0A (10) | Linefeed | *(Standard ASCII)* |
| $0C (12) | Clear screen | *(Standard ASCII)* |
| $0D (13) | Carriage return | *(Standard ASCII)* |
| $0E (14) | Set column 0 to 79 (2nd byte is the column number) or 0 to 39 for a 40 char line | |
| $0F (15) | Set row 0 to 24 (2nd byte is the row number) | |
| $10 (16) | Delete start of line | |
| $11 (17) | Delete to end of line | |
| $12 (18) | Delete to start of screen | |
| $13 (19) | Delete to end of screen | |
| $14 (20) | Scroll up | |
| $15 (21) | Scroll down | |
| $16 (22) | Scroll left | |
| $17 (23) | Scroll right | |
| $18 (24) | Set font attribute for the current line | |
| $1A (26) | Treat next byte as a character (to allow PC DOS char codes 1 to 31 to be displayed on screen) | |
| $1B (27) | ESC - reserved for ANSI sequences | |
| $1C (28) | Cursor right | |
| $1D (29) | Cursor Left | |
| $1E (30) | Cursor up | |
| $1F (31) | Cursor down | |
| $20 (32) to... | | |
| $7E (126) | Standard ASCII codes | |
| $7F (127) | Delete | |
| $80 (128) to... | | |
| $FF (255) | PC (DOS) extended characters | |

## Appendix D. Character Control Bits

| *Character Control Attribute $18 (24) and bitfield of following byte* | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Graphics | Spare | Spare | Spare | Spare | Double Height | Bold | 80 Columns |

## Appendix E. ANSI Card Register Table

| | ANSI Function |
|---|---|
| 0 | Output byte (Output Register B) |
| 1 | PA7 = Ack, PA6 = Avail, PA5-0 not used. (Output Register A) |
| 2 | Data Direction Register B. Should be set to $FF |
| 3 | Data Direction Register A. Should be set to $40 |

It should be noted that should one change the direction registers to incorrect values, there will be contention between the ANSI processor and the W65C22S6-TPG14. This not only will prevent the ANSI card from functioning but will result in increased pin currents. Try to keep the IO directions set correctly wherever possible.

# Appendix F. Tower of Eightness Memory Overview

## *Bank Structure and IOMAP oveview.*

| | Page 0 | Page 1 |
|---|---|---|
| Bank 3 $C000-$FFFF | TowerOS, BASIC and FS | TowerOS, BASIC and FS |
| Bank 2 $8000-$BFFF | RAM. Ramtop is normally at the top of this page. | RAM |
| Bank 1 $4000-$7FFF | RAM | RAM |
| Bank 0 $0000-$3FFF | RAM. Pages 0 & 1 are special. Take care when switching | RAM. Pages 0 & 1 are special. Take care when switching |

| IO Page $C000-$C0FF |
|---|
| $C0FF - Bank Select |
| $C0E0-$C0EF AY Soundcard |
| $C060-$C0FE Uncommitted. |
| $C050-$C05F GPIO Card 2 (if present) |
| $C040-$C04F GPIO Card 1 |
| $C030-$C03F Uncommitted |
| $C020-$C02F TPB/Centronics Card |
| $C014-$C017 ACIA 2 (If present) |
| $C010-$C013 ACIA 1 |
| $C000-$C00F ANSI Card |

The below register selects which page is in any given bank. This register is write-only and if you don't keep track of it, you might find yourself in the wrong place in memory.

*Bank Select Register Bitfield ($C0FF)*

| Bit | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| Default | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Reserved. Keep at 0. | | | | Bank 3 Page 1 Select (EEPROM) | Bank 2 Page 1 Select | Bank 1 Page 1 Select | Bank 0 Page 1 Select (Warning CPU Page 0 & 1 included) |

## Appendix G. AY Hardware Details

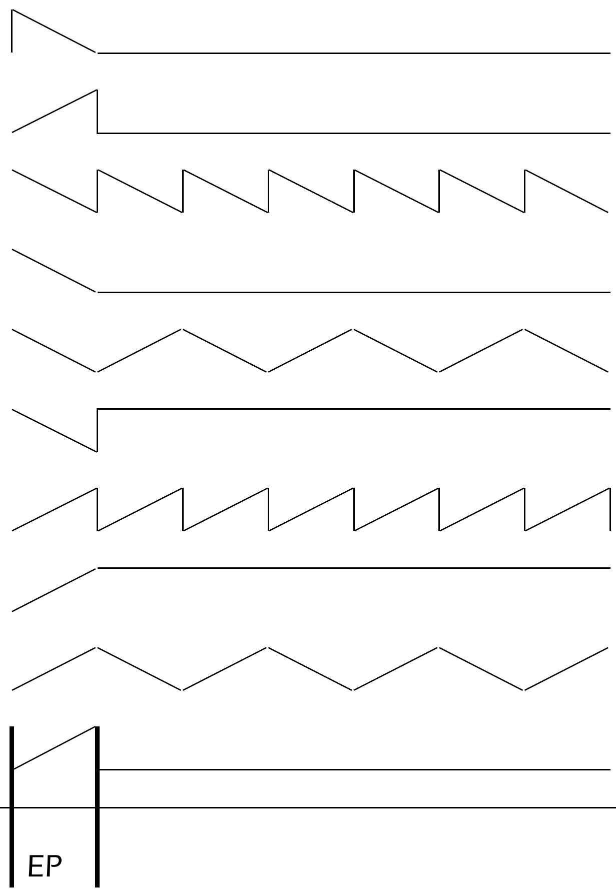
*AY-3-8912A Register Map*

| Reg | Bit | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|
| R0 | Channel A Tone Period | 8-Bit fine tune A. | | | | | | | |
| R1 | Channel A Tone Period | | | | | 4-Bit coarse tune A. | | | |
| R2 | Channel B Tone Period | 8-Bit fine tune B. | | | | | | | |
| R3 | Channel B Tone Period | | | | | 4-Bit coarse tune B. | | | |
| R4 | Channel C Tone Period | 8-Bit fine tune C. | | | | | | | |
| R5 | Channel C Tone Period | | | | | 4-Bit coarse tune C. | | | |
| R6 | Noise Period | | | | 5-Bit Period control. | | | | |
| R7 | nEnable | nIOB | nIOA | nNOISE Ch C | nNOISE Ch B | nNOISE Ch A | nTONE Ch C | nTONE Ch B | nTONE Ch A |
| R8 | Channel A Amplitude | | | | M | L3 | L2 | L1 | L0 |
| R9 | Channel B Amplitude | | | | M | L3 | L2 | L1 | L0 |
| R10 | Channel C Amplitude | | | | M | L3 | L2 | L1 | L0 |
| R11 | Envelope Period | 8-Bit Fine Tune Envelope | | | | | | | |
| R12 | Envelope Period | 8-Bit Coarse Tune Envelope | | | | | | | |
| R13 | Envelope Shape/Cycle | | | | | Continue | Attack | Alternate | Hold |
| R14 | I/O Port A Data Store | 8-Bit parallel I/O on Port A. Not connected to anything. | | | | | | | |
| R15 | I/O Port B Data Store | 8-Bit parallel I/O on Port B. Pins not available on AY-3-8912A | | | | | | | |

*Table of Bus States*

| BC1 | BDIR | State |
|---|---|---|
| 0 | 0 | Inactive. |
| 1 | 0 | Read from AY. The AY selected register is on the data lines. |
| 0 | 1 | Write to AY. The data lines are transferred to the AY |
| 1 | 1 | Latch Address. Write register address to AY from data lines. |

| *Envelope Shape/Cycle Operation* | | | | |
|---|---|---|---|---|
| Mode bits (AY reg 13) | | | | |
| B3 | B2 | B1 | B0 | |
| Continue | Attack | Alternate | hold | Graphic Representation of Envelope Generator Output. |
| 0 | 0 | X | X | |
| 0 | 1 | X | X | |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 1 | |




**EP** is the envelope period (duration of one cycle).

NOTE: - The AY clock from which all periods is generated is 1.842MHz.

# Appendix H. Pinouts

## TowerBUS Pinout

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Top row LH Side | nIOSEL | nBANK3 | nBANK2 | nBANK1 | nBANK0 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 |
| Pin No. | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 |
| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 |
| Bottom Row LH Side | VCC | VCC | OSC | GND | GND | RDY | IRQB | NMIB | RESB | PHI2 | RWB | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | A0 |

## Centronics Port Pinout

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Top | NC | NC | NC | ACK | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 | STROBE |
| | 25 | 23 | 21 | 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| Bottom | 26 | 24 | 22 | 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| | NC | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND | GND |

## GPIO Card Port Pinouts

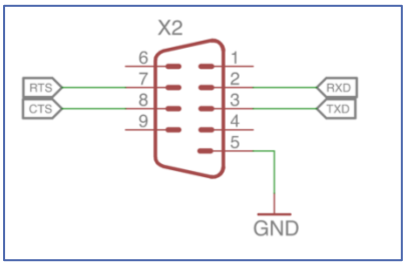
**Port A. Base address offset 1, Data Direction Register Offset 3.**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| 0V | 0V | 0V | 0V | 0V | 0V | 0V | 0V | +5V | +5V |
| 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 | CA2 | CA1 |

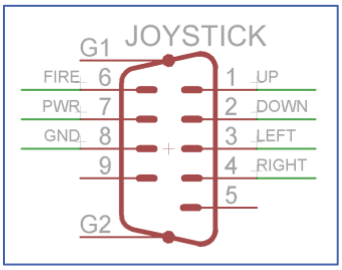
**Port B. Base address offset 0, Data Direction Register Offset 2.**

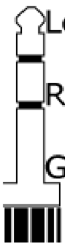
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| 0V | 0V | 0V | 0V | 0V | 0V | 0V | 0V | +5V | +5V |
| 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 | CB2 | CB1 |

## *ACIA Port Pinout*




## *Joystick Port Pinout*




| *AY Soundcard Output Jack Pinout* | | |
|---|---|---|
|  Left | Tip | Left Audio Out |
| Right | Ring | Right Audio Out |
| Ground | Sleeve | Ground |

# Appendix I. W65C02 Instruction List

| Mnemonic | Operation<br># Immediate<br>~ NOT<br>^ AND<br>v OR<br>xv XOR | a | (a,x) | a,x | a,y | (a) | A | # | i | r | s | zp | (zp,x) | zp,x | zp,y | (zp) | (zp),y | Processor Status Register (P) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | N | V | 1 | 1 | D | I | Z | C |
| ADC | A+M+C→A | 6D | | 7D | 79 | | | 69 | | | | 65 | 61 | 75 | | 72 | 71 | N | V | | | | | Z | C |
| AND | A^M→A | 2D | | 3D | 39 | | | 29 | | | | 25 | 21 | 35 | | 32 | 31 | N | | | | | | Z | |
| ASL | C←7..0←0 | 0E | | 1E | | | 0A | | | | | 06 | | 16 | | | | N | | | | | | Z | $M_7$ |
| BBR0 | Branch on b0 reset | | | | | | | 0F | | | | | | | | | | | | | | | | | |
| BBR1 | Branch on b1 reset | | | | | | | 1F | | | | | | | | | | | | | | | | | |
| BBR2 | Branch on b2 reset | | | | | | | 2F | | | | | | | | | | | | | | | | | |
| BBR3 | Branch on b3 reset | | | | | | | 3F | | | | | | | | | | | | | | | | | |
| BBR4 | Branch on b4 reset | | | | | | | 4F | | | | | | | | | | | | | | | | | |
| BBR5 | Branch on b5 reset | | | | | | | 5F | | | | | | | | | | | | | | | | | |
| BBR6 | Branch on b6 reset | | | | | | | 6F | | | | | | | | | | | | | | | | | |
| BBR7 | Branch on b7 reset | | | | | | | 7F | | | | | | | | | | | | | | | | | |
| BBS0 | Branch on b0 set | | | | | | | 8F | | | | | | | | | | | | | | | | | |
| BBS1 | Branch on b1 set | | | | | | | 9F | | | | | | | | | | | | | | | | | |
| BBS2 | Branch on b2 set | | | | | | | AF | | | | | | | | | | | | | | | | | |
| BBS3 | Branch on b3 set | | | | | | | BF | | | | | | | | | | | | | | | | | |
| BBS4 | Branch on b4 set | | | | | | | CF | | | | | | | | | | | | | | | | | |
| BBS5 | Branch on b5 set | | | | | | | DF | | | | | | | | | | | | | | | | | |
| BBS6 | Branch on b6 set | | | | | | | EF | | | | | | | | | | | | | | | | | |
| BBS7 | Branch on b7 set | | | | | | | FF | | | | | | | | | | | | | | | | | |
| BCC | Branch on C=0 | | | | | | | 90 | | | | | | | | | | | | | | | | | |
| BCS | Branch on C=1 | | | | | | | B0 | | | | | | | | | | | | | | | | | |
| BEQ | Branch if Z=1 | | | | | | | F0 | | | | | | | | | | | | | | | | | |
| BIT | A^M | 2C | | 3C | | | | 89 | | 24 | | 34 | | | | | | $M_7$ | $M_6$ | | | | | Z | |
| BMI | Branch if N=1 | | | | | | | 30 | | | | | | | | | | | | | | | | | |
| BNE | Branch if Z=0 | | | | | | | D0 | | | | | | | | | | | | | | | | | |
| BPL | Branch if N=0 | | | | | | | 10 | | | | | | | | | | | | | | | | | |
| BRA | Branch Always | | | | | | | 80 | | | | | | | | | | | | | | | | | |
| BRK | Break | | | | | | | | 00 | | | | | | | | | | | | 1 | 0 | 1 | | |
| BVC | Branch if V=0 | | | | | | | 50 | | | | | | | | | | | | | | | | | |
| BVS | Branch if V=1 | | | | | | | 70 | | | | | | | | | | | | | | | | | |
| CLC | 0→C | | | | | | 18 | | | | | | | | | | | | | | | | | | 0 |
| CLD | 0→D | | | | | | D8 | | | | | | | | | | | | | | | 0 | | | |
| CLI | 0→I | | | | | | 58 | | | | | | | | | | | | | | | | 0 | | |
| CLV | 0→V | | | | | | B8 | | | | | | | | | | | | 0 | | | | | | |
| CMP | A-M | CD | | DD | D9 | | | C9 | | | | C5 | C1 | D5 | | D2 | D1 | N | | | | | | Z | C |
| CPX | X-M | EC | | | | | | E0 | | | | E4 | | | | | | N | | | | | | Z | C |
| CPY | Y-M | CC | | | | | | C0 | | | | C4 | | | | | | N | | | | | | Z | C |
| DEC | Decrement | CE | | DE | | | 3A | | | | | C6 | | D6 | | | | N | | | | | | Z | |
| DEX | X-1→X | | | | | | | | CA | | | | | | | | | N | | | | | | Z | |
| DEY | Y-1→Y | | | | | | | | 88 | | | | | | | | | N | | | | | | Z | |
| EOR | A⊻M→A | 4D | | 5D | 59 | | | 49 | | | | 45 | 41 | 55 | | 52 | 51 | N | | | | | | Z | |
| INC | Increment | EE | | FE | | | 1A | | | | | E6 | | F6 | | | | N | | | | | | Z | |
| INX | X+1→X | | | | | | | | E8 | | | | | | | | | N | | | | | | Z | |
| INY | Y+1→Y | | | | | | | | C8 | | | | | | | | | N | | | | | | Z | |
| JMP | Jump to location | 4C | 7C | | | 6C | | | | | | | | | | | | | | | | | | | |
| JSR | Jump to subroutine | 20 | | | | | | | | | | | | | | | | | | | | | | | |
| LDA | M→A | AD | | BD | B9 | | | A9 | | | | A5 | A1 | B5 | | B2 | B1 | N | | | | | | Z | |
| LDX | M→X | AE | | | BE | | | A2 | | | | A6 | | | B6 | | | N | | | | | | Z | |
| LDY | M→Y | AC | | BC | | | | A0 | | | | A4 | | B4 | | | | N | | | | | | Z | |
| LSR | 0→7..0→C | 4E | | 5E | | | 4A | | | | | 46 | | 56 | | | | 0 | | | | | | Z | $M_0$ |
| NOP | No Operation | | | | | | | | EA | | | | | | | | | | | | | | | | |
| ORA | A∨M→A | 0D | | 1D | 19 | | | 09 | | | | 05 | 01 | 15 | | 12 | 11 | N | | | | | | Z | |

| Mnemonic | Operation # Immediate ~ NOT ^ AND v OR xv XOR | a | (a,x) | a,x | a,y | (a) | A | # | i | r | s | zp | (zp,x) | zp,x | zp,y | (zp) | (zp),y | Processor Status Register (P) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | N | V | 1 | 1 | D | I | Z | C |
| PHA | A→Ms, S-1→S | | | | | | | | | | 48 | | | | | | | N | | | | | | Z | |
| PHP | P→Ms, S-1→S | | | | | | | | | | 08 | | | | | | | | | | | | | | |
| PHX | X→Ms, S-1→S | | | | | | | | | | DA | | | | | | | | | | | | | | |
| PHY | Y→Ms, S-1→S | | | | | | | | | | 5A | | | | | | | | | | | | | | |
| PLA | S+1→S. Ms→A | | | | | | | | | | 68 | | | | | | | N | | | | | | Z | |
| PLP | S+1→S. Ms→P | | | | | | | | | | 28 | | | | | | | N | V | | B | D | I | Z | C |
| PLX | S+1→S. Ms→X | | | | | | | | | | FA | | | | | | | N | | | | | | Z | |
| PLY | S+1→S. Ms→Y | | | | | | | | | | 7A | | | | | | | N | | | | | | Z | |
| RMB0 | Reset Memory b0 | | | | | | | | | | | 07 | | | | | | | | | | | | | |
| RMB1 | Reset Memory b1 | | | | | | | | | | | 17 | | | | | | | | | | | | | |
| RMB2 | Reset Memory b2 | | | | | | | | | | | 27 | | | | | | | | | | | | | |
| RMB3 | Reset Memory b3 | | | | | | | | | | | 37 | | | | | | | | | | | | | |
| RMB4 | Reset Memory b4 | | | | | | | | | | | 47 | | | | | | | | | | | | | |
| RMB5 | Reset Memory b5 | | | | | | | | | | | 57 | | | | | | | | | | | | | |
| RMB6 | Reset Memory b6 | | | | | | | | | | | 67 | | | | | | | | | | | | | |
| RMB7 | Reset Memory b7 | | | | | | | | | | | 77 | | | | | | | | | | | | | |
| ROL | C←7..0←C | 2E | | 3E | | | 2A | | | | | 26 | | 36 | | | | N | | | | | | Z | $M_7$ |
| ROR | C→7..0→C | 6E | | 7E | | | 6A | | | | | 66 | | 76 | | | | N | | | | | | Z | $M_0$ |
| RTI | Return from Interrupt | | | | | | | | | | 40 | | | | | | | N | V | | | D | I | Z | C |
| RTS | Return from Subroutine | | | | | | | | | | 60 | | | | | | | | | | | | | | |
| SBC | A-M-(-C) →A | ED | | FD | F9 | | | E9 | | | | E5 | E1 | F5 | | F2 | F1 | N | V | | | | | Z | C |
| SEC | 1→C | | | | | | | | 38 | | | | | | | | | | | | | | | | 1 |
| SED | 1→D | | | | | | | | F8 | | | | | | | | | | | | | 1 | | | |
| SEI | 1→I | | | | | | | | 78 | | | | | | | | | | | | | | 1 | | |
| SMB0 | Set Memory b0 | | | | | | | | | | | 87 | | | | | | | | | | | | | |
| SMB1 | Set Memory b1 | | | | | | | | | | | 97 | | | | | | | | | | | | | |
| SMB2 | Set Memory b2 | | | | | | | | | | | A7 | | | | | | | | | | | | | |
| SMB3 | Set Memory b3 | | | | | | | | | | | B7 | | | | | | | | | | | | | |
| SMB4 | Set Memory b4 | | | | | | | | | | | C7 | | | | | | | | | | | | | |
| SMB5 | Set Memory b5 | | | | | | | | | | | D7 | | | | | | | | | | | | | |
| SMB6 | Set Memory b6 | | | | | | | | | | | E7 | | | | | | | | | | | | | |
| SMB7 | Set Memory b7 | | | | | | | | | | | F7 | | | | | | | | | | | | | |
| STA | A→M | 8D | | 9D | 99 | | | | | | | 85 | 81 | 95 | | 92 | 91 | | | | | | | | |
| STP | Stop (1→PHI2) | | | | | | | | DB | | | | | | | | | | | | | | | | |
| STX | X→M | 8E | | | | | | | | | | 86 | | | 96 | | | | | | | | | | |
| STY | Y→M | 8C | | | | | | | | | | 84 | | 94 | | | | | | | | | | | |
| STZ | 00→M | 9C | | 9E | | | | | | | | 64 | | 74 | | | | | | | | | | | |
| TAX | A→X | | | | | | | | AA | | | | | | | | | N | | | | | | Z | |
| TAY | A→Y | | | | | | | | A8 | | | | | | | | | N | | | | | | Z | |
| TRB | ~A^M→M | | | | | | | | | | | 14 | | | | | | | | | | | | Z | |
| TSB | A∨M→M | | | | | | | | | | | 04 | | | | | | | | | | | | Z | |
| TSX | S→X | | | | | | | | BA | | | | | | | | | N | | | | | | Z | |
| TXA | X→A | | | | | | | | 8A | | | | | | | | | N | | | | | | Z | |
| TXS | X→S | | | | | | | | 9A | | | | | | | | | | | | | | | | |
| TYA | Y→A | | | | | | | | 98 | | | | | | | | | N | | | | | | Z | |
| WAI | 0→RDY | | | | | | | | CB | | | | | | | | | | | | | | | | |