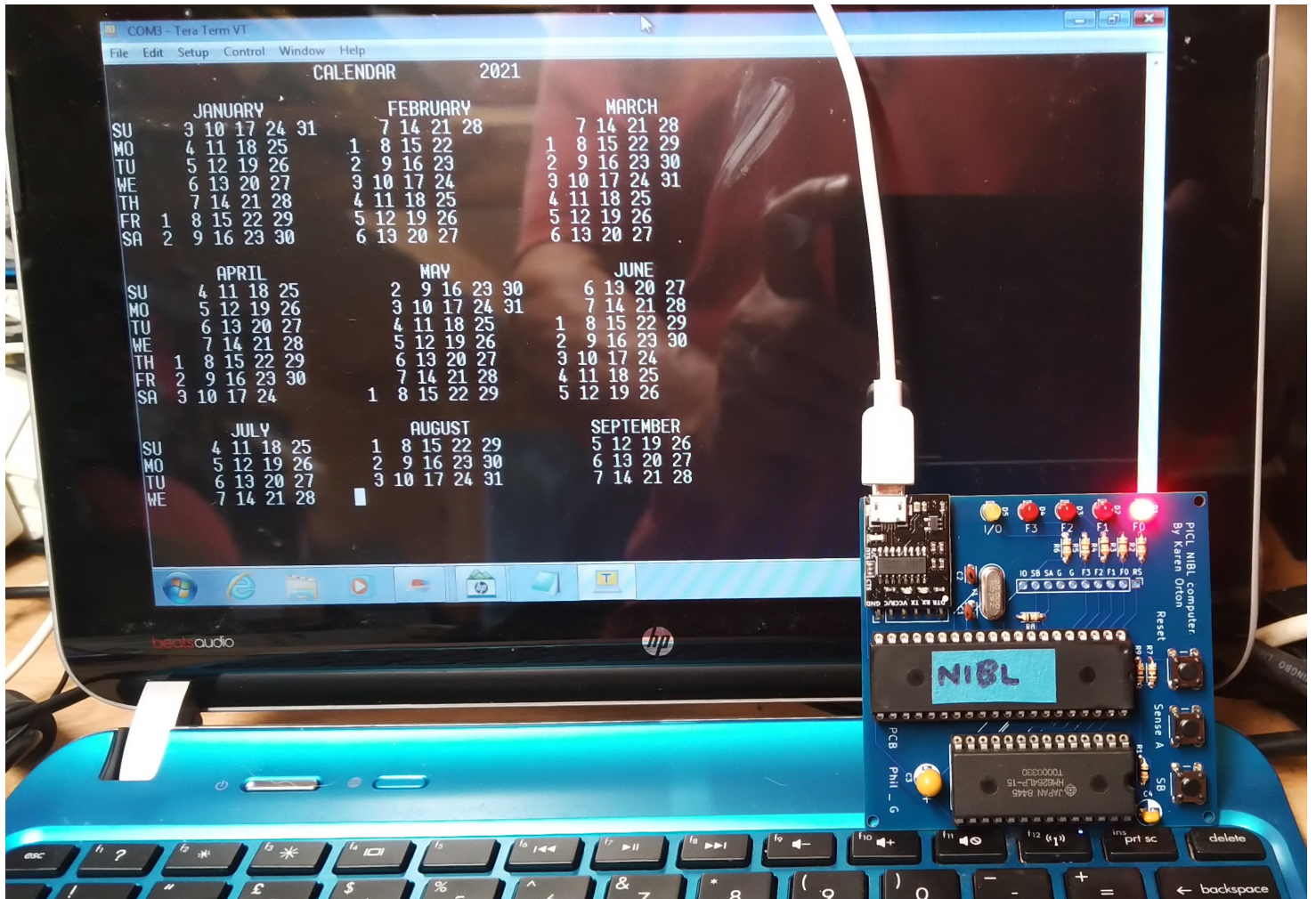


PICL National Industrial Basic Language computer by Karen Orton



The PICL is one of Karen Orton's earlier NIBL designs, published in 2007 and using her superb cycle-perfect SC/MP emulation on a Microchip PIC processor. It is probably the simplest possible NIBL machine, comprising only of the processor and a memory chip, and as such is an ideal introductory project for anyone with an interest in NIBL or the SC/MP.

Whilst Karen usually documented her projects well, information on the PICL project is sparse, hence this attempt at a brief construction and use manual :-)

In Karen's own words:

PICL A PIC Emulation of a NIBL Tiny Basic Computer

Eager to promote the SC/MP microprocessor as an industrial microcontroller, National released their 'National Industrial Basic Language' for this chip. It was quickly seized upon by hobbyists who used it in early home computers. The European publication Elektor ran a particularly popular series of articles on this and other SC/MP projects, and I believe even released programs on 45RPM records!

This project is a three chip NIBL computer [the original had a MAX232] with 8 kilobytes of data and program storage.

You talk to the computer using a simple serial link set for 1200 baud, eight data bits, no parity, one stop bit. No hardware handshaking should be selected. The Windows Hyperterminal program can be easily set up to do this, though you may need to acquire a USB-Serial bridge device if your computer is even remotely modern [this implementation replaces the MAX232 with an inbuilt USB serial module].

NIBL is a fairly typical sixteen bit integer tiny Basic but has a few nice features. I [Karen] won't provide an in-depth description of NIBL here, but here are the highlights:

Program elements: IF / THEN GOSUB / RETURN FOR / NEXT DO / UNTIL Bitwise operators: AND OR NOT Variables: A-Z (twenty-six signed sixteen bit integer variables) Special operators: @ (indirection operator – implements peek and poke) # (hexadecimal number entry) \$ (string at specified memory address) Pseudo variables: PAGE (current page) TOP (first free address after program) STAT (SC/MP status register) Special statements: LINK (call machine code subroutine)

The PAGE pseudo variable recognizes the SC/MP 4kbyte paging. The program code in each page is treated like a separate program (i.e. line numbers may begin from 1 in each). Moving between these separate program segments is achieved by assigning the PAGE pseudo variable, on which control will be transferred to the first statement on the identified page. Two pages are available in the PICL – PAGE=1 and PAGE=2.

The STAT pseudo variable represents the SC/MP status register and allows access to the flags and sense inputs. For example, the statement STAT=#0F will cause the four flag LEDs to illuminate (as with the PIC14, the IE status bit is brought out as a fourth flag output).

Finally, no speed rating is supplied for the 6264 component as you'll be hard pressed to find a part that is too slow!

Karen Orton, 2007

This PCB mostly follows Karen's published diagram:

http://techlib.com/area_50/Readers/Karen/micro.htm#PICL

(Should this site ever disappear, in order to preserve Karen's work, we do have a mirror.)

The only change is that since few modern PCs have a serial port, I've replaced the old MAX232 driver with a modern USB serial comms board.

The PIC16F877 should be programmed with the hex from Karen's page *[but see later update]*.

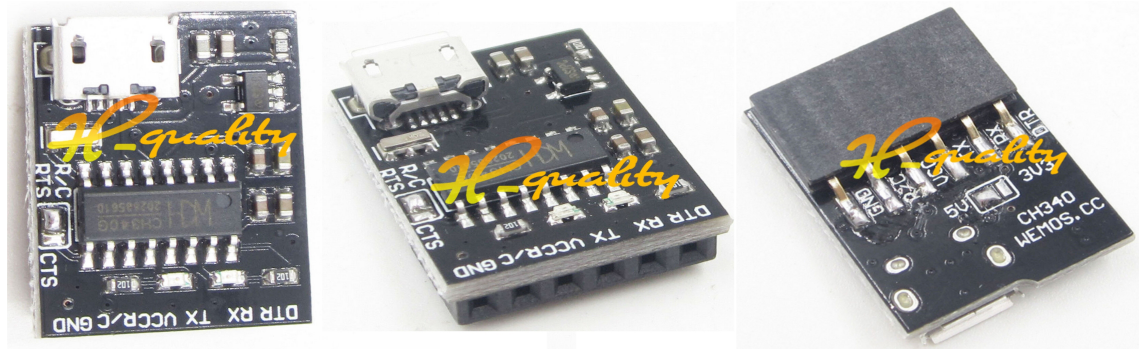
I don't know if an '877A' variant would work so I would suggest sticking to the original 877.

The RAM chip is a common 6264 which is 8k bytes and since NIBL works in 4k pages, this equates to two pages, 1 and 2. Separate or linked programs can exist in different pages but remember that all page content is lost when you power-off so remember to save your work ! *[see addendum...]*

The serial board is a CH340G about an inch square with a mini-USB and a 6-pin header socket. Its mounted component-side upwards the PCB - check that the pins correspond with the PCB header. Heres an example of the board: <https://www.ebay.co.uk/itm/142177268249>

Here's a 340G driver tutorial:

<https://learn.sparkfun.com/tutorials/how-to-install-ch340-drivers/all>



Communication

...with the board is via a Terminal Emulator such as Teraterm, the baudrate is fixed in software at 1200 and you will need local-echo turned on.

By setting a line-delay of (say) 300ms Teraterm can be used to stream programs into the PICL using 'File', 'Send'. The line-delay is necessary to give NIBL time to store each received line.

A small character-delay is beneficial too, say 5ms, this helps prevent overrun on long programs.

There's no 'SAVE' command as such but NIBL programs can be saved by using the 'Log' function on Teraterm as follows:

Type 'LIST' but don't press return yet. Enable logging, then press Return to stream the listing to a log file. Use a logfile name such as 'PROG.BAS'. Remember to turn off the log file or the remainder of your session will be appended to the listing! If you edit the log file with notepad, and append 'RUN' to the end, the program will run automatically after loading.

All NIBL statements must be upper case, lower case will simply report 'SNTX' error. PRINT statement contents in NIBL can contain upper or lower case, ie PRINT "Hello".

The I/O LED simply indicates serial activity which can also be seen on the CH340 LED.

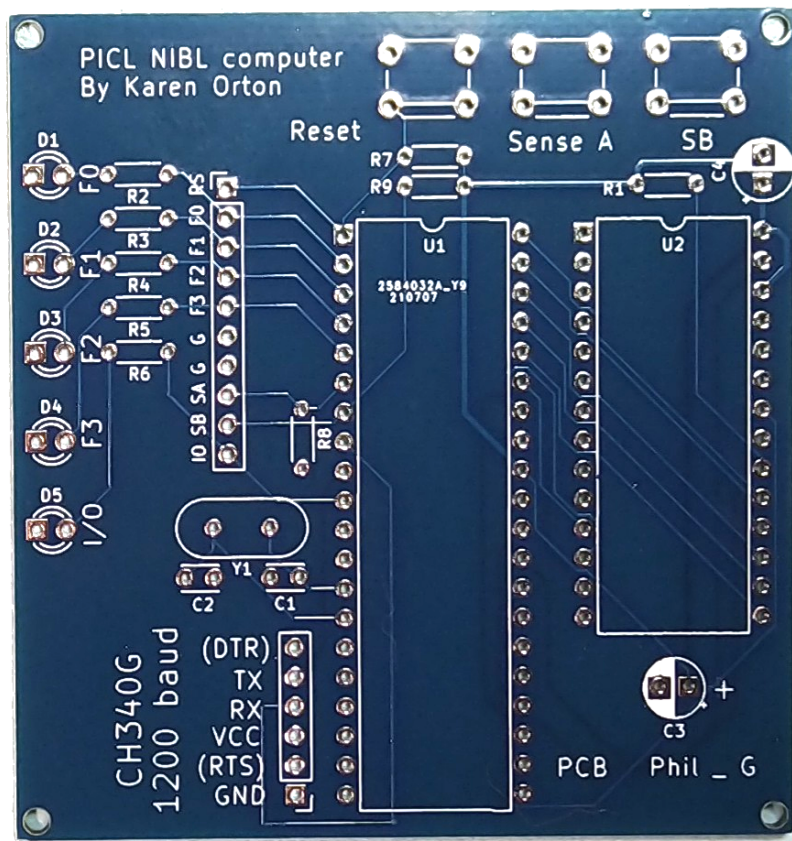
Components

All the resistors feeding the LEDs – R2, R3, R4, R5 & R6 - are 330, 470 or 560 ohm, non-critical. The remainder – R1, R7, R8 & R9 - are pull-ups and can be 4k7 or 10k. All are 1/8 watt. There are two decoupling capacitors, they're not critical and are in parallel across the 5v supply. I make one a 22 μ F tantalum for smoothing and the other a 100n for noise decoupling. Ensure the tantalum polarity is correct, reversed tantalums can catch fire !

To mount the CH340G serial USB board, I find it best to pull the 90° pins from the plastic strip and to fit them individually into the CH340 module, then fit the module into the PCB before soldering the pins, that way the module can be aligned neatly and as close as possible to the PIC to ensure theres no overhang beyond the main PCB outline.

The crystal is a common 20mhz short can with 20pf either side, these cost pennies from RS. The push-buttons are standard 6x6x5mm tactile momentary switches such as:
<https://www.ebay.co.uk/itm/304040262037>

The header plug is optional and provides flag, sense, I/O & reset connections, should you want to mount them remotely on a front panel.



Test programs:

```
10 FOR A=1 TO 15
20 FOR B=1 TO A:PRINT " ";:NEXT B
30 PRINT"PICL BY KAREN ORTON"
40 STAT=A
50 NEXT A
```

Button test:

```
10 P=STAT
20 IF P AND 32 < 32 PRINT"YOU PRESSED SENSE-B"
30 IF P AND 16 < 16 PRINT"YOU PRESSED SENSE-A"
40 GOT0 10
```

Pause for Sense button:

```
10 PRINT"PRESS SENSE-A TO CONTINUE":DO:UNTIL STAT AND 16=0
20 PRINT"THANK YOU"
```

'Kit' lights:

```
10 B=1
20 FOR A=1 TO 4
30 STAT=B
40 FOR T=1 TO 5:NEXT T
50 B=B*2
60 NEXT A
70 FOR A=1 TO 4
80 B=B/2
90 STAT=B
100 FOR T=1 TO 5:NEXT T
110 NEXT A
120 GOTO 10
```

Using NIBL

Program Entry

A Line without a program number executes immediately

A line with a number is inserted in order in the program

Line numbers must be from 0 to 32767

No blanks in keywords (LET, IF, THEN, GOTO, GOSUB, GO, TO, SUB, RETURN, INPUT, PRINT, LIST, CLEAR, RUN)

Blanks outside keywords are optional

SHIFT/O (Back-arrow) deletes last character typed

CONTROL/H (Backspace) has the same function as SHIFT/O (for use with CRT's)

CONTROL/U Deletes the entire current line

Program Control

CLEAR clears all variables (A-Z) and all stacks

NEW deletes the program in PAGE 1 (the default page)

NEW n (n is from 2-7) deletes the program in page n

LIST lists the program starting at the lowest line number or the line number given

RUN starts the program at the lowest line number

NOTE

Repeated pressing of the BREAK key interrupts execution and returns the system to programming mode.

Expressions

All expressions are 16-bit, twos-complement values

26 variable names: A through Z.

Relational Operators: <, >, =, <=, >=, <>

Arithmetic operators: +, -, *, /

Logical Operators: AND, OR, NOT

Decimal Constants in the range -32767 to 32767

Hexadecimal Constants denoted by '#' followed by Hex digits

Expressions can be on individual lines or several can be inserted on the same line if they are separated with a colon (e.g., 100 PRINT "HOW MANY";:INPUT X)

Functions

RND(a,b) returns the random number in the range a through b

MOD(a,b) returns the remainder of a/b

STAT returns the value of the INS8060 Status Register

PAGE returns the number of the current page

TOP returns the highest address of the NIBL program in the current page

Input/Output Statements

INPUT X

INPUT X,Y,Z

PRINT "A STRING"

PRINT "F=",M*A

PRINT "THERE ARE",X,"DAYS TO CHRISTMAS";

NOTE

The semicolon suppresses an otherwise automatic carriage return after any PRINT statement.

String Operations using '\$'

NIBL has primitive but useful string handling. Memory must be manually allocated for strings, using TOP to find free memory. Strings can be copied and individual characters can be accessed or modified using the '@' indirection operator. To fully understand string handling, its best to study the NIBL manual.

Assignment Statements

```
LET X=7
E=I*R
STAT=#70
PAGE=PAGE+1
LET @A=255
@(T+36)=#FF
B=@(TOP+5)
```

Control Statements

```
GO TO 15 or GOTO 15
GOTO X+5
GO SUB 100 or GOSUB 100
RETURN
IF X+Y>=#1A GOTO 15
IF A=B LET A=B-C
FOR I=10 to 0 STEP -2
NEXT I
FOR K=1 TO 5
DO: X=X+1: UNTIL (X=10)OR(@X=13)
```

Indirect Operator

If the value of V is #2000, then "LET @V=100" stores 100(dec) at memory location 2000(hex) and "LET W=@V" sets W to the contents of the location specified by V. Values 00-FFh, 0-255 decimal.

Additional Statements

LINK "address" causes control to be transferred to the SC/MP Machine Language Routine starting at "address" which is a decimal number
REM - This statement is used to insert comments into the NIBL program
END - this statement is useful for inserting breakpoints into the NIBL program while it is being debugged. When NIBL encounters an END statement, it prints a break message and the current line number, and returns to edit mode

Message Description

CHAR Character after logical end of statement
DIV0 Division by zero
END" No ending quote on string
FOR For without next
NEST Nesting limit exceeded in expression, FORs, GOSUBs, etc
NEXT NEXT without FOR
NOGO No line number corresponding to GOTO or GOSUB
RTRN RETURN without GOSUB
SNTX Syntax error
STMT Statement type used improperly
UNTL UNTIL without DO
VALU Constant format or value error
AREA No more room left in current page for program

All NIBL statements must be upper case, lower case will simply report 'SNTX' error.
PRINT statements and '\$' strings in NIBL can use upper and/or lower case.

Have fun with your PICL, Karen designed it as a fun project. I think she would have liked this PCB :-)

Cheers

Phil_G

[Revised 23/7/2021 for 2nd batch of PCBs]

Addendum: 10/08/2021

I wrote a simple monitor in NIBL, it has block copy, hex memory dump, modify memory, execute a machine-code program, convert hex-decimal and decimal-hex, memory fill, and a free memory display. I called it BASYS :-)

It looks messy and its very slow, but string handing is minimal in NIBL so theres a lot of jiggery pokery going on. Theres no syntax checking so if you do anything wrong and it errors out, just type run again. Any hex input (addresses, data) is prefixed '#'.

The hex conversion method only works up to 32767 (at which point it goes negative) which I can live with as the PICL has only 8k of memory. For clarity I left all the REMs in and didnt abbreviate any commands, so it can be shortened a lot once you've had a read through:

```
40 REM BASYS MONITOR FOR NIBL - PHILG
50 REM USE ABBREVIATIONS AND DELETE REMS
60 A=TOP
70 B=TOP+20
80 $B="Z"
90 $A="0123456789ABCDEF"
100 PRINT"":PRINT"BASYS COMMANDS:"
120 PRINT"-----"
140 PRINT"1=copy"
150 PRINT"2=display"
160 PRINT"3=execute program"
170 PRINT"4=convert Hex > Dec"
180 PRINT"5=convert Dec > Hex"
190 PRINT"6=modify"
200 PRINT"7=fill mem"
210 PRINT"8=free memory start"
220 PRINT ""
230 PRINT "COMMAND ";:INPUT C
300 IF C=1 GOTO 1000
310 IF C=2 GOTO 2000
320 IF C=3 GOTO 3000
330 IF C=4 GOTO 3500
340 IF C=5 GOTO 5500
350 IF C=6 GOTO 5000
360 IF C=7 GOTO 8000
370 IF C=8 GOTO 6000
400 GOTO 100
999 REM*****
1000 PRINT"From: prefix hex with # ":INPUT H
1020 PRINT"To: prefix hex with # ":INPUT G
1050 PRINT"Length: prefix hex with # ":INPUT F
1070 FOR Q=1 TO F
1080 @G=@H
1090 H=H+1:G=G+1
1100 NEXT Q
1110 PRINT"COPIED":GOTO 100
1999 REM*****
2000 PRINT"From: prefix hex with # ":INPUT H
2015 PRINT"HEX DUMP:"
2016 PRINT"ADD- 0/ 1/ 2/ 3/ 4/ 5/ 6/ 7/"
2018 PRINT"RESS /8 /9 /A /B /C /D /E /F ASCII"
2019 PRINT ""
2020 FOR D=0 TO 7
2030 X=H/256:GOSUB 20000:X=H-(256*X):GOSUB 20000:PRINT" ";
2045 FOR E=0 TO 7:X=@(H)
2050 GOSUB 20000:PRINT" ";:H=H+1:NEXT E:PRINT" ";
2053 H=H-8
2054 FOR E=0 TO 7:@(TOP+20)=#2E
2055 IF @(H)<32 GOTO 2059
2056 IF @(H)>126 GOTO 2059
2057 @(TOP+20)=@(H)
2059 PRINT $B;:H=H+1:NEXT E
2060 PRINT "":NEXT D
2080 GOTO 100
```

```

2999 REM*****
3000 PRINT"Execute address: ";:INPUT H
3050 LINK(H):REM JUMP TO M/C PROGRAM
3499 REM*****
3500 PRINT"Hex: prefix with # ";:INPUT H
3510 PRINT"Dec = ";:PRINT H
3590 GOTO 100
4999 REM*****
5000 PRINT"From: prefix hex with #, N for next, Q to quit: ";:INPUT H
5010 N=256:Q=257
5020 X=H/256:GOSUB 20000:X=H-(256*X):GOSUB 20000:PRINT" ";
5030 X=@(H):GOSUB 20000:PRINT" ";
5040 PRINT"Hex: prefix with # ";:INPUT E
5045 IF E=257 GOTO 100
5050 IF E<0 H=H+E:GOTO 5020
5060 IF E=256 H=H+1:GOTO 5020
5062 @H=E:H=H+1
5070 GOTO 5020
5499 REM*****
5500 PRINT"Decimal: ";:INPUT K
5510 PRINT"Hex = ";
5530 X=K/256:GOSUB 20000:X=K-256*X:GOSUB 20000
5540 PRINT"":GOTO 100
5999 REM*****
6000 PRINT"Dec ";:PRINT TOP+24:H=TOP+24
6010 PRINT"Hex ";
6020 X=H/256:GOSUB 20000:X=H-(256*X):GOSUB 20000
6030 PRINT""
6040 GOTO 100
8000 PRINT"From: prefix hex with # ";:INPUT H
8030 PRINT"To: prefix hex with # ";:INPUT L
8050 PRINT"With byte: prefix hex with # ";:INPUT K
8070 FOR M=H TO L:@M=K:PRINT". ";:NEXT M:PRINT"Filled"
8080 GOTO 100
19998 REM*****
19999 REM PRINT DEC VARIABLE 'X' IN HEX
20000 U=X/16
20010 @(TOP+20)=@(A+U)
20020 PRINT $B;
20040 U=X-(U*16)
20050 @(TOP+20)=@(A+U)
20060 PRINT $B;
20100 RETURN

```

BASYS can be downloaded from the PICL section of <http://philg.uk/>

A software update

Karens PICL used an early implementation of her superb SC/MP emulation on the PIC16F877. One of the differences between the emulation used in the PICL and her later machines is that the PICL didnt fully implement the SC/MP DLY instruction.

This was most probably because this was to be a dedicated NIBL machine, and DLY would only have been used for the serial I/O routines, which she'd extracted and given pseudo ops to PIC serial code. DLY would therefore never be used and perhaps, whilst under development, the memory space was more valuable than an unused routine.

This came to light when calling (LINK) machine-code routines from NIBL which would lock-up if they included any form of DLY - though these were taking the PICL outside its intended envelope, within which it works absolutely perfectly.

This, for example, should waggle the flag lines up and down continuously, but in practise, flips them once then locks up:

```
; Waggle flag lines continuously
;
LOOP: 06          CAS          ; GET STATUS
      E4 07      XRI 7        ; FLIP FLAG BITS
      07          CSA          ; REPLACE STATUS
      C4 FF      LDI FF        ; MAX COUNT IN ACC
      8F FF      DLY FF        ; MAX DELAY
      90 F6      JMP LOOP
```

What I've done is to take Karens own DLY code from her later emulation, and patched it into PICL. The updated PICL code therefore remains entirely Karens own work and her name alone appears in the source credit. With the update, NIBL operation is unaffected but DLY now works as intended, and machine-code programs using DLY now run properly and as far as I can tell, with the correct delays. The update is in the PICL section of <http://philg.uk/>

I must emphasise that if the PICL is used only for NIBL basic programming, then the update is of no extra value, it only comes into use with m/c routines since the PICL version of NIBL doesnt use DLY at all. Hence it is not in any way a bug and Karens original code is flawless for its intended purpose.

I do hope everyone approves of this change, and I sincerely hope it doesnt appear disrespectful.

I've been using BASYS to play with machine code stuff on the PICL and of course I get the occasional lock-up where the only option is a hard reset.

This clears the current NIBL program and so BASYS has to be reloaded. Or does it ?

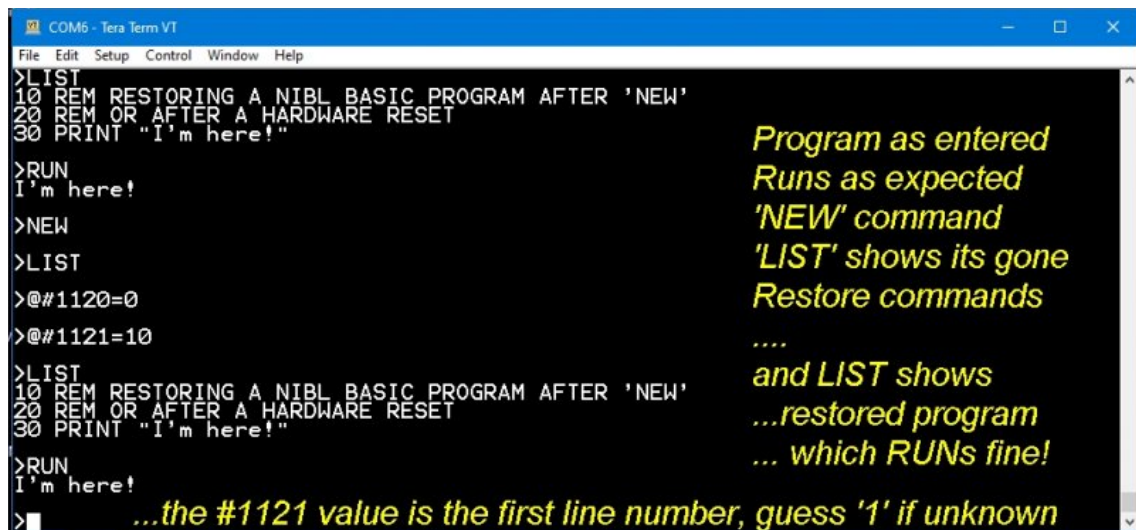
It transpires that both 'NEW' and hardware-reset only change the first couple of bytes of the program, the rest of the program remains intact in memory.

These two magic bytes hold the line number of the very first line of NIBL, which in the example shown is 10, hence the two bytes are '00' and '10' decimal. To restore a program, against the > prompt, enter these two lines (ie in direct mode):

```
@#1120=0
@#1121=10
```

If you dont know the first line number, you can enter '00' and '01' to set it to line 1, then do another @#1121=nn later when you realise what it should be.

Its a 'word' value so if your first line number is greater than 255 set #1120 to the MSB, for example if the first line is 1000 its 3 and 232 (#3E8)



```
COM6 - Tera Term VT
File Edit Setup Control Window Help
>LIST
10 REM RESTORING A NIBL BASIC PROGRAM AFTER 'NEW'
20 REM OR AFTER A HARDWARE RESET
30 PRINT "I'm here!"

>RUN
I'm here!

>NEW

>LIST

>@#1120=0

>@#1121=10

>LIST
10 REM RESTORING A NIBL BASIC PROGRAM AFTER 'NEW'
20 REM OR AFTER A HARDWARE RESET
30 PRINT "I'm here!"

>RUN
I'm here!

>
```

*Program as entered
Runs as expected
'NEW' command
'LIST' shows its gone
Restore commands
....
and LIST shows
...restored program
... which RUNs fine!
...the #1121 value is the first line number, guess '1' if unknown*

I've not seen this NIBL trick documented anywhere, it reminds me of the 'OLD' command on NASCOM basic to restore a program after a 'NEW'

NIBL programs in memory

NIBL programs occupy memory as follows:

PAGE 1

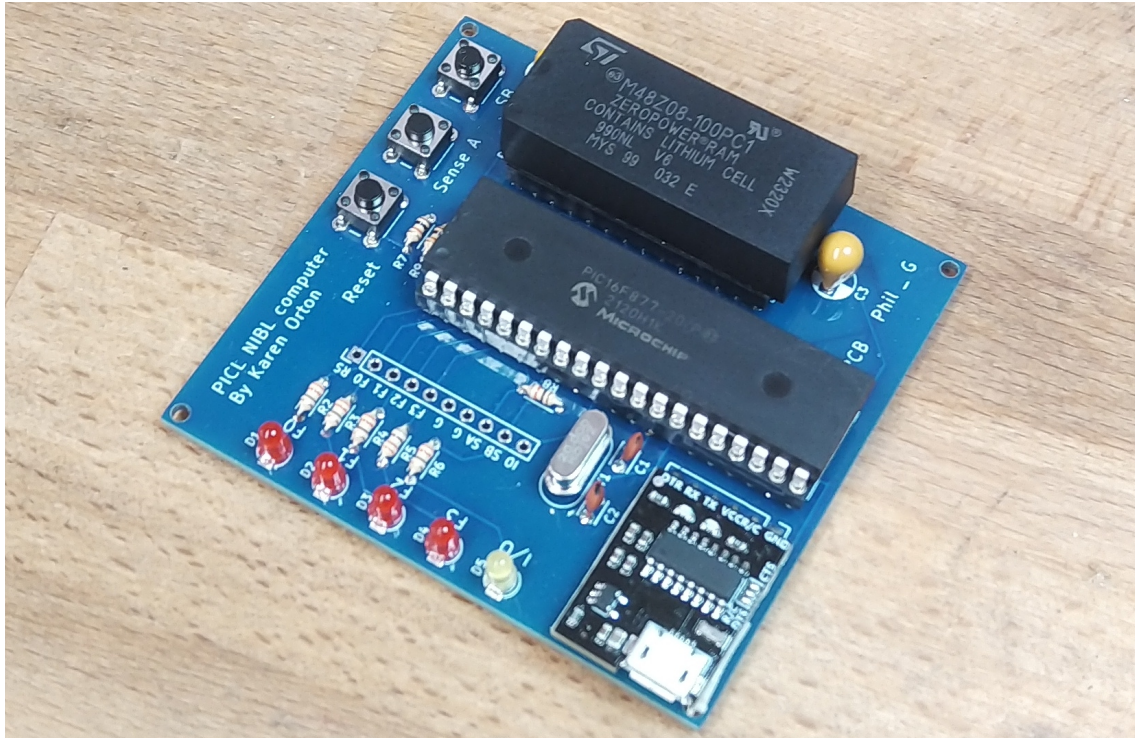
0x111E	0x00	; these first two bytes are a dummy line
0x111F	0x0D	; with a 'return'. Their content doesnt seem critical.
0x1120	0x00	; MSB of the first line number
0x1121	0x0A	; LSB of the first line number, line 10 in this example
0x1122	byte	; this is the byte-count of the entire line including its number and the CR
0x1123	ASCII 0x20	; ... this is the start of the line text, ie the space before the 'P' of print
0x1124	ASCII 'P'	; in this example, followed by the rest of the first line...
0x1124	ASCII 'R'	; etc for the rest of the first line, terminating in CR

```
10 PRINT"HELLO, WORLD"<cr>
```

PAGE 2 is similar, starting at address 0x2000 rather than 0x111E, with the first line number at 0x2002

Further to this 'restore' discovery, I've now swapped the PICL 6264 static ram with an M48Z08 NVRAM which is a direct replacement having the same 8k x 8 format and identical pinout. With the NVRAM, programs are now retained indefinitely through long power-off periods, which is handy. On powering up, NIBL having no 'warm-start' as such, the two direct commands @#1120=0 and @#1121=10 are typed, restoring everything exactly as it was before power down. The M48Z08 was £15 from RS, not too bad considering present day silicon prices: <https://uk.rs-online.com/web/p/nvram/1892428/>

Heres the PICL board with its NVRAM module:



Auto-run a PAGE 2 ROM program

I've yet to experiment with the "auto-run a page 2 ROM program" feature of NIBL, that would turn the PICL into a useful stand-alone controller if sense inputs and flag outputs are sufficient for an application. I've not achieved this yet but here are some preliminary thoughts:

On the PICL, A12 splits the 8k SRAM into two pages, PAGE1 and PAGE2, so I think /WR can be gated with A12 in order to make PAGE2 read-only – my preference would be to have the PIC code do the same thing in software by checking A12 in the RAM write routine (macro). My intention is to use two special characters from the terminal to set PAGE2 to read-write or read-only. The current status will be saved in EEPROM, thus a PAGE2 program can be entered, then made read-only and should auto-execute on the next power-up.

I suspect its a once only test on startup, if subsequently PAGE2 became read-write, I think everything would still work. This also would have the advantage of allowing self-modifying code in PAGE2 'ROM'

Cheers
Phil_G