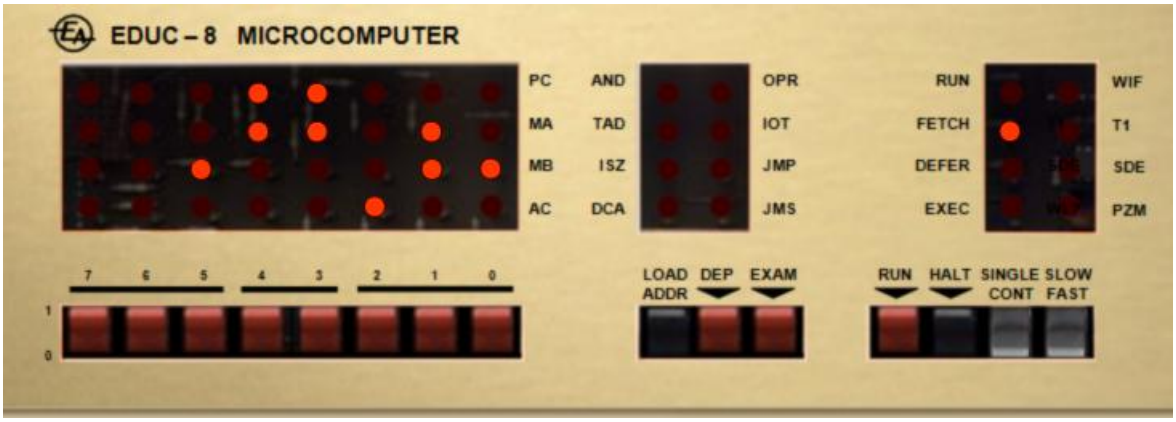
EDUC-8

EDUCATIONAL MICROCOMPUTER EMULATOR



Project Design by Tony Nixon tnixon059@gmail.com
http://www.teenix.org

I'd like to add my appreciation to Steven Pietrobon for all of his assistance in helping me to get this project working as close to the original EDUC-8 as I could.

Visit Stevens web blog on building a real EDUC-8.

http://www.sworld.com.au/steven/educ-8/

Contents

Introduction

The EDUC-8 is an 8-bit microcomputer that was designed by Jim Rowe and the project construction details were published in Electronics Australia magazine between August 1974 and January 1975. It did not have a microprocessor as such as it was constructed with 100 TTL Integrated Circuits that populated 8 circuit boards It looked like it was going to be the first digital computer to be presented for home construction, but unfortunately it was pipped at the post by the “Mark-8” project based on the Intel 8008 microprocessor IC.

To keep the circuit boards simpler, the data paths were serial not parallel and this of course slowed the processing down as it took longer to shift the information between the various parts of the circuit.

Processing instructions is based around cycles, each of which requires 24 master clock pulses to complete. An instruction may take either 2 or 3 of these cycles depending on the mode of addressing used.

Extra peripherals, such as a keypad, LED display and a punched paper reader were described in future articles.

To better understand the help file and the computer operation, you could take a look at the original EDUC-8 Articles. These were published in the Electronics Australia magazine and are available here, unless the link is broken, in which case contact the author of this project.

https://archive.org/stream/Educ-8/educ8#page/n11/mode/2up

Basic Specifications

Original Enclosure size	11.5 x 4 x 14 inches
IC Count	100 TTL
Microprocessor	None
Circuit Boards	8 single sided
Memory Bytes	32, expanded to 256 at publication
Input Channels	1 Serial, 1 Serial/Parallel
Output Channels	2 Serial
Data Entry	Front Panel Switches
Instruction Execution	Normal 10KHz, or Slow 24 seconds, @ 500KHz main clock
Instruction Set	28 Instructions
Memory page Size	16 Bytes
Addressing Modes	Direct and Indirect
Subroutine Capability	Yes, only available memory is the limit
Display	Front panel LEDs
Power Supply	5 volt, 60Watt

Note:
The PC emulation cannot keep up with the fast original serial clock rate required for the front panel LEDs without slowing the code execution, so the emulator display is refreshed at a slower rate. The hardware project has multiplexed LEDs and is not as described in the original project and so refresh differently. These issues may give the user the impression that the LEDs are not working properly. For example, a LED may be flashing on and off very fast but the refresh rate nearly always coincides when the LED being off and therefore the LED may not appear to do anything, or just briefly flicker. Operating in slow or single step mode will verify the LED operation.

EDUC-8 Architecture

The EDUC-8 is an 8 bit machine with 26 instructions

Each instruction incorporates the operation code, the memory address and the type of addressing mode. It can also include the input/output type and device number.

There are two addressing modes available

- Direct
- Indirect

There are four internal registers for processing.

Accumulator **AC**
Used for data computations and transfer

Program Counter **PC**
Holds the address for the memory access

Memory Buffer **MB**
Acts as a holding register for the memory

Memory Address **MA**
Holds the address for accessing memory

Instruction Processing

An instruction using Direct Addressing requires two cycles to complete.

1. **Fetch**
   Fetch an instruction from memory and decode it
2. **Execute**
   Process the instruction

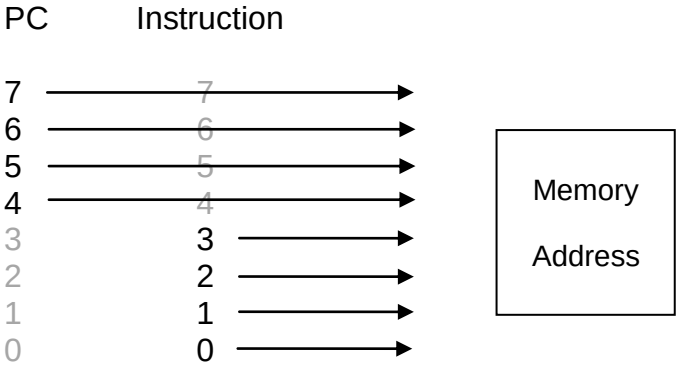
An instruction using Indirect Addressing requires three cycles to complete.

1. **Fetch**
   Fetch an instruction from memory and decode it
2. **Defer**
   Additional processing required for indirect address computation
3. **Execute**
   Process the instruction

Addressing Modes

Direct — Address values encoded in the instruction are 4 bits wide and can only access blocks of 16 memory addresses, called pages. With 256 bytes of memory available, there are 16 pages. Instructions can only access memory within the page that contains the instruction.

Memory Address = (PC AND $F0) + (Instruction AND $0F)

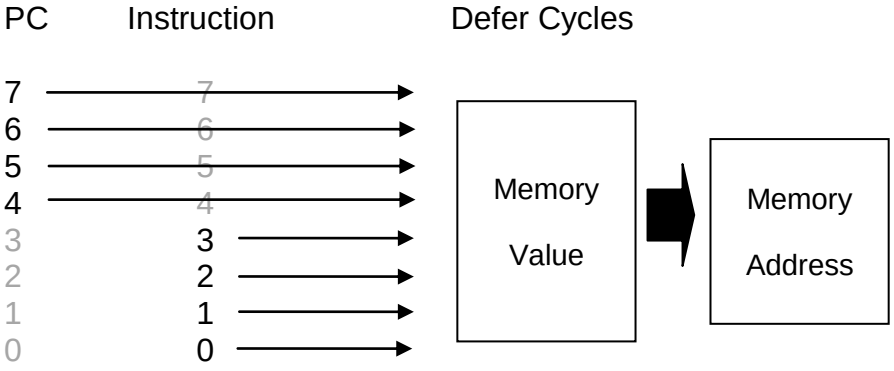


Indirect — Address values encoded into the instruction are 4 bits wide and can only access memory within the page that contains the instruction. However, in this case the contents of the memory location that the address refers to is used to provide an 8 bit indirect address. This then can be used to access any address in the 256 byte memory.

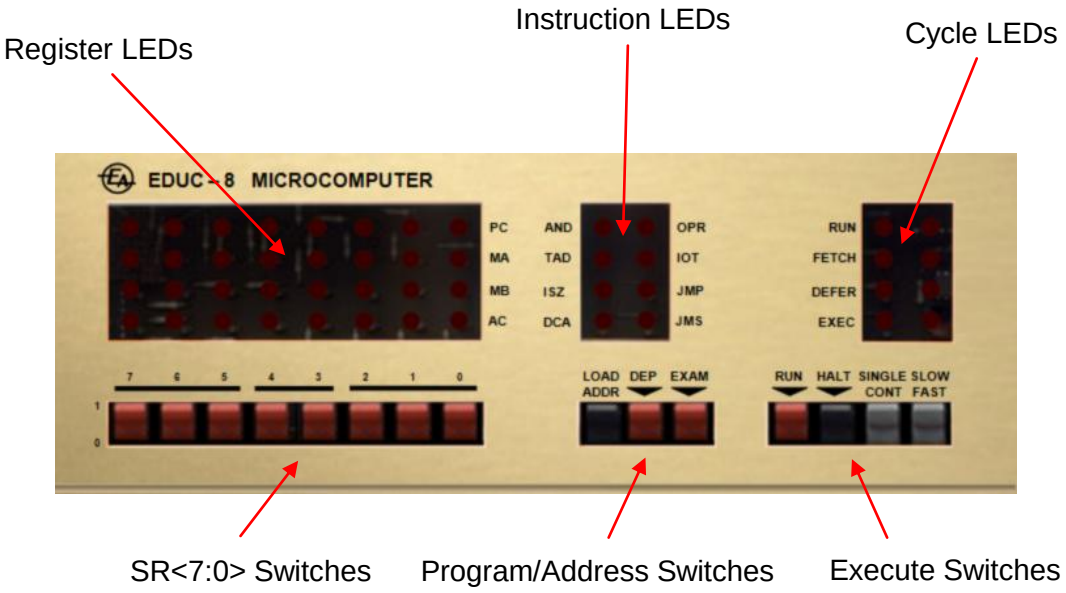
Memory Address = (PC AND $F0) + (Instruction AND $0F)

Indirect Address = Contents of memory at that address

The Defer cycle is used to set up the indirect memory address.



Front Panel



Register LEDs	These display the 8 bit status of the PC, MA, MB and AC registers.
Instruction LEDs	These display the types of instruction being executed.
Cycle LEDs	These display the cycle phase of the current instruction.
Program/Address Switches	These control how the user inputs or reads information.
Execute Switches	These control program execution
SR<7:0> Switches	These set an 8 bit binary value. Logic 1 is up, Logic 0 is down.
LOAD ADDR	This writes the memory address as set by the 8 bit switches and appears in the MA register.
DEP	This deposits the value as set by the 8 bit switches into the current memory address, then the PC is incremented. MA shows the address of the byte just written. ie. Enter a program or data byte
EXAM	This examines the value in memory from an address set by the 8 bit switches. It is displayed in the MB register, then the PC is incremented. MA shows the address of the byte just read. ie. Read a program or data byte

RUN — This causes a program to begin execution, starting at the current memory address

HALT — This stops a running program after the end of the current EXEC cycle. For slow execution mode, the Halt key must be held down until the last cycle of the current instruction before it will be recognised.

SINGLE/CONT — If set to SINGLE, only 1 instruction will execute if RUN is pressed.

If set to CONT, a program will run continuously or until a HALT instruction is executed, or the HALT key is pressed.

SLOW/FAST — If set to SLOW, the main clock cycle rate will be 2Hz and each instruction cycle will take 12 seconds to complete. Therefore normal instructions will take 24 seconds to complete and defer instructions will take 36 seconds to complete

If set to FAST, the main clock cycle rate is 500KHz. Normal instructions execute at about 10KHz and Defer instructions execute at about 6.7KHz.

Switch Operation

For each switch, place mouse in this area and left click for the down position

For each switch, place mouse in this area and left click for the up position

Switches for LOAD ADDR, DEP, EXAM, RUN and HALT will automatically return to the up position when the mouse button is released.

SR<7:0> Input Switches

These switches enter data as binary numbers, but are arranged in a special EDUC-8 OCTAL format as shown in the lines drawn above the switches.

The first 3 switches correspond to the instruction bits [7,6,5], so the OCTAL format is set as 7.3.7, not 3.7.7 as would normally be the case for an 8 bit number. The 7.3.7 format is also the original instruction code format used for the EDUC-8 and helps understand the instruction decoding.

Input / Output

The EDUC-8 has 4 serial I/O ports, which includes 2 inputs and 2 outputs. Input Port Dev0 also has access to the Deposit control and the Load Address control, as well as a parallel connection to the SR<7:0> switches.

Input Port Connections

Clock	Clocks the data on the low going signal
Data In	The data bit being received into the computer
Flag Reset	A low going pulse to reset the external device
Flag	A signal from the external module to inform the computer that the external device is ready (LOW)

Output Port Connections

Clock	Clocks the data on the low going signal
Data Out	The data bit being transmitted from the computer
Flag Reset	A low going pulse to reset the external device
Flag	A signal from the external module to inform the computer that the external device is ready (LOW)

Parallel Connections (Part of Input Device 0)

D7 – D0	Parallel connections to the switches
DEP	Parallel connection to the DEPOSIT switch
LA	Parallel connection to the LOAD ADDR switch

Some of the extra devices that were described in the articles were:

- OCTAL 3 Digit LED display
- Keypad with 16 keys, providing 30 functions
- Paper Tape read/write
- Music Player
- Burroughs self scan display panel

Instruction Format

The EDUC-8 had 26 8 bit instructions and are grouped into 3 separate categories.

Memory Reference Instructions (MR)

These instructions always reference a memory address

Bits	7 6 5	4	3 2 1 0
Opcodes		Indirect	Memory
000 to 101		Address	Address
		Control	16 available
Bit 4	0	Direct addressing mode	
	1	Indirect Addressing mode	

Input / Output Instructions (IOT)

These instructions control the input / output data flow

Bits	7 6 5	4	3	2 1 0
Opcode		Input	Device	Operation Type
110		Output	Select	3 available
		Select		
Bit 4	0	Instruction refers to an Input Port		
	1	Instruction refers to an Output Port		
Bit 3	0	Instruction refers to Port 0		
	1	Instruction refers to Port 1		

Operational Instructions (OPR)

These are generally for the accumulator

Bits	7 6 5	4	3 2 1 0
Opcode		Instruction	Instruction Type
111		Group	4 available for each group
		Select	
Bit 4	0	Bits 3-0 decode instruction Group 0	
	1	Bits 3-0 decode instruction Group 1	

Memory Reference Instructions

AND 000 IND ADDR

Logical AND

```
AC = AC [AND] Memory[ADDR]
```

TAD 001 IND ADDR

Two's Compliment Addition

```
AC = AC [+] Memory[ADDR]
```

Bit 7 is the sign bit 0 = Positive
1 = Negative

ISZ 010 IND ADDR

Increment and Skip if Zero

```
Memory[ADDR] = Memory[ADDR] + 1
```

If Memory[ADDR] = 0 then PC = PC + 1

(The PC will be incremented twice if this instruction executes as TRUE)

INC 010 IND ADDR

Increment

This instruction does exactly the same as the **ISZ** instruction. It can be used when you only want to increment a register and know that it will not wrap around to zero.

DCA 011 IND ADDR

Deposit and Clear Accumulator

```
Memory[ADDR] = AC
```

If Memory[ADDR] = 0 then PC = PC + 1

JMS 100 IND ADDR

Jump To Subroutine Also see JMP

After this instruction is decoded, the PC is incremented by 1

The PC value is then stored at the address specified by the instruction [ADDR]

```
PC = (PC and $F0) + [4 bit ADDR]
```

This is used as the return address for the subroutine

The PC is then incremented again and code continues for the subroutine

To return to the main code, the end of the subroutine should be an indirect jump to the memory location of the return address.

Example

PC	Instruction	
4	JMS 10	
5	main code continues after subroutine returns	
10	The computer stores 5 in this location	
11	subroutine code	
. . .	. . .	
15	JMP 10 I	An indirect jump to memory address 10

The [5] stored there becomes the new PC value

The main code continues from address 5

JMP 101 IND ADDR

Jump To Memory Address

```
PC = (PC and $F0) + [4 bit ADDR]
```

As with the JMS instruction, for a DIRECT instruction, the computed memory address will only be in the 16 byte block where the instruction resides.

Example

PC = $02 JMP can only be between $00 and $0F

PC = $24 JMP can only be between $20 and $2F

If the JMP or JMS is an INDIRECT instruction, the computed memory address will be the 8 bit value stored in the address from the instruction. Therefore, the PC can be set to anywhere within the 256 byte memory range.

Operational Instructions (OPR)

GROUP 0 (Bit 4 = 0)

IAC 111 0 0001

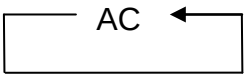
Increment Accumulator

```
AC = AC + 1
```

RAL 111 0 0010

Rotate Left Accumulator

Shift all bits to the left, bit 7 becomes bit 0.



If Bit 7 = 0, then after RAL, the value in AC will double, as long as the initial value was less than 128.

CMA 111 0 0100

Complement Accumulator

All 1 bits become 0, all 0 bits become 1

```
AC = AC XOR 255
```

CLA 111 0 1000

Clear Accumulator

```
AC = 0
```

The EDUC 8 computer can execute all of the instructions of the same Group. It will execute each of the instructions starting from bit 3 down to bit 0. Therefore you can have combined instructions that will save memory space. Some bit combinations will not make sense so only the following are implemented.

CLA,IAC 111 0 1001

Clear Accumulator, then Increment accumulator

```
AC = 1
```

CLA,CMA 111 0 1100

Clear Accumulator, then Complement Accumulator

```
AC = 255
```

CMA,IAC 111 0 0101

Complement Accumulator, then Increment Accumulator

```
AC = negative AC
```
(Bit 7 is the sign)

Operational Instructions (OPR)

GROUP 1 (Bit 4 = 1)

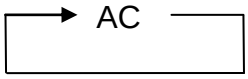
HLT 111 1 0001

Halt program execution

RAR 111 1 0010

Rotate Right Accumulator

`Shift all bits to the left, bit 0 becomes bit 7.`



If Bit 0 = 0, then after RAL, the value in AC will halve.

SMA 111 1 0100

Skip on Minus Accumulator

The PC is incremented during the FETCH cycle, and then the EXEC cycle begins.

`If AC Bit 7 = 1, then PC = PC +1`

(ie. The PC will be incremented twice if this instruction executes as TRUE)

SZA 111 1 1000

Skip on Zero Accumulator

The PC is incremented during the FETCH cycle, and then the EXEC cycle begins.

```
If AC = 0 then PC = PC +1
```

(The PC will be incremented twice if this instruction executes as TRUE)

The EDUC 8 computer can execute all of the instructions of the same Group. It will execute each of the instructions starting from bit 3 down to bit 0. Therefore you can have combined instructions that will save memory space. Some bit combinations will not make sense so only the following are implemented.

SZA,SMA 111 0 1100

Skip if Accumulator is Zero or Minus

```
If (AC = 0) OR (AC bit 7 = 1) then PC = PC +1
```

(ie. The PC will be incremented twice if this instruction executes as TRUE)

Input / Output Transfer Instructions (IOT)

INPUT DEVICES

SKF 110 0 Dev 001 (Dev = 0, or Dev = 1)

Skip on Input Device Flag

```
If the Input (Device) flag is set (LOW) then PC = PC + 1
```

(ie. The PC will be incremented twice if this instruction executes as TRUE)

KRS 110 0 Dev 010 (Dev = 0, or Dev = 1)

Read Input Data

8 clock cycles will be presented to the Input Device. The 8 bits of data from the device will be read into the Accumulator on the falling edges of these clock cycles

---

RKF 110 0 Dev 100 (Dev = 0, or Dev = 1)

Reset Input Flag

A brief LOW pulse will be generated on the Reset Flag line for the Input Device.

KRB 110 0 Dev 110 (Dev = 0, or Dev = 1)

(Combination of KRS and RKF)

Read Input Data

Reset Input Flag

8 clock cycles will be presented to the Input Device. The 8 bits of data from the device will be read into the Accumulator on the falling edges of these clock cycles

A brief LOW pulse will then be generated on the Reset Flag line for the Input Device.

Input / Output Transfer Instructions (IOT)

OUTPUT DEVICES

SDF 110 1 Dev 001 (Dev = 0, or Dev = 1)

Skip on Output Device Flag

```
If the Output (Device) flag is set (LOW) then PC = PC + 1
```

(The PC will be incremented twice if this instruction executes as TRUE)

LDS 110 1 Dev 010 (Dev = 0, or Dev = 1)

Send Output Data

8 clock cycles will be presented to the Output Device. The 8 bits of data from the AC will be sent to the Output Device on the falling edges of these clock cycles.

Also clears AC to 0.

RDF 110 1 Dev 100 (Dev = 0, or Dev = 1)

Reset Output Flag

A brief LOW pulse will be generated on the Reset Flag line for the Output Device.

LDB 110 1 Dev 110 (Dev = 0, or Dev = 1)

(Combination of LDS and RDF)

Send Output Data

Reset Output Flag

8 clock cycles will be presented to the Output Device. The 8 bits of data from the AC will be sent to the Output Device on the falling edges of these clock cycles.

A brief LOW pulse will be generated on the Reset Flag line for the Output Device.

NOTE: A link on the EDUC-8 Input / Output PCB could be set to change the order of the reset flag and the data output.

Reset Output Flag at T1

Send Output Data

or

Send Output Data

Reset Output Flag at T13

Instruction Summary

MEMORY REFERENCE INSTRUCTIONS - (XX = operand address and mode)

Mnemonic	Operation	OCTAL Code
AND	Logical AND	OXX
TAD	2 's complement add	1XX
ISZ	Increment and skip if zero	2XX
DCA	Deposit and clear AC	3XX
JMS	Jump to subroutine	4XX
JMP	Jump	5XX

OPERATE (OPR) MICROINSTRUCTIONS

Mnemonic	Operation	OCTAL Code
NOP	No operation	700
IAC	Increment AC	701
RAL	Rotate AC one bit left	702
CMA	Complement AC	704
CLA	Clear AC	710
NOP	No operation	720
HLT	Halt at end of execute cycle	721
RAR	Rotate AC one bit right	722
SMA	Skip on minus AC	724
SZA	Skip on zero AC	730

COMBINED OPR MICROINSTRUCTIONS

Mnemonic	Operation		OCTAL Code
CLA. IAC	Set AC to contain 0		171
CLA.CMA	Set AC to contain1		714
SZA.SMA	Skip if AC is zero or minus		734
CMA.IAC	Complement and increment AC	(2’s comp)	705

INPUT/OUTPUT TRANSFER (IOT) INSTRUCTIONS

Mnemonic	Operation	OCTAL Code
SKF	Skip on input flag	601,611
SDF	Skip on output flag	621,631
KRS	Read input buffer	602,612
LDS	Load output buffer	622,632
RKF	Reset input flag	604,614
RDF	Reset output flag	624,634

COMBINED IOT INSTRUCTIONS

Mnemonic	Operation	OCTAL Code
KRB	Read input buffer, reset flag	606,616
LDB	Load output buffer, reset flag	626,636

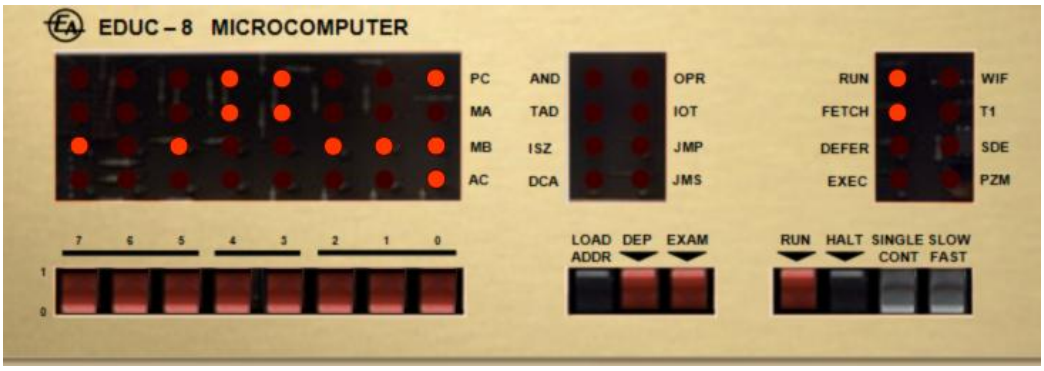
Original First EDUC-8 Test Program

```
START,     CLA             / 710 clear AC
INCR,      IAC             / 701 increment AC
BACK,      NOP             / 700 a small delay
           NOP             / 700 a small delay
           ISZ INDX        / 211 if INDX <> 0 then
           JMP BACK        / 502 jump to BACK
           ISZ INDY        / 212 incr INDY and if INDY <> 0 then
           JMP INCR        / 501 jump to INCR
           HLT             / 721 else, INDY = zero, all done, stop
INDX, #0                   / 000 Delay counter 1
INDY, #0                   / 000 Delay counter 2
```

This file is available in the install directory.

```
EDUC first.asm
```

EDUC-8 Emulator



The EDUC-8 Emulator simulates the operation of the original EDUC-8 Microcomputer.

Menu Items

Accessed by right clicking the computer

View Memory	Open ROM Memory viewer window
View RAM	Open RAM Memory viewer window - WIF only
View EEPROM	Open EEPROM Memory viewer window – WIF only
Assembler	Opens the Code Assembler window
Modules	Connect EDUC-8 to any of the available modules
EDUC-8 via USB	Opens the PC Interface
Set SR Switches	Sets all SR<7:0> switches to Logic 0 or Logic 1
Initialize	Set registers to 0, 255, or random.
Options	See Options
Speed / Animate	Opens an area on the main window to set the emulation speed of the clock cycles in slow mode and use the auto animate instruction feature.
Code Trace	Opens a window to view the current code execution trace
Help	Opens this help file
About	View about information
Exit	Closes the program On closing, the current status of all open windows will be saved and they will appear again on program start.

The computer can be moved around by left clicking the computer above the LED windows and dragging it.

View Memory Window

The Memory Viewer window shows the values of all the main internal registers.

Program Counter	PC	
Memory Address	MA	
Memory Buffer	MB	
Accumulator	AC	
IN Port 0	IN 0	
IN Port 1	IN 1	
OUT Port 0	OUT 0	In / Out Flags
OUT Port 1	OUT 1	In / Out Flags

All memory locations are displayed beneath these registers.
The values are displayed in Decimal, Hexadecimal, Octal and Binary.
The bottom bar shows the status of various operating flags and options.
The code column will show the disassembled code as created from the Assembler Window.

Menu Items

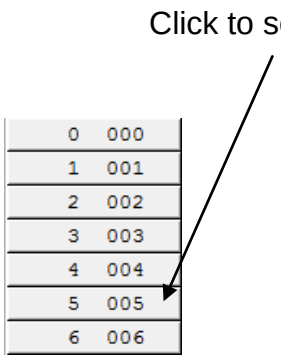
Count	Clears the current executed code counter value	
Paper Tape	Save the current memory as a Tape File.	
Breaks	Ignore	Set break points are ignored
	Clear All	Clears all break points

Break Points

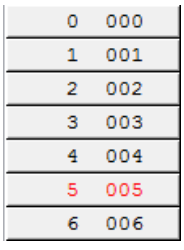
Code break points can be set or cleared by clicking on the address cell of a memory location from the memory window.

If the code has been assembled and transferred to memory, then break points can also be set or cleared by double clicking a line in the source code.

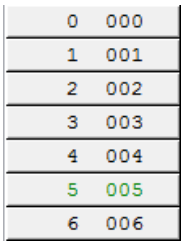
If you wish to ignore all of the set break points, then click the Ignore Breaks check box from the Editor menu.



No Break Point Set



Break point is set



Break point is set but ignored

Assembler Window

This window allows you to create source code for the EDUC-8 computer and then assemble it into binary code which can be executed.

Menu Items

New	Create a new file
Open	Open a previously saved file
Open Last	Opens the last file that was used
Recent Files	A list of previously saved files which can be opened.
Save	Save the current file
Save As	Save the current file with a new filename
Exit	Closes the Assembler Window
Help	Opens this help file
Assemble	Assemble the current file
Transfer to memory	On successful assembly, transfer the binary code to the computer and memory windows.
Set PC at cursor	Sets the PC at the cursor address
Set memory at cursor	Sets the memory viewer top row at the cursor address
Find	Find text in the opened file
Clean	Trim code of all trailing spaces and format codes (ie tabs)

Creating an Assembler File

General Line Format

The format of each line is fairly straight forward.

Label	Instruction	Comment
`START,`	`CLA`	`/ clear AC`

The text for the assembler file is not case sensitive.

`CLA` and `cla` or `Start,` and `start,` are considered to be the same.

To be assembled correctly, lines cannot wrap around to the next line.

Comments

Comments are preceded by the “/” character. Everything after this character on a particular line will be ignored by the assembler.

Labels

Labels are used to provide names for addresses in the computer memory. They are used as targets for instructions like JMP or JMS, or they provide a memory address for Memory Reference instructions like AND or TAD

Labels on each line are optional, but must be at the start of a code line.
The label name must be unique and terminated by a comma.
The maximum characters allowed for each label name is 10 and allowed characters are in the ranges: “0..9”, “a..z”, “A..Z”.

Examples:

```
Start,
Here,
AddNumber,
```

Code Examples:

```
Here,       RAL         / target label for JMP
            JMP Here
            TAD Save    / target label for Save address
            HLT

Save,       #d0         / Save initialized with 0
Mult,                   / Mult initialized with 0
Total,      Here        / Total initialized with Here
                        / address
```

Any address can be specified by a label, or as a discrete value.

Discrete values must be in the range 0 – 255.
They must be preceded by the pound character (#).
They can be specified as:

Type	Number Preceded by	Example
Decimal	#D	#D255
Hexadecimal	#$	#$FF
OCTAL	#0 (zero)	#0377
EDUC OCTAL	#E	#E737
Binary	#B	#B11111111
Music Note	#N [Delay] [Octave] [Note]	#N11C#

Examples:

```
ISZ #$FF          / Truncated to 15 ($0F) to fit in
                  / instruction ISZ bits 3-0
INDX, #d255       / INDX initialized to 255 decimal
INDX, #$0F        / INDX initialized to 15 decimal
```

Two’s Complement numbers can be represented by a negative sign.

```
INDX, #D-2        / Value entered in RAM = 254
```

Negative numbers can only be in the range -128 to -1.

JMP Computed PC

Address begins with a period (.)

Expects (+) or (-) next

Expects decimal value next (0 ..15)

Examples

```
JMP .+2    / jumps to current PC + 2
JMP .-4    / jumps to current PC – 4
```

Computed jumps must fit inside the current 16 byte page.

Music Notes

Start with	#N
Delay	0 – 3
Octave	0 – 3
Note	C, C#, D, D#, E, F, F#, G, G#, A, A#, B, X (X is a silent note for the selected delay)

Examples

```
#N21A      / Note, Delay=2, Octave=1, A
#N12G#     / Note, Delay=1, Octave=2, G Sharp
#N30X      / Note, Delay=3, Octave=0, Silent
```

Characters

Printer Chars — These characters conform to the Printer module character set Each character must be enclosed in double quotes.

Examples

```
#P"S"
#P"^"
```

ASCII Chars — ASCII characters within the range Chr(0) to Chr(127) can be entered. Each character must be enclosed in double quotes.

Examples

```
#"A"       / ASCII A       (65)
#" "       / ASCII space   (32)
```

Note: The Alphanumeric Display module only uses ASCII characters in the range Chr(32) to Chr(126). ([space] to [tilde])

Instructions

The instructions are called mnemonics and are used to tell the assembler what binary value will be generated for the computer.

```
CLA             / clear AC
IAC             / increment AC
NOP             / do nothing
```

Instruction Data

Some instructions require extra data to be specified to execute properly.

Memory Reference instructions require a memory address and whether that address is to be used as a DIRECT address or an INDIRECT address.

OPR Instructions do not require extra data.

IOT Instructions require the DEVICE number specified.

MR Examples:

Direct Memory Reference

`ISZ INDX`	the compiler will generate the binary value	0100aaaa
`TAD MEMORY`	the compiler will generate the binary value	0010aaaa

If the argument **`[I]`** follows a memory reference instruction then the address is considered to be indirect.

Indirect Memory Reference

`ISZ I INDX`	the compiler will generate the binary value	0101aaaa
`TAD I MEMORY`	the compiler will generate the binary value	0011aaaa

Note: (aaaa) represents the address where the label resides in memory.

Assume `INDX` is assembled for memory addresses 15, or 31, or 63 or 255.

`ISZ INDX I`	the compiler will generate the binary value	01011111

OPR Examples:

`CLA`	the compiler will generate the binary value	11100000
`IAC`	the compiler will generate the binary value	11100001

I/O Instructions

The actual I/O instruction specifies whether the instruction refers to an input or output device, but to complete the instruction, it requires a device argument.

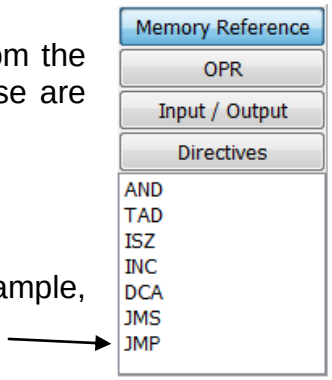
There are 2 devices available for the EDUC-8.

Device arguments are:	0	Device 0
	1	Device 1

Examples

```
LDB  0          / output device 0
SDF  1          / output device 1

SKF  0          / input device 0
KRS  1          / input device 1
```

Apart from typing in instructions, they can also be entered from the instruction listings on the right of the assembler screen. These are sorted into the different code types.



To include one of these instructions in the source code, for example, JMP, double click on it.

Assembler Directives

Assembler Directives are additional instructions that are not compiled into computer code. They are used to do various functions during assembly.

Assembler Directives start with an underscore character.

_PAGEBREAK This directive sets the PC to the start of the next available memory page.

Examples

1) If PC = $0C
   _PAGEBREAK
   PC now = $10

2) If PC = 135 decimal ($87 hex)
   _PAGEBREAK
   PC now = 144 decimal ($90 hex)

_SETPC This directive sets the PC to a specific memory address.
_setpc values can be in any radix, Eg Dec, Hex, Oct, or Bin

Examples

1) _setPC #$22
   PC = $22

2) _setPC #b1001
   PC = 9 decimal

_SUBRETURN This directive inserts the value '0' into the current PC address. It is useful for remembering that the location is used for the return address of a subroutine.

_NOTEDELAY This directive sets the note delay for the Melody Player module.
The lower the number, the shorter the delay.
The delay range is 50 to 500.
The delay value must be in entered in Decimal format.

Example _NOTEDELAY 100

_STRING This directive allows entry of strings into memory
String characters conform to the Printer module character set
The maximum string length is 18 characters
Strings must be enclosed in double quotes.

Example `_STRING "Hello from EDUC-8"`

_ISZEROPAGEOFF
_ISZEROPAGEON These directives will test to see if the Page Zero flag is set properly from the options menu. It is used to serve as a reminder that one of these modes of operation is required for the program being written. If the current Page Zero mode is not matched then an error message will be displayed.

_ISRESETT1
_ISRESETT13 These directives will test to see if the T1 T13 flag is set properly from the options menu. It is used to serve as a reminder that one of these modes of operation is required for the program being written. If the current T1 or T13 mode is not matched then an error message will be displayed.

Key Short Cuts for the editor screen.

Assemble	F9
Transfer to memory	F8
Load Address	F2
Run	F3
Halt	F4

Trace Function

The trace function allows you to see each code line execute in the Memory View window when in Single or Slow modes of operation.

Trace mode is enabled after successfully transferring the assembled file to the EDUC-8 memory.

You can also get a trace listing of the code execution by right clicking the computer for the popup menu and select [Code Trace] and make sure [On] is checked.

Once enabled a trace buffer will log the instructions as they are executed.

The trace buffer will log a maximum of 500 instructions and during fast run mode will fill in around 50mS. When full, the first item captured is deleted from the buffer so that a new item can be added to the end.

The best way to use the code trace is with break points set at critical execution points. You can then see what code led up to the break point, otherwise the buffer can fill with code that is of no use.

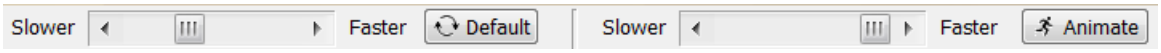
The code trace buffer is cleared on each of the following...

- The Run switch is pressed in continuous mode
- The assembled code is transferred to EDUC-8 memory
- The Code Trace is selected to On
- Manually cleared

EDUC-8 Emulator Options

System Slow Clock Speed

The Slow master clock speed can be adjusted from the menu item [Speed Control]. The lower main screen will open to show a slider bar to increase or decrease the clock speed. This can slow simulations down and make them more readable. The default button sets the cycle to the default 2Hz.



The Animate button will step the emulation automatically at a rate determined by the animate speed control. Animate will only work if the Single/Cont switch is in the Cont position and the Slow/Fast switch is in the Fast position.

From the [Options] menu item.

System Fast Clock Speed

This item sets the FAST master clock speed for simulation, as the actual clocking speed might vary between different computers. Click [Default] to set the saved default speed. Click[Save Speed as Default] to set the current speed as default.

The EDUC-8 "`First`" program takes 262,657 instructions to complete. At around 10KHz instruction cycle time, the program should run for about 26 seconds. You should be able to set the Fast Speed adjustment to accomplish this time then if you like, save that speed setting as default.

Operation Mode

Set Normal or WIF operational mode.

Set Output Reset Flag

The original EDUC-8 I/O board has a circuit board link that can select if the Output Device is reset on the T1 cycle or the T13 cycle. This allows some flexibility for output devices that require a reset before the data is transferred or a reset after the data is transferred. This option sets the IO mode of operation.

Computer Page Zero Operation Mode – Normal mode only

This switch can activate a special page zero addressing modification which allows variables from RAM pages to be accessed from Page 0. Only instructions AND, TAD, ISZ, DCA and JMS I can be used to take advantage of this type of addressing.

Example

Operation Mode = Page Zero

```
VAL,    #$0F           / Value in Page 0
        _SETPC #E720   / Code is in Page 15
START,  CLA            / AC = 0
        CMA            / AC = $FF
        AND VAL        / AC = VAL AND AC = $0F AND $0F = $0F (VAL was in Page Zero)
        HLT
```

Radix

This option sets the default radix that shows how data listed in various parts of the program. It can be set to 737 or 377 OCTAL, HEX or Decimal.

Sound Effects

Check this item if you want sound effects enabled.

Clear all memory before code compile

Check this item if you want all unused memory set to zero when compiling new code Unused memory is memory which is not part of the assembled file.

Clear all RAM before code compile – WIF only

Check this item if you want all RAM set to zero when compiling new code.

Clear all EEPROM before code compile – WIF only

Check this item if you want all EEPROM set to $FF when compiling new code.

Allow Slow Exam Deposit – Normal mode only

The EDUC-8 computer will start Run Mode if either the Deposit or Examine switches are activated when Slow running is selected. This can be annoying at times so you can enable the Deposit or Examine functions in slow mode by selecting this item.

Note: Extra LED indicators are lit when the following user options are enabled.
WIF Mode, Slow Deposit/Examine, T1 Reset, Page Zero Mode
These LEDs were not part of the original EDUC-8 design.

Upload only assembled code

Usually all memory is transferred to the project module from the USB interface. Only assembled code will be transferred to the project module if this item is checked.

Auto slow step on break

If this item is checked, then if a break point is reached during code execution, the Single/Cont switch will be put into the Single position.

Modules

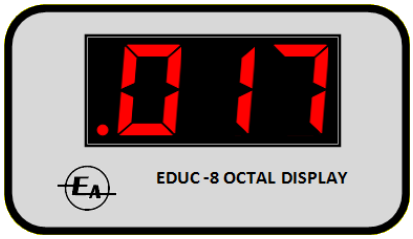
Most of the emulated modules presented here are representative of the modules described in the original project articles, others were added.

OCTAL LED Display

Only one OCTAL LED Display module can be used.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.



Menu Items

Accessed by right clicking the LED display module.

Options

Display Format	377	Normal OCTAL
	737	EDUC-8 OCTAL
Ready Delay	Sets the delay before the $\overline{\text{Ready Flag}}$ is reset to LO. 2 – 20 seconds	
Display Type	Select LED or Nixie Tube	
Reset	Resets the LED Display Module	
Disconnect	Disconnects the LED Display Module	

A sample code file for this module is available in the install file and is written for Output Device 0.

```
EDUC8 LED.asm

EDUC8 Multiply.asm
```

The LED Display can be moved around by left clicking the module and dragging it

DEC HEX LED Display

Only one Decimal/Hexadecimal LED Display module can be used.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.

Menu Items

Accessed by right clicking the LED Display module.

Options

Display Format Leading Blank Zeros
Leading Zeros
Signed Byte (Bit 7 indicates negative)
Hexadecimal

Ready Delay Sets the delay before the $\overline{\text{Ready Flag}}$ is reset to LO.
2 – 20 seconds

Display Type Select LED or Nixie Tube

Reset Resets the LED Display Module

Disconnect Disconnects the LED Display Module

A sample code file for this module is available in the install file and is written for Output Device 0.

```
EDUC8 LED.asm

EDUC8 Multiply.asm
```

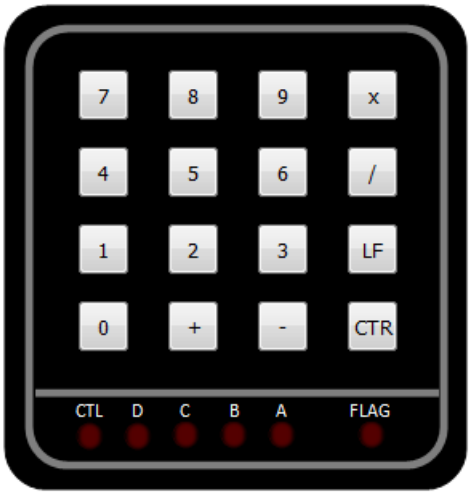
The LED Display module can be moved around by left clicking the module and dragging it.

Keypad

Only one Keypad module can be used.

When selected, a small window will open to allow selection of a free input device.

Your code should refer to this device when accessing the module.



Menu Items

Accessed by right clicking the Keypad module.

Options

Reset Resets the Keypad module

Disconnect Disconnects the Keypad module

A sample code file for this module is available in the install file and is written for Output Device 0.

```
EDUC8 Keypad.asm
```

The Keypad module can be moved around by left clicking the module and dragging it.

Keypad 2

Only one Keypad 2 module can be used.

When selected, a small window will open to allow selection of a free input device.

Your code should refer to this device when accessing the module.

Menu Items

Accessed by right clicking the Keypad module.

Options

Key Data **Set** A window will open and allow you to set the captions for each key on the keypad. The maximum characters for each key name is 3.

Keys can also be assigned a value and made invisible.

If the key caption is blank, the key will not trigger an event when pressed.

Load Load a key configuration file
Save Save the current key configuration

Reset Resets the Keypad module

Disconnect Disconnects the Keypad module

A sample code file for this module is available in the install file and is written for Output Device 0.

```
EDUC8 Keypad.asm
```

A more complex calculator program is available called `WIF Calculator.asm` and must be used in WIF mode.

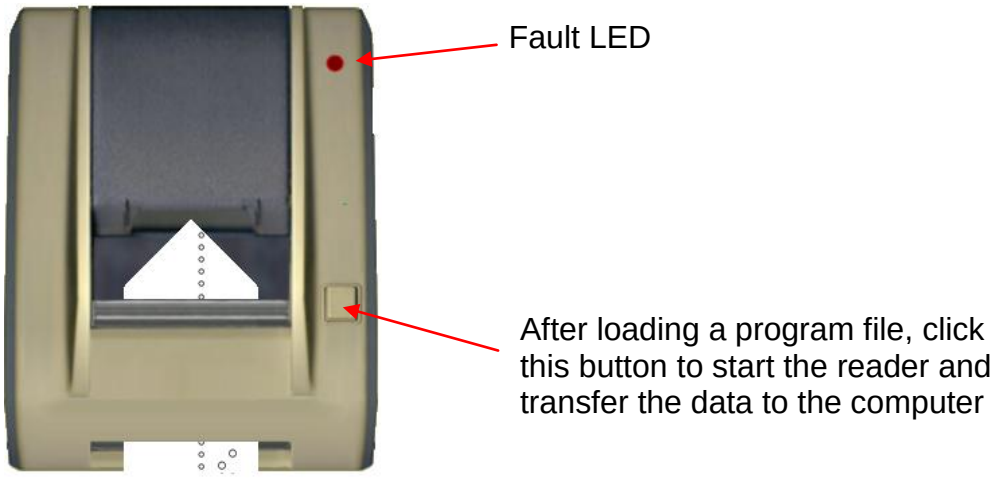
The Keypad module can be moved around by left clicking the module and dragging it.

Paper Tape Reader

Only one Paper Tape Reader module can be used, in normal mode only

As this device uses the parallel connections for the SR<7:0>, LOAD ADDR and DEP switches, this device can only connect to Input Device 0.

Your code should refer to this device when accessing the module.



Menu Items

Accessed by right clicking the Paper Tape Reader module.

Load Tape	Loads a Paper Tape file into the reader. If successful the program tape will be inserted into the reader ready for use. The first byte on the tape is a “Start” byte – 128 decimal The second byte on the tape is the “Start Address” byte where the program will start loading into memory.
Reset	Resets the Paper Tape Reader module
Disconnect	Disconnects the Paper Tape Reader module

Code is not required for the Paper Tape Reader to operate, as the reader controls the computer data transfers.

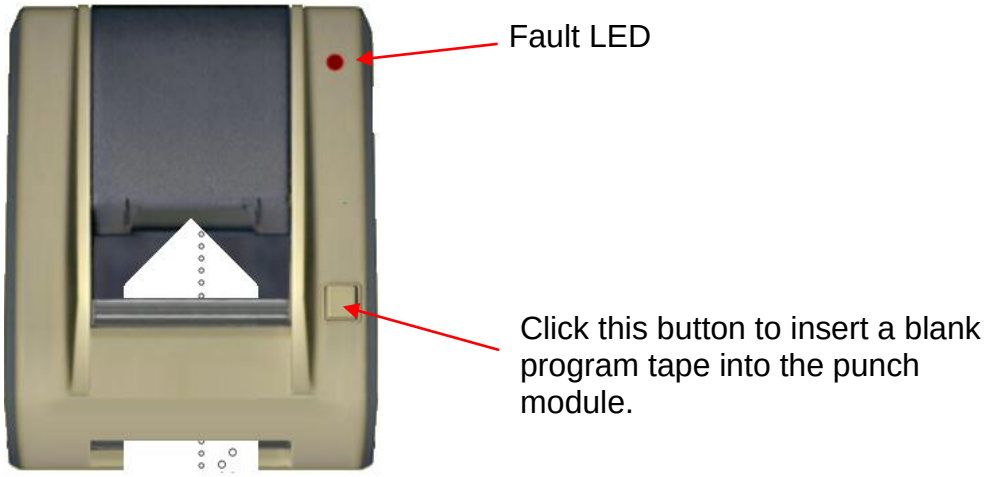
The Paper Tape Reader module can be moved around by left clicking the module and dragging it.

Paper Tape Punch

Only one Paper Tape Punch module can be used, , in normal mode only.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.



Menu Items

Accessed by right clicking the Paper Tape Punch module.

Save Tape	Saves the punched paper tape to a disk file.
Reset	Resets the Paper Tape Punch module
Disconnect	Disconnect the Paper Tape Punch module

A sample code file for this module is available in the install file and is written for Output Device 0. This is a copy of the program from the original article *Interfacing to Punched paper Tape – pg 73*.

```
EDUC8 Punch.asm
```

There is also another simple program that will transfer a section of computer memory to tape.

```
Mem Punch.asm
```

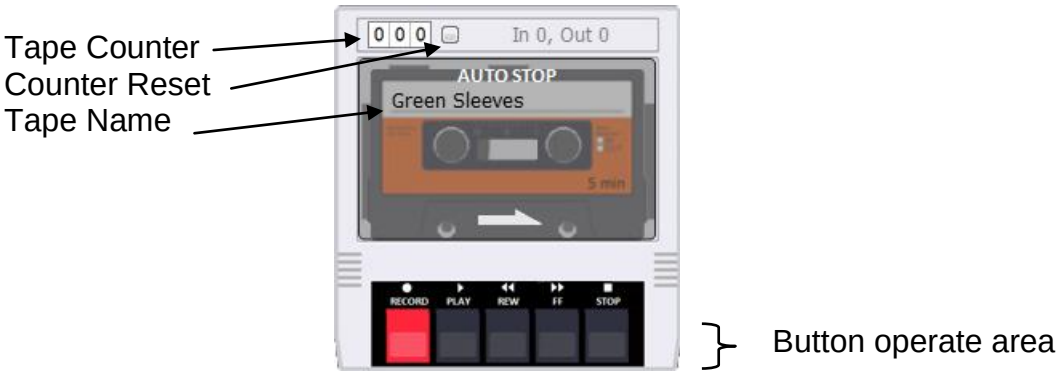
The Paper Tape Punch module can be moved around by left clicking the module and dragging it.

Magnetic Tape Storage

Only one Magnetic Tape Storage module can be used, , in normal mode only.

When selected, a small window will open to allow selection of a free output device and a free input device.

Your code should refer to these devices when accessing the module.



Tape Buttons

Menu Items

Accessed by right clicking the Tape Storage module.

Tape	New	Place a new tape in the player
	Load	Load a tape for the player
	Save	Save current tape in player
	Erase	Erase the current tape in player
	Eject	Eject the tape from the player
	Playlist	Opens a text editor to enter stored program details
Reset		Resets the Tape Storage module
Disconnect		Disconnect the Tape Storage module

A sample code file for using this module in RECORD mode is available in the install file and is written for Output Device 0.

```
EDUC8 Tape Write.asm
```

This code will save a program to tape that will play the Green Sleeves melody.

To operate the tape player for recording, position the tape ready to record by using the FF, RWD or Play buttons. Rewind back to the start for this exercise. If the counter is not showing [000], reset it.

Load Address 3. Press RECORD, then press Run on the computer.

The computer will stop writing data after the COUNT value has reached 0. Press the STOP button on the tape player.

Save the tape by selecting the Tape Save menu item.

There is also a sample code file to use the player in PLAY mode and will read the file back into the computer memory using Input Device 0.

```
EDUC8 Tape Read.asm
```

Load, assemble and transfer the program to memory.

Connect the Melody Player module to Output Device 1.
Rewind the tape.
Load Address 6, then press Run on the computer.
Press the PLAY button.

The ADDRESS memory location will increment each time a byte of data is read from the tape. When this stops incrementing, the program has loaded.

Press HLT.

Press STOP on the tape player.

The Green Sleeves program starts at EDUC-8 OCTAL address 111. Set this on the switches and click LOAD ADDR.

Press RUN. The Green Sleeves melody should play.

There is a pre-recorded tape file that has this program and can be loaded with the Tape Load menu item. The program is at the start of the tape.

```
Green Sleeves.mtf
```

NOTE: If the tape player is left in PLAY mode and there are additional bytes found on tape they will also be loaded into the computer unless your program can halt itself after loading.

The Magnetic Tape Player module can be moved around by left clicking the module and dragging it.

Melody Player

Only one Melody Player module can be used.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.

Fault LED

This will light if the PC Midi interface is not working with the software

Menu Items

Accessed by right clicking the Melody Player module.

Voice	Sets a voice for the Melody Player module Piano Chimes Synth Noise
Speed	Sets the note delay period
Reset	Resets the Melody Player module
Disconnect	Disconnect the Melody Player module

A sample code file for this module is available in the install file and is written for Output Device 0. This is a copy of the program from the original article *Teaching your EDUC-8 to play a melody – pg 81*.

```
EDUC8 Melody1.asm
```

There is also a copy of this music with the data in note format and words included.

```
EDUC8 Melody2.asm
```

See: Music Notes

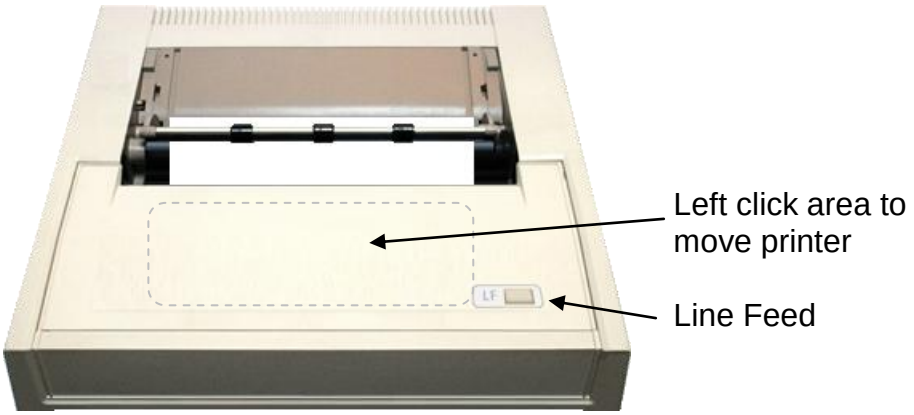
The Melody Player module can be moved around by left clicking the module and dragging it.

Printer

Only one Printer module can be used.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.



Menu Items

Accessed by right clicking the Melody Player module.

Printer Roll	Load	Load a saved printer roll
	Save	Save the current printer roll
	Print	Print the current printer roll to a real printer
	Discard	Discard the current printer roll
Reset	Resets the Printer module	
Disconnect	Disconnect the Printer module	

The printer uses the character set as described for the Philips 60SR Printer Unit.

Allowed characters are:

```
0..9 A..Z  @[\]!"$%&'()*+,-./:;<=>? Space
# = Pound £
^ = ⌐
_ = ␣
```

Example

```
/ print message from article
/ 1st 3 lines have a repeating space to fill a complete line

_STRING "HELLO,I'M EDUC-8 --~ "
_STRING "SPEAKING TO YOU VIA~ "
_STRING "THE PHILIPS 60SR~ "
_STRING "MATRIX PRINTER@"  / ends with msg terminator char (@)
```

```
HELLO,I'M EDUC-8 --
SPEAKING TO YOU VIA
THE PHILIPS 60SR
MATRIX PRINTER
```

The “~” character adds 128dec to the following printer character. This causes the following character to be printed across the remaining print line.

Example

```
_PRINT "~*HELLO WORLD!~ ~*"          / character repeat example
```

```
**********************
HELLO WORLD!
**********************
```

A sample code file for this module is available in the install file and is written for Output Device 0. This is a copy of the program from the original article *Interfacing the EDUC-8 with the Philip 60SR Printing Unit – pg 66*.

```
EDUC8 Printer.asm
```

Also included is a program to print out all of the printer codes as described in the EA ’75 article on page 69.

```
PHILIPS 60SR
CHARACTER CODE
(77 FORMAT OCTAL)

CHAR    CODE
@       00
A       01
B       02
C       03
D       04
```

Note: To better understand how the printer works, please read the original article – EA April 1975 Pg 66.

ASCII Keyboard

Only one ASCII Keyboard module can be used.

When selected, a small window will open to allow selection of a free output device.

Your code should refer to this device when accessing the module.

Please note that in order for the ASCII keyboard to respond to the PC keyboard, the ASCII keyboard should have focus by clicking on it. This is indicated by a Green LED glowing.

Menu Items

Accessed by right clicking the ASCII Keyboard module.

Reset Resets the ASCII Keyboard module

Disconnect Disconnect the ASCII Keyboard module

A sample code file for this module is available in the install file and is written for Input Device 0. It also uses the Decimal LED module on Output Device 0.

```
EDUC8 ASCII.asm
```

The ASCII Keyboard module can be moved around by left clicking the module and dragging it.

Keyboard ASCII Table

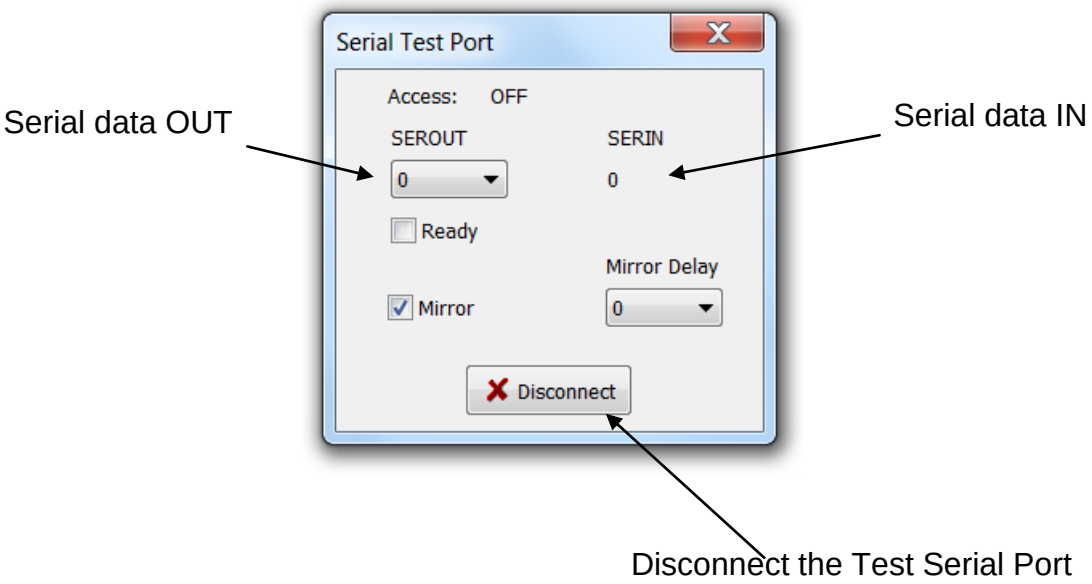
Value	Character	PC Key(s)	Value	Character	PC Key(s)
0	NUL	CTRL 0	64	@	SHIFT 2
1	SOH	CTRL 1	65	A	SHIFT A
2	STX	CTRL 2	66	B	SHIFT B
3	ETX	CTRL 3	67	C	SHIFT C
4	EOT	CTRL 4	68	D	SHIFT D
5	ENQ	CTRL 5	69	E	SHIFT E
6	ACK	CTRL 6	70	F	SHIFT F
7	BELL	CTRL 7	71	G	SHIFT G
8	BACK SPACE	BK SPC	72	H	SHIFT H
9	HOR TAB	TAB	73	I	SHIFT I
10	LINE FEED	SHIFT ENTER	74	J	SHIFT J
11	VERT TAB	SHIFT TAB	75	K	SHIFT K
12	FORM FEED	PG UP	76	L	SHIFT L
13	CAR RETN	ENTER	77	M	SHIFT M
14	SHIFT OUT	HOME	78	N	SHIFT N
15	SHIFT IN	END	79	O	SHIFT O
16	DLE	ALT 0	80	P	SHIFT P
17	DC1	ALT 1	81	Q	SHIFT Q
18	DC2	ALT 2	82	R	SHIFT R
19	DC3	ALT 3	83	S	SHIFT S
20	DC4	ALT 4	84	T	SHIFT T
21	NAK	ALT 5	85	U	SHIFT U
22	SYN	ALT 6	86	V	SHIFT V
23	ETB	ALT 7	87	W	SHIFT W
24	CAN	CTRL ALT 0	88	X	SHIFT X
25	EM	CTRL ALT 1	89	Y	SHIFT Y
26	SUB	CTRL ALT 2	90	Z	SHIFT Z
27	ESCAPE ESC		91	[	[
28	FS	CTRL ALT 4	92	\	\
29	GS	CTRL ALT 5	93	]	]
30	RS	CTRL ALT 6	94	^	SHIFT 6
31	US	CTRL ALT 7	95	_	SHIFT -
32	SPACE	SPACE	96	`	`
33	!	SHIFT 1	97	a	A
34	"	SHIFT '	98	b	B
35	#	SHIFT 3	99	c	C
36	$	SHIFT 4	100	d	D
37	%	SHIFT 5	101	e	E
38	&	SHIFT 7	102	f	F
39	'	'	103	g	G
40	(	SHIFT 9	104	h	H
41	)	SHIFT 0	105	i	I
42	*	SHIFT 8	106	j	J
43	+	SHIFT =	107	k	K
44	,	,	108	l	L
45	-	-	109	m	M
46	.	.	110	n	N
47	/	/	111	o	O
48	0	0	112	p	P
49	1	1	113	q	Q
50	2	2	114	r	R
51	3	3	115	s	S
52	4	4	116	t	T
53	5	5	117	u	U
54	6	6	118	v	V
55	7	7	119	w	W
56	8	8	120	x	X
57	9	9	121	y	Y
58	:	SHIFT ;	122	z	Z
59	;	;	123	{	SHIFT [
60	<	SHIFT ,	124	|	SHIFT \
61	=	=	125	}	SHIFT]
62	>	SHIFT .	126	~	SHIFT `
63	?	SHIFT /	127	DEL	DEL

Basic Serial Port

Only one Test Serial Port module can be used.

WIF mode use only.

The Test Serial module acts as a dumb terminal for the emulator.



Data that is received by the module is displayed in the SERIN box.

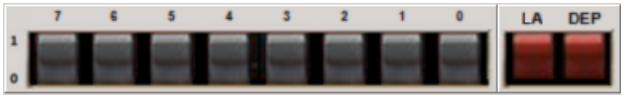
Data to be transmitted can be set in the SEROUT drop down list then when ready to send it, check the Ready item. As soon as the data is sent, the Ready item will be cleared.

If the Mirror item is checked, any data that is received is re-transmitted.

Re-transmitting of the data can be delayed by selecting a value (in seconds) in the Mirror Delay drop down list. If this value is [0] then the data will be re-transmitted as soon as it is received.

External Switches

Only one External Switches module can be used.

Normal Mode 

WIF Mode 

The External Switches module provides simulation for the switches connected to the Dev 0 parallel input port. These connect in parallel to the SR<7:0>, LA and DEP (TRIG1) switches.

Menu Items

Accessed by right clicking the External Switches module.

Set Switches Sets all switch positions to 1 or 0

Disconnect Disconnect the External Switches module

The External Switches module can be moved around by left clicking the module above the switches and dragging it.

Note: See IO Port Interfaces

10 Digit Display Module

Only one 10 digit display module can be used.

This display module emulates a 10 digit miniature 7 segment display.

Calculator Display

Clock Display

Menu Items

Accessed by right clicking the 10 Digit Display module.

Reset Resets the 10 Digit Display module

Disconnect Disconnect the 10 Digit Display module

When the display module is connected, the Ready line will be pulled LOW so display data can be sent at any time.

Character Set

```
0   Digit 0      10  -          20  F          30  P
1   Digit 1      11  :          21  G          31  q
2   Digit 2      12  .          22  H          32  r
3   Digit 3      13  BLANK      23  h          33  S
4   Digit 4      14  A          24  I          34  t
5   Digit 5      15  b          25  i          35  U
6   Digit 6      16  C          26  J          36  u
7   Digit 7      17  c          27  L          37  Y
8   Digit 8      18  d          28  n
9   Digit 9      19  E          29  o
```

Control Codes

```
$40..$49    Set Character index
$50         Turn display OFF
$60         Displays "Error"
$5F         Turn display ON
$70         Initialize display
```

7 Segment Control

```
B10000000 - B11111111

When bit 7 = 1, the other bits
control each of the 7 segments

 7-1 6-A 5-B 4-C 3-D 2-E 1-F 0-G
```

All other data is ignored.
The character index is incremented each time a new character is sent.
These characters and codes have default assembler constants defined.

The 10 Digit Display module can be moved around by left clicking the module above the switches and dragging it.

10 Digit Display Example Code

```
MOVVA $F8          / Initialize display
WRITOD 0 A

MOVVA $FF          / Turn display on
WRITOD 0 A

MOVVA #$81         / Set display character index to 1
WRITOD 0 A

MOVVA #D0          / 0 displayed in character #1 position
WRITOD 0 A

MOVVA #D34         / : displayed in character #2 position
WRITOD 0 A

MOVVA #D1          / 1 displayed in character #3 position
WRITOD 0 A



MOVVA $EE          / Displays "Error"
WRITOD 0 A
```

A sample code file for this module is available in the install file and is written for Output Device 0. It also uses the Keypad module on Input Device 0.

```
WIF Clock.asm
```

A calculator program and Number Guess game also use this module.

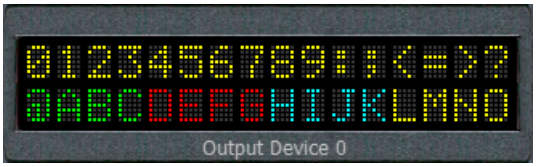
```
WIF Calculator.asm
WIF NumberHunt.asm
```

Note: These programs are for WIF mode

Alphanumeric Display Module

Only one alphanumeric display module can be used.

This display module emulates a 2 line by16 digit coloured alphanumeric display.



Menu Items

Accessed by right clicking the Alphanumeric Display module.

Reset Resets the Alphanumeric Display module

Disconnect Disconnect the Alphanumeric Display module

When the display module is connected, the Ready line will be pulled LOW so display data can be sent at any time.

Character Set

ASCII characters 32 (space) to 127 (tilde) can be used with this display module.

Control Codes

```
$00..$0F    Set Character index
$10..$13    Set new character colour - red green yellow blue
$80         Initialize display *
$81         Clear all display
$82         Clear display line 1 *
$83         Clear display line 2 *
$84         Select top line for data input *
$85         Select bottom line for data input *
$86         Turn display OFF
$87         Turn display ON

* => character index = 0
```

All other data is ignored by the module.
The character index is incremented each time a new character is sent.
Some characters and codes have default assembler constants defined.

The Alphanumeric Display module can be moved around by left clicking the module above the switches and dragging it.

Alphanumeric Display Code Examples.

```
movva AL_INI            / initialize display
WRITOD DEV0 A


movva AL_ON             / turn display on
WRITOD DEV0 A

movva AL_TLN            / select top line for data input
WRITOD DEV0 A
movva AL_GRN            / new characters are green digits
WRITOD DEV0 A

movva #A"A"             / write ASCII 'A' to digit 1, top line
WRITOD DEV0 A
movva #A"B"             / write ASCII 'B' to digit 2, top line
WRITOD DEV0 A

movva AL_BLN            / select bottom line for data input
WRITOD DEV0 A
movva AL_BLU            / blue digits
WRITOD DEV0 A
movva #A"C"             / write ASCII 'C' to digit 1, top line
WRITOD DEV0 A
movva #A"D"             / write ASCII 'D' to digit 1, top line
WRITOD DEV0 A
movva AL_YEL            / yellow digit
WRITOD DEV0 A
movva #A"E"             / write ASCII 'E' to digit 2, top line
WRITOD DEV0 A
```

See the assembler file `wif 21.asm` for an example using this display.

Note: This program is for WIF mode

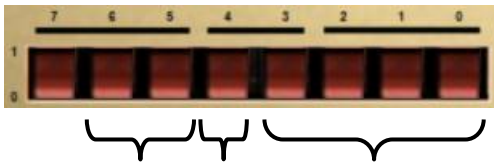
Switch Functions

Some switches on the EDUC-8 project front panel can be used for different functions.

These functions can also be accessed through the USB Interface.

Note: For clarity, the emulator switches are shown.

Set / Clear Mode Flags (SW7 not used, SW4 must be 0).



Function Select 0 Unused

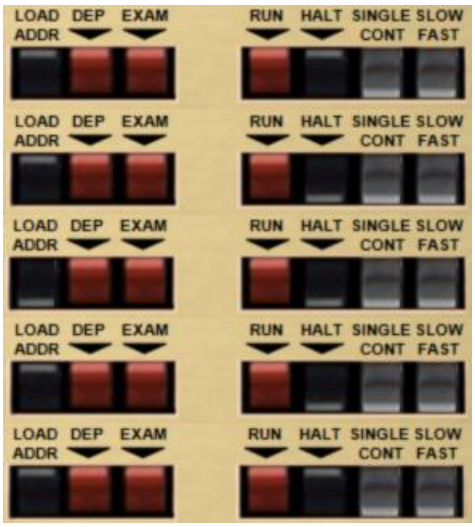
SW6	SW5	SW4	
0	0	0	Set Normal or WIF operation
0	1	0	Set / Clear Slow Exam Deposit Mode (SED)
1	0	0	Set T1 or T13 Reset
1	1	0	Set / Clear Zero Page Mode (ZPM)

Note: T1 T13 becomes Before Output (BO) and After Output (AO) for WIF mode.
SED and ZPM are not available if WIF mode is active
Changing between Normal and WIF modes resets program memory.

To execute the function press and hold HALT down, the press and release LA, then release HALT.

(2)

(3)



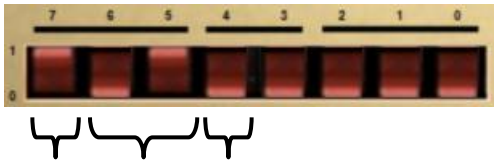
(Normal)

(1)

(4)

On successful completion, the PC register LED [0] will flash once.

For example, to enable Slow Deposit Mode, set the switches as shown, and then execute the LA and HALT switch procedure shown above.



A 01 0 A = Set (1) Clear (0)

Set Normal Operation

Set WIF operation

Set T13 (or WIF AO)

Set T1 (or WIF BO)

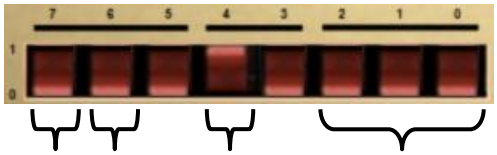
Set Normal Mode

Set Page Zero Mode

Loading and Saving Programs (SW4 must be 1).

Programs can also be loaded and saved via the front panel switches if a program is not running already.

There is storage available for 8 EDUC-8 programs and numbered 0 – 7.
There is storage available for 3 WIF programs and numbered 0 – 2.



LS Mode 1 Address

SW7	LS	Load	= 1	Save	= 0
SW6	Mode	EDUC-8	= 1	WIF	= 0
SW5	Not Used				
SW4	1				
Address	EDUC-8	SW2 SW1 SW0	(0 – 7)		
	WIF	SW1 SW0	(0 – 2) (3 selects address 0)		

Use the HALT LA switch procedure shown above to read/write the program

Examples

Load EDUC-8 Program #2

Save EDUC-8 Program #7

Load WIF Program #0

Save WIF program #1

Some configuration information is saved with each program and is based on the current operational modes.

Valid	Program is valid or not valid
ZPM	Zero Page Mode
SED	Slow Exam Deposit Mode
T1T13	IO Reset T1, T13 (BO, AO)

This information is recalled when loading a program and the current operation modes are set to these values.

- If the selected operation is successful, then the PC register LED [0] will flash once.
- If the selected program for loading is not valid, then the PC register LED [0] will flash continuously.
- If external storage EEPROM access fails, then the PC register LEDs [7] and [0] will flash continuously.

To clear the error, press the LA switch

EDUC-8 Project - Circuit Description

The circuit for the EDUC-8 emulator is fairly simple thanks to the PIC18F47K40 microcontroller. This particular chip is easily capable of driving the project and in fact a lot of its features are not used. The main reason for using it is because it is quite cheap. If it is mounted in an optional 40 pin IC socket, then it can also be reprogrammed and used for other projects if required.

The PIC clock source is internal and therefore no external crystal oscillator is required.

The contents of various EDUC-8 registers are displayed using 3mm red LEDs. To keep the project IC count to a minimum, the LEDs are driven in a multiplexed manner and have current limiting set by the series 680R resistors. This value was chosen in case the PIC freezes for an unknown reason then the maximum current through the LEDs that are turned on is limited.

The LEDs are arranged in 8 rows by 6 columns and driven by 2 of the PIC ports. PortD drives the rows of LED anodes, while PortA, RA0-RA5 sequentially drive the columns of LED cathodes. The display will not mimic the exact operation of the original EDUC-8 because they are driven in a multiplexed manner and not directly as in the original project. The refresh rate for each column is about 4mS and was chosen because the multiplexing of the LEDs can give an odd display appearance when running in fast mode.

The switches are also multiplexed with the LEDs and make use of the PIC ports capability of changing from inputs to outputs and vice versa. The switches are arranged as a matrix of 8 rows by 2 columns. As there are only 15 switches required, the second column only has 7 switches. To stop each switch from interfering with another, they are all isolated by diodes. If the diodes were omitted, then the PIC with its multiplexed method of operation would not be able to determine which switch contact was closed and they would also interfere with the LED display.

The switch rows are connected to the same LED rows that are controlled by PortD. The switch columns are controlled by PORTE pins RE0 and RE1.

During normal LED operation, all the PortD pins are set as outputs. Pins RE0 and RE1 are set as inputs and these therefore have no effect on the LEDs. It is possible at this stage that all of the switches could be in an open state which could leave RE0 and RE1 floating. To stop this happening, the PIC has internal pull-ups enabled on these 2 pins and only work when the pins are set as inputs.

The PIC scans the switch rows at approximate intervals of 25mS also which serves as a switch debounce period.

When a switch read is required, all of PortD is set to Logic 0 to discharge the pins. These pins are then reconfigured as inputs and all of the LED cathode rows are set to 5 volts to make sure no LEDs can light.

PortE pin RE0 is then configured to be a low output, thus driving the corresponding switch column low. In this configuration, the state of switches SW1 – SW7 can be read into the PIC via PortD. PortD also has internal pull-ups enabled when set as inputs, so any open switches keep the corresponding PortD pin high, while switches that are closed will cause the corresponding PortD pin to go low.

After a small delay, RE0 is set back to an input with pull-ups enabled and RE1 is set to a low output. This enables the remaining switches SW8 – SW15 to be read in the same manner.

The real EDUC-8 input port Dev0 as well as being serial, also has provision to connect to the SR<7:0> switches. In the PIC circuit, directly connecting the switches like this won't work due to the multiplexing arrangement, so these inputs which appear via connector CN7 are buffered by a 74HC244 octal buffer/line driver chip.

After reading the column of switches via RE1, the PIC sets RE1 to be an input again. It then sets RE2 to a low output. This pin is connected to the G1 and G2 inputs to the 74HC244 chip which changes the outputs from high impedance to match the state of its inputs.

These outputs now form another diode isolated switch column for the PIC to read via PortD. Once read, RE2 is set high to change the 74HC244 chip outputs back into a high impedance state. After this, the PortD pins are reconfigured as outputs and the LED operation continues until the next switch scan is required.

As it is possible for the inputs to the 74HC244 to be left unconnected, a 10K SIP resistor package (RP1) is used as pull-ups for these pins.

The real EDUC-8 input port Dev0 also has connections for the Load Address and Deposit switches.

These parallel inputs are available on pins 19 and 17 on connector CN7 and are connected to PIC pins RB7 and RB6. R19 and R18 serve as pull-up resistors for these two pins if left unconnected.

A jumper arrangement is used via CN3 for PIC pins RB6 and RB7. If required, these pins can also be used for PIC In Circuit Serial Programming (ICSP) via connector CN1. These are used just in case the external LA and DP inputs are inadvertently connected to something during programming.

As in the real EDUC-8, the PIC code ANDs the external and front panel SR<7:0> switches together, as well as the external and LA and DEP switches. Therefore to read any of the external switches, the corresponding front panel switches should all be at a logic 1 as is mentioned in the original articles. Momentary switches LA and DEP are normally logic 1.

In the real EDUC-8, the external drivers for these inputs are required to be open collector types, but in the PIC circuit these inputs can be driven high or low.

U2 is a 74HC139 dual 2 to 4 line decoder/de-multiplexer and is used because of the limited pin count on the PIC. Its function is to control some of the output pins for Dev0 and Dev1. The SELA and SELB pins are connected together and are driven by PIC pins RB0 and RB1. These two pins select one of the four active outputs from each de-multiplexer so in this configuration, the same outputs are selected.

The four outputs from U2(A) effectively steer the clock pulses to each of the Dev0 and Dev1 inputs and outputs. The four outputs from U2(B) do the same for the device resets for each of the inputs and outputs. As one output will always be active from the decoder chip, the PIC makes sure that these remain in a high state when not used. Inactive outputs from U2 revert to a logic high state which is the default for the input/output devices.

All input or output device inputs connect directly to the PIC pins via the input/output connectors CN4 – CN7. If unconnected, these PIC pins go high via pull-up resistors R10 – R15.

U1 is a I2C serial EEPROM chip with 8K by 8 bit memory available. It has the capacity to store up to 8 programs for the EDUC-8, plus an additional 3 programs for WIF mode. It can be accessed by the front panel switches or via the USB PC interface. R16 is used as a pull-up for the SDA pin.

C1, C2, C3 and C5 are used as decoupling capacitors for each of the chips while C4 is used as a small tank capacitor for the PIC.

R9 is used as a pull-up for the PIC master clear (MCLR) pin and holds it high in normal operation. When PC communications are connected via the USB to Serial module, this pin is controlled by the DTR pin under PC control.

Power for the EDUC-8 board is provided via USB connector CN1. If the recommended USB to Serial module is inserted then power is supplied via the PC. A suitable connector can be made up to supply a regulated 5V supply if other power sources are required. Pin 4 is VCC and pin 6 is GND. 4.5 volts from 3 AA cells could also be used.

Be careful with polarity or serious damage could result. All of the chips should work with a supply of around 4 – 5V. Do not attempt to use the serial convertor connected to the PC and an external supply.

The PIC master reset (MCLR) is connected to pin 1 and is usually controlled by the PC, but can be controlled externally by pulling the input low. It is held high by resistor R9.

*With the PIC running on its own internal oscillator, the internal UART error is specified as 0.16%. This in itself is not a problem, and even though the PICs internal timer is laser trimmed for accuracy, temperature effects may still affect this timing slightly. As such the serial communication “packets” are kept short so that only a few bytes are sent at a time.

EDUC-8 Project – Construction

Please check the PCB for damage and read through these notes before starting construction. The four PCB mounting holes may need to be enlarged if you decide to mount the project in the plastic enclosure.

It is easy to construct this project if you have a small bench top vice to hold the PCB while soldering the project parts. There are some PCB overlay diagrams immediately following this section to help with construction.

The parts look neat if they are placed properly, but there is no need to be perfect. It is important not to overheat these small parts.

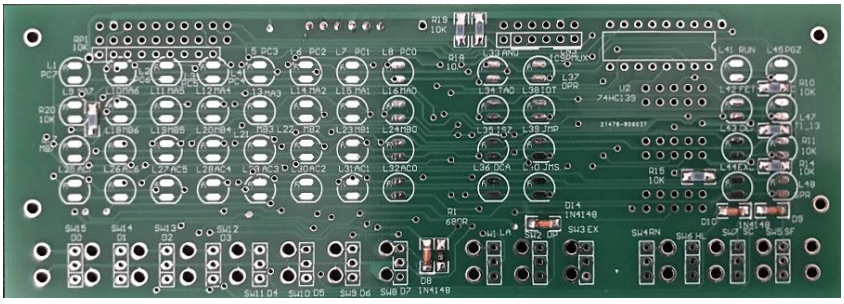
Start with soldering the surface mount components on the top side of the PCB. Try to solder only one part type at a time, eg 10K resistors. The solder pads for all SMT components have been enlarged slightly to make construction easier. An easy method of soldering these parts is as follows.

Melt a small pool of solder onto a single PCB pad of all the SMT parts. For example, resistors have two to pads to solder, so only melt a solder pool for one pad and leave the other PCB pad clean.

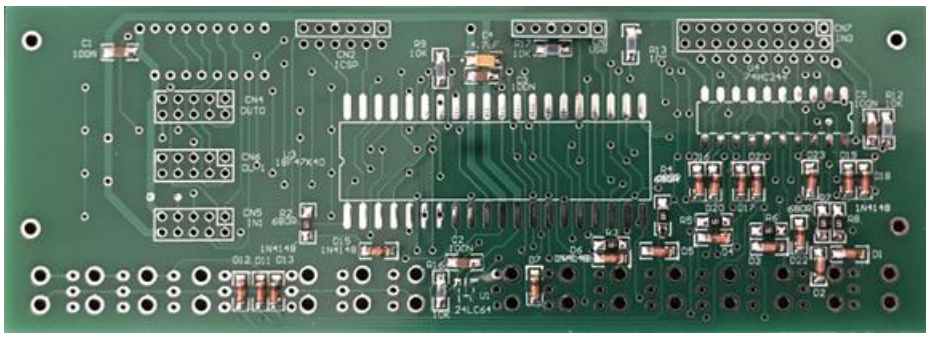
Place the SMT components on the PCB and (as usual) they will probably land upside down. Tip them all over so that the label is facing upward. It is nice to have all the labels oriented in the same direction.

Assuming a right handed constructor, with some tweezers, move the component to the PCB pads with your left hand. Place the soldering iron tip in the centre of the solder pool to melt it again and immediately slide the component to the right until it butts up against the soldering iron tip. The component should be in the correct place. You may need to rub the soldering iron up and down the edge of the component to get the solder to bond. Remove the soldering iron and hold the component for a few seconds until the solder cools. Repeat this procedure for all other SMT components for this side of the board.

Take care when mounting the diodes to make sure they are oriented correctly. The cathodes are marked with a black line and they match up with the heavier lines on the solder mask. Once all parts are mounted, you can go around and solder the opposite PCB pads for all components. After this recheck your work.

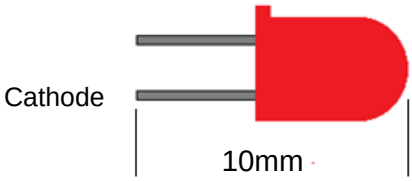


When you are satisfied with the top of the PCB, you can mount the SMT components for the bottom side in the same manner. Some components are placed close together and there is not much room remaining for solder mask component values. These are mostly the 680R resistors and diodes.



Please note: At this stage, the 24LC64 memory chip may remain off the board.

The 3mm red LEDs can be mounted next. The leads need to be cut as follows.



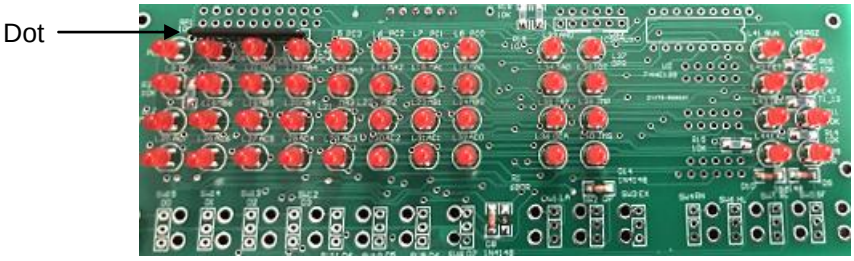
To make it easy to cut the leads and mount the LEDs onto the PCB, if you like you can make up a small holding jig. The one shown in the diagram was made out of balsa wood, but can be anything that is convenient. A hole was drilled into one end with a 7/64” drill. The 3mm LEDs fit snugly into this diameter hole.

One end was marked to match with the cathode of each LED, which has a flat edge moulded into the side of the LED package.

Using a similar procedure to mounting the SMT components, place a pool of solder onto 1 pad of each LED on the top of the PCB. All LEDs have the cathode (flat surface) towards the switches.

Starting at one end of the PCB, using the tool, align and hold the LED on the top of the solder pool. Melt the solder pool and the LED will drop down to the PCB surface. Remove the soldering iron and let the solder cool. The LEDs have short leads and can be damaged by too much heat so be careful. Removing a defective LED later on will be difficult. Complete an entire row of 12 LEDs using this method. Once complete, check the orientation again and solder the other leads.

You can use the tool to align each LED vertically.



Mount the 10K SIP resistor (RP1) next as shown. Observe the polarity as the component has a common pin marked with a dot. This pin should be closest to L1.

The switches can be mounted next. There are 2 types. There are 10 switches that are normal toggle types and 5 switches that are momentary types. The following diagram shows the normal toggle types mounted.

The 5 momentary type can be mounted in the remaining positions. If all of these switches are mounted so that the levers point towards the LEDs, they will be in the correct orientation.

Note: The PCB has provision for the larger C&K type toggle switches if the miniature ones cannot be purchased.

U2, a 74HC139 can be mounted next. Solder the power supply pins first. Check orientation and solder the remaining pins.

There are now static sensitive components on the PCB, so please handle it with due care. Holding the PCB by the edges helps.

This completes the top side PCB construction.

All of the IDC connectors on the bottom side of the PCB can be mounted next. CN1 and CN2 are female 6 pin connectors. The others are all male.

If required, a 40 pin IC socket can be mounted for the PIC18F47K40 chip. The socket needs to have all its pins bent outward at 90 degrees so that it becomes a surface mount socket. Align the socket and solder 1 corner pin. Check alignment and solder the opposite corner pin. If satisfied with the positioning, solder all the other pins.

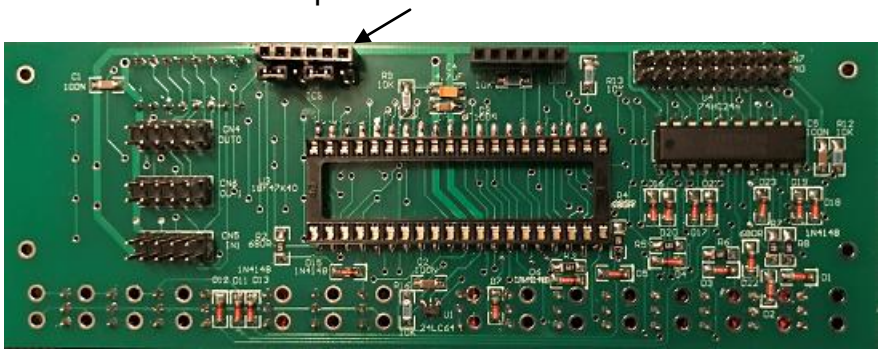
If the PIC chip is to be mounted directly to the PCB, then all the pins need to be bent inwards before soldering. (See mounting the 74HC244 chip later in text)

U3, a 74HC244 chip, needs the pins bent inwards so that it can be mounted as a surface mount part. The component can be held carefully by its ends with the pins rested on a hard surface at about 45 degrees. Then press gently downwards and all the pins will bend inwards where the pins narrow. Do the same to the other side. You may have to lay the component flat on a hard surface and press gently down to complete the 90 degree bends all at once.

Position the part, which will be slippery on the PCB, and solder the power supply pins first. Check orientation and solder the remaining pins.

Mount the 24LC64 memory chip next. The pins are small so care is required. Use the same technique as described previously by placing a small solder pool on one of the pads on the PCB, then solder that single pin while holding the part with tweezers. Then solder the remaining pins.

Place the 2 jumper shunts in the positions shown for normal operation. One shorts pins 2 and 3 and the other shorts pins 5 and 6.



This completes the board construction.

If a PIC programmer is available, then the PIC18F47K40 can be programmed using the HEX file `educPIC.hex` available in the installation files.

Once programmed, the PIC can be inserted into its socket. Pin 1 is closest to the 3 10 pin IO sockets. See ISCP Programming.

Note: If the PCB cannot be purchased, there are Gerber files which can be used by a PCB manufacturer to create the board.
See installation/schematics/gerber directory – `educ8.zip`

EDUC-8 Project – Case Assembly

The enclosure chosen for this project is available from Altronics – Part No. H0290.

The inside of the case needs to be machined out so that the PCB can fit comfortable inside. There are walls and support areas for the battery which need to be removed as well. The PCB will sit about 10mm above the case bottom.

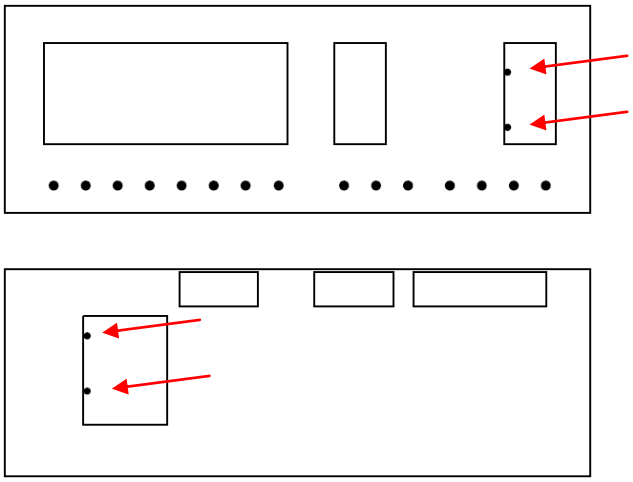
An easy way to accomplish this is to use a Dremmel Tool (Part 196) which comes in packs of 2. It is easiest if the tool is placed in a bench drill so that the case can be held and manoeuvred by hand while the excess plastic is removed. Time and care should be taken as trying to remove too much plastic in one “bite” may cause the tool to dig in and fling the case out of your hand or damage it. Moving the tool up and down is best. Moving the case sideways to remove plastic may cause the tool to bite in.

The battery enclosure door can be glued in place using araldite or other plastic glue.

If you decide to use this enclosure, there is a PDF file available in the installation sub directory, `../Schematics` that can be used as templates for the front and back panels. If you print this file with 100% scaling, the templates should match the PCB and case dimensions.

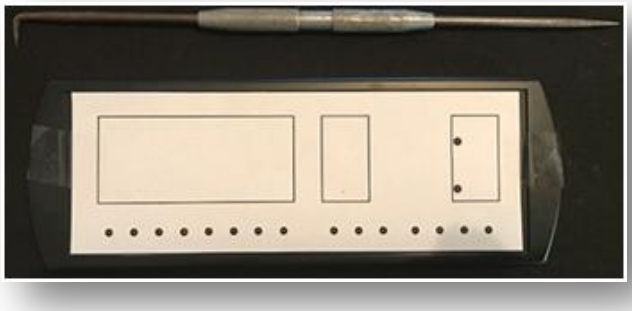
```
Front_Back_PanelsTemplate.PDF
```

The marked holes are alignment holes. If you cut out the front panel template and fasten it centrally onto the front panel of the case with sticky tape. You can then drill 2 x 1/16” holes completely through the front and back case panels using the template as a guide. The back panel template can then be aligned to these holes.



Start with the front panel.

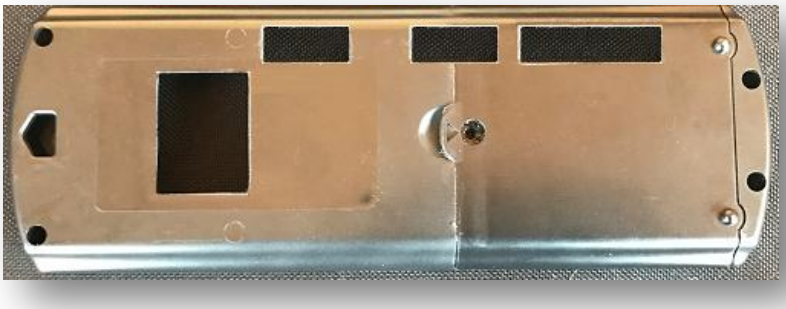
Use a scriber point to push down (not too hard) though each corner of the square holes and in the centre of each switch. This procedure will leave a small indent on the plastic face panel when the paper template is removed. Use the scriber to join up the 4 sides of each hole on the plastic surface.



The holes can be cut out by drilling 3/16” holes along the inner sides of the lines, making sure the drill does not protrude to the outer side of the lines. Once the inner plastic is removed, use small files to square up the holes. It is not imperative that the holes be perfectly square as the front panel artwork should cover them to a certain degree.

The switch holes can be drilled next. Drilling one large hole can cause the drill bit to bite and tear the plastic, especially if the drill is sharp. Start will smaller drills, say 1/8” and progressively drill larger holes up to 13/64”.

Use a similar procedure for the rear panel holes.



There are 4 standoffs moulded into the inside of the upper casing to allow mounting of a smaller PCB. These need to be removed using the Dremmel tool taking care not to go right through the case.

One the case halves are all completed, the four 10mm spacers can be attached to the rear of the PCB with the M3 screws. The mounting holes in the PCB will need to be enlarged with a 1/8th inch drill.

The back panel can either be drilled to mount the PCB spacer or you can use 5 minute Araldite glue to attach the spacers to the inside of the back case wall. Place the glue at the approximate positions for the spacers then place the PCB into the back case and attach the front case half to allow the switches to align the PCB.

Make sure the PCB is sitting in the correct position and let the glue set for at least a few hours, or overnight. More glue might need to be added when the PCB is removed to strengthen the mounts.

The front panel artwork for the prototype was printed onto a sheet of clear transparency with an inkjet printer. The artwork is a reversed image so that it will appear normal when the sheet is flipped over.

The artwork is contained in a file called `EDUC8_FACES.pdf` and is available in the installation directory. After printing and drying, these images can be trimmed to fit the top enclosure face. It is difficult to create neat holes for the switches to poke through, so the artwork was trimmed to sit just above the switches.

The prototype had 4 small holes drilled into the corners of the top enclosure half to accept small self tapper screws. Two slightly larger holes were punched into the overlay sheet matching the top two hole positions. This has to be done carefully as the sheet can tear. I used a hole punch and hammer with the sheet on a hard surface. Use the self tapping screws to lightly secure the sheet to the front panel. The bottom 2 screws are there just to balance the appearance. This method allows easy changing of the panel artwork for the two operating modes and of course, any other suitable method will do.

Once the enclosure modifications are completed, it can be stripped down again. The top face edges can be masked with tape and then the inner area sprayed with gold coloured paint. The overlay will show up nicely with this colour. Leave it to dry for some time before final assembly.

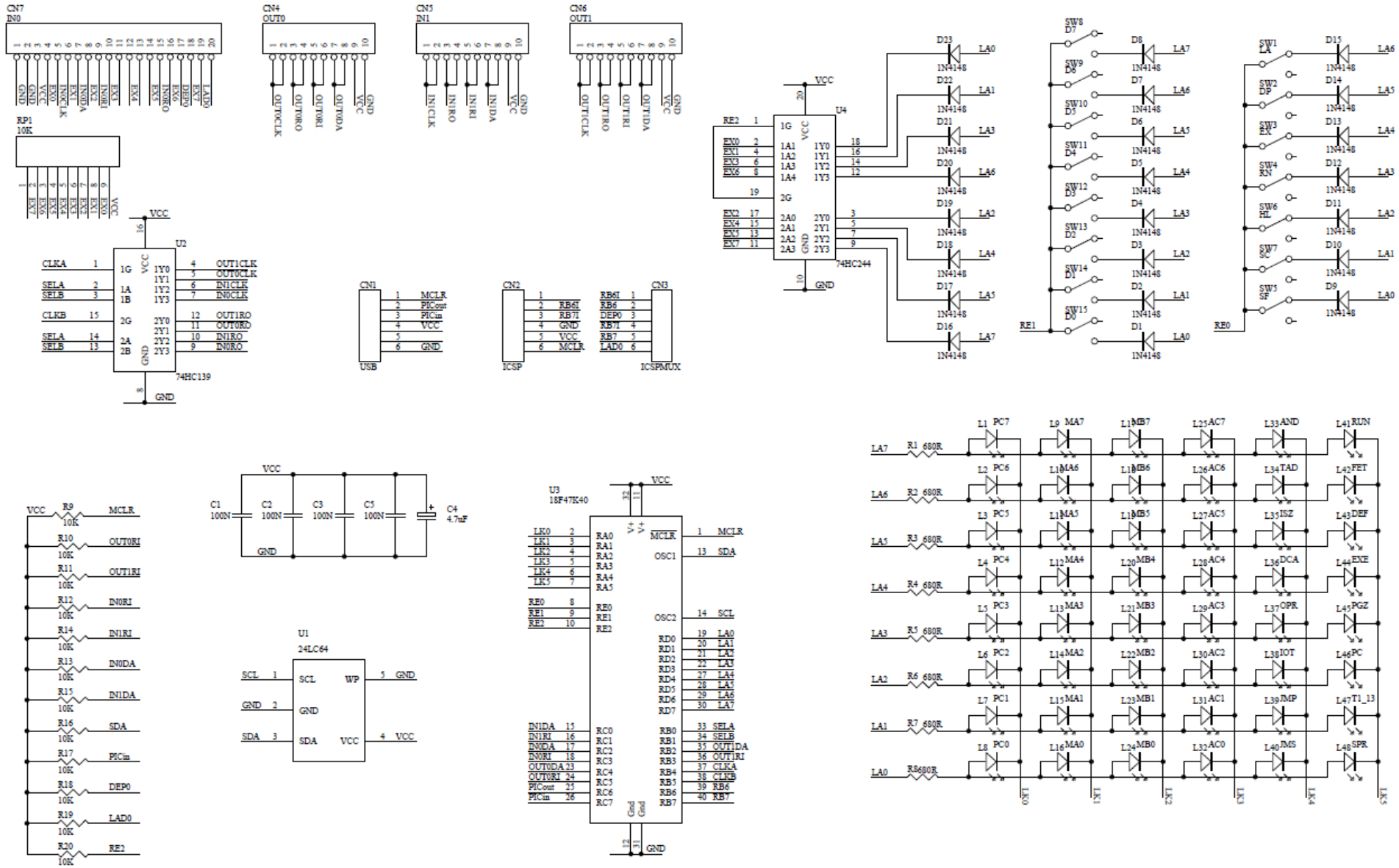
Once the paint is dry, some red plastic sheet can be taped on the inside of the top casing to provide a sealed window for the LEDs.

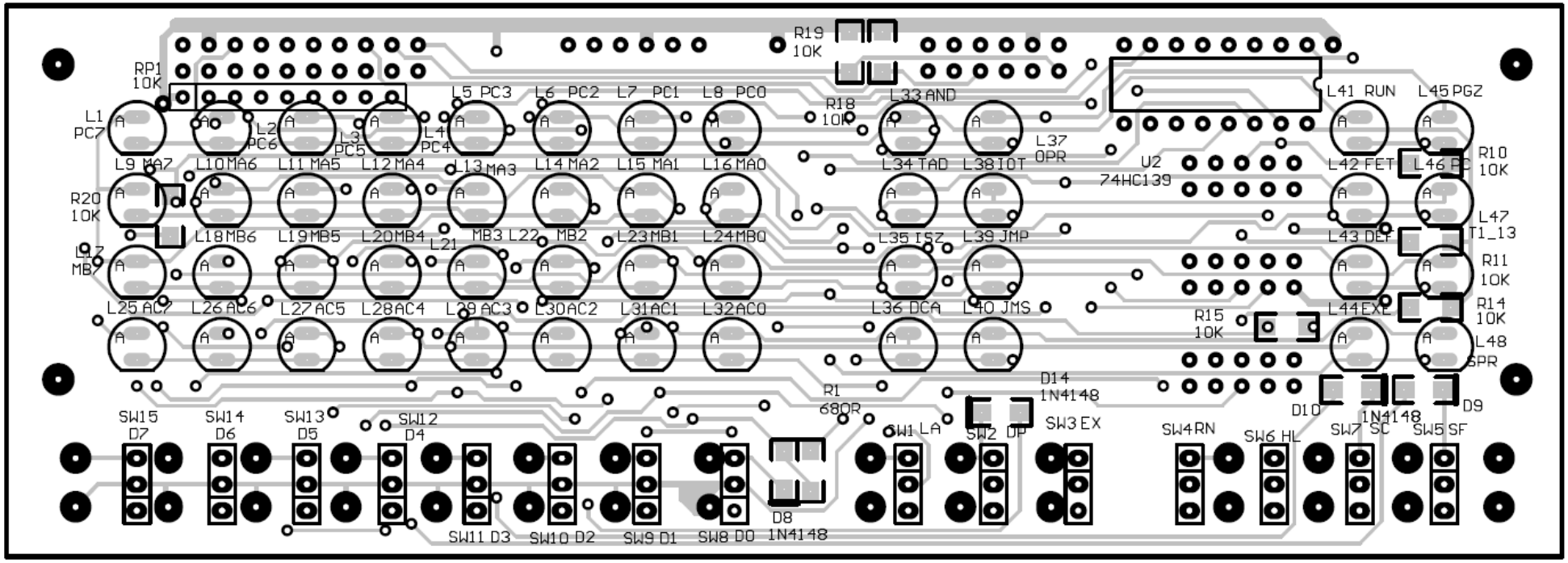
There is a file called `deviceLabels.pdf` that has some text that you can use to make up some labelling for the rear panel. These were printed, cut to size and stuck to the rear case wall with some clear nail polish.

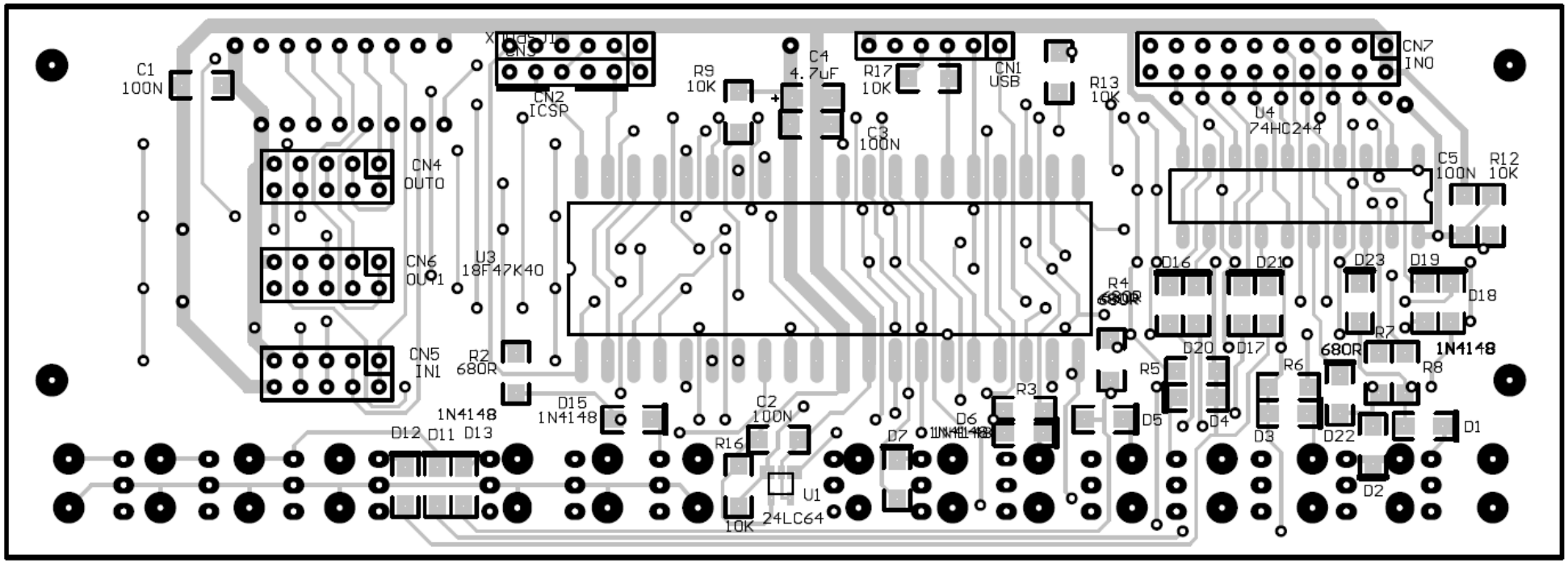
Completed Front Panel

Completed Rear panel

Working EDUC-8 with 2 completed modules

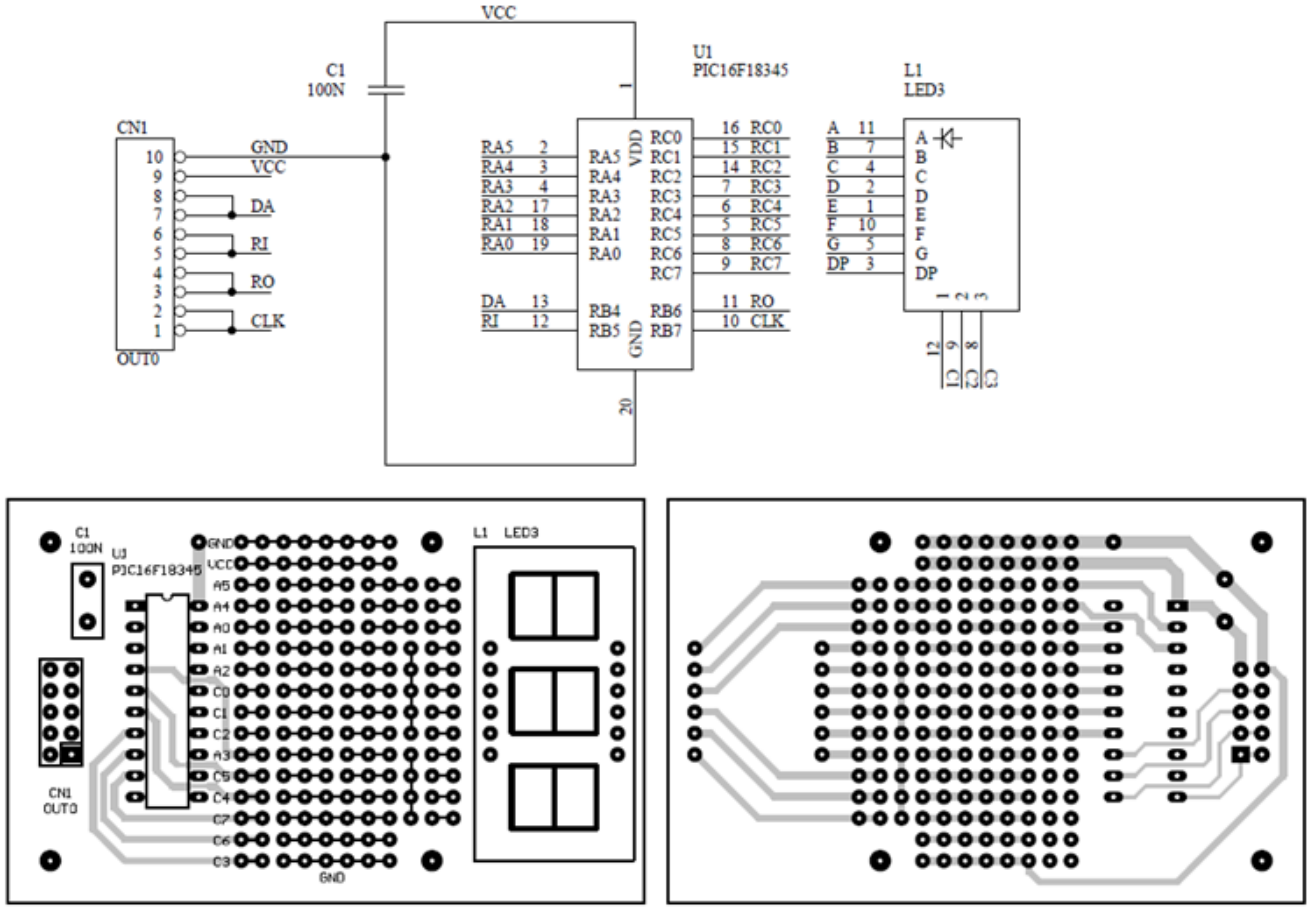

CN7
IN0
CN4
OUT0
CN5
IN1
CN6
OUT1
RP1
10K
U2
74HC139
U4
74HC244
CN1
USB
CN2
ICSP
CN3
ICSPMUX
SW8
SW1
RE1
RE0
C1
C2
C3
C5
C4
4.7mF
U3
18F47K40
U1
24LC64
L1 PC7
L2 PC6
L3 PC5
L4 PC4
L5 PC3
L6 PC2
L7 PC1
L8 PC0
L9 MA7
L10 MA6
L11 MA5
L12 MA4
L13 MA3
L14 MA2
L15 MA1
L16 MA0
L17 MB7
L18 MB6
L19 MB5
L20 MB4
L21 MB3
L22 MB2
L23 MB1
L24 MB0
L25 AC7
L26 AC6
L27 AC5
L28 AC4
L29 AC3
L30 AC2
L31 AC1
L32 AC0
L33 AND
L34 TAD
L35 ISZ
L36 DCA
L37 OPR
L38 IOT
L39 JMP
L40 JMS
L41 RUN
L42 FET
L43 DEF
L44 EXE
L45 PGZ
L46 PC
L47 T1_13
L48 SPR


RP1
10K
R19
10K
R18
10K
L1
PC7
L2
PC6
L3
PC5
L4
PC4
L5 PC3
L6 PC2
L7 PC1
L8 PC0
L9 MA7
L10 MA6
L11 MA5
L12 MA4
L13 MA3
L14 MA2
L15 MA1
L16 MA0
R20
10K
L17
MB7
L18 MB6
L19 MB5
L20 MB4
L21
MB3
L22
MB2
L23 MB1
L24 MB0
L25 AC7
L26 AC6
L27 AC5
L28 AC4
L29 AC3
L30 AC2
L31 AC1
L32 AC0
L33 AND
L34 TAD
L35 ISZ
L36 DCA
L37
OPR
L38 IOT
L39 JMP
L40 JMS
U2
74HC139
L41 RUN
L42 FET
L43 DEF
L44 EXE
L45 PGZ
L46 PC
R10
10K
L47
T1_13
R11
10K
R14
10K
L48
SPR
R15
10K
R1
680R
D14
1N4148
D10
1N4148
D9
SW15
D7
SW14
D6
SW13
D5
SW12
D4
SW11 D3
SW10 D2
SW9 D1
SW8 D0
D8
1N4148
SW1 LA
SW2 DP
SW3 EX
SW4 RN
SW6 HL
SW7 SC
SW5 SF


C1
100N
CN2
ICSP
R9
10K
C4
4.7uF
R17
10K
CN1
USB
R13
10K
CN7
INO
U4
74HC244
C3
100N
C5
100N
R12
10K
CN4
OUT0
CN6
OUT1
U3
18F47K40
CN5
IN1
R2
680R
D15
1N4148
1N4148
D12 D11 D13
C2
100N
R16
10K
U1
24LC64
D7
D6
R3
R4
D16
D21
D23
D19
D18
D20
D17
680R
R7
1N4148
R5
R6
R8
D5
D4
D3
D22
D1
D2

Device Interface Circuit

This circuit provides an experimenters interface to the EDUC-8 IO ports. The circuit board details are available as a Gerber file which can be sent to PCB manufacturer to make these boards. See `installation/schematics/gerber directory – device.zip`

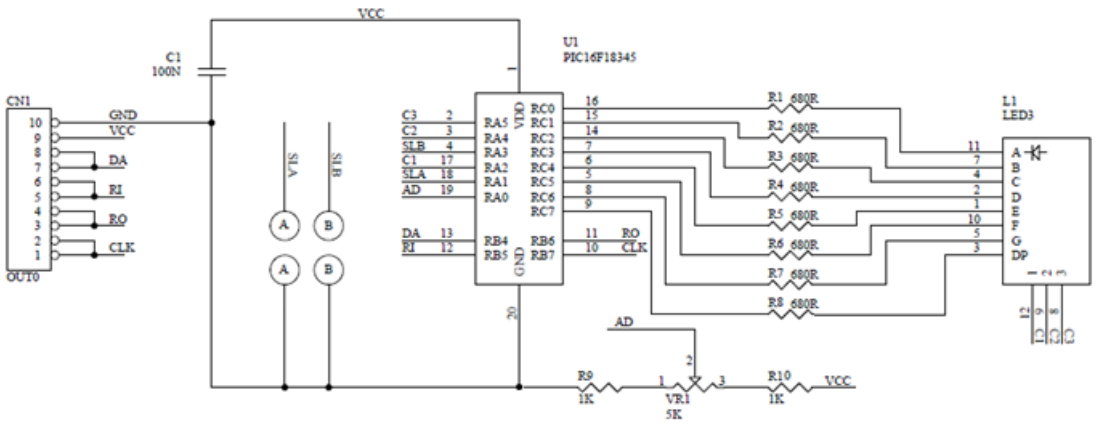


LED Module Circuit

This circuit demonstrates a connection to an output port. The module assumes you know something about programming a PICmicro in assembler. There is a source code file called `LEDmodule.asm` available to control this module in the `installation/PIC` directory. The compiled HEX code is also available and is called `LEDmodule.hex`. The PIC16F18345 was chosen as it has plenty of memory and peripherals and is quite cheap. It can also be used for other projects as desired.

Sample programs for the EDUC-8 in both normal and WIF modes are available in files `EDUC8_LED.asm` and `WIF_LED.asm`.

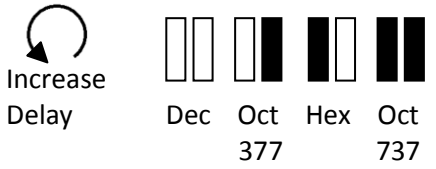
Note: You can use ICSP to program the PIC by wiring up a simple circuit on a breadboard.



MODE	A	B
DEC	0	0
HEX	0	1
377	1	0
737	1	1

0 = Open
1 = Closed

Trimpot VR1 is used to control the delay the EDUC-8 program uses to update the LED display. Increasing the voltage at the wiper, will increase the delay.

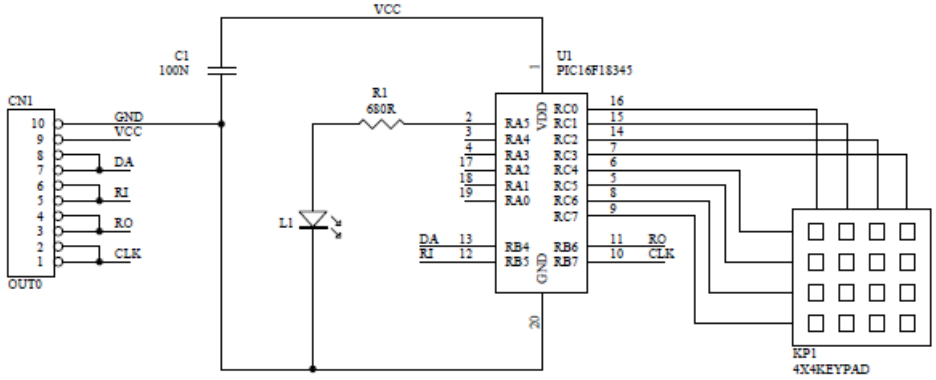


Key Module Circuit

This circuit demonstrates a connection to an input port.

There is a PICmicro source code file called `KEYmodule.asm` available for this project in the installation/PIC files. The complied HEX code is also available and is called `KEYmodule.hex`.

Sample programs for the EDUC-8 in both normal and WIF mode are available as `EDUC8_BasicKeypad.asm` and `WIF_BasicKeypad`.asm.



[D] key toggles data bit 5

EDUC-8 Project Parts List

Part	Used	Part Type	Designators		Element14 Part
1	1	PCB	EDUC-8		
2	1	24LC64	U1		1700992
3	1	74HC139	U2		9591117
4	1	PIC18F47K40	U3		2564267
5	1	74HC244	U4		1470753
6	10	SPDT 2MS1T2B2M2RE*	SW5 SW7 SW8 SW9 SW10 SW11 SW12 SW13 SW14 SW15		9472967
7	5	SPDT (Mom) 2MS2T2B2M2RE*	SW1 SW2 SW3 SW4 SW6		9472983
8	23	LS4148	D1 - D23	SOD-80	9549994
or	23	1N4148	D1 - D23	SOD123	2433353
9	48	3mm RED LED	L1 – L48		1581113
10	1	4.7Uf TANTALUM	C4	1206	1672491
11	4	100nF CERAMIC	C1 C2 C3 C5	1206	2497075
12	8	680R RESISTOR	R1 – R8	1206	2671204
13	12	10K RESISTOR 1/8W 5%	R9 – R20	1206	1632523
14	1	10K X 8 SIP RESISTOR	RP1		1612533
15	2	2.54mm HEADER SOCKET 6x1	CN1 CN2		1593462
16	1	2.54mm HEADER 6x1	CN3		1593415
17	3	2.54mm HEADER 5x2	CN4 CN5 CN6		1593442
18	1	2.54mm HEADER 10x2	CN7		1918006
or	2	2.54mm HEADER 5x2	CN7 (Same as part 17)		1593442
19	1	40 pin IC SOCKET			4285669
20	2	JUMPER SHUNT			1632170
21	1	CASE	ALTRONICS	H 0290	
22	1	FTDI USB/SERIAL MODULE	JAYCAR	XC4464	
or	1	FTDI USB Breakout	ALTRONICS	Z6225 (Cheaper)	
or	(See Ebay: Real cheap)				
23	4	METAL THREADED SPACERS	JAYCAR	HP0900	1466761
24	4	M3 x 6mm SCREWS	JAYCAR	HP0400	2494538
25	1	RED PLASTIC FILM	Ebay		
		FRONT PANEL ARTWORK	Home print		
		BACK PANEL LABELS	Home print		

* Larger C&K type switches are catered for on the PCB design.

Basic Project Module Parts List

Part	Used	Part Type	Designators	Element14 Part
1	1	PCB	Serial Module	
2	1	PIC16F18345 DIP	U1	2474834
3	1	100N MKT Capacitor	C1	211275102
4	1	2.54mm HEADER 5x2 or	CN1	1918006
	1	2x5 IDC Transition Connector	CN1	
5	1	20 Pin IC Socket	(Optional)	4285608
6	1	10 way IDC cable 150mm		2217607

Cable length to suit user but not too long or capacitive effects may upset signals

LED Module Extra Parts List

Part	Used	Part Type	Designators	
1	1	3 Digit 7 Segment Display	LD1	2627649
2	8	680R Resistor 1/8W 5%	R1-R8	1128121
3	2	1K Resistor 1/8W 5%	R9-R10	1128859
4	1	5K Trimpot	VR1	2859737
5	2	2.54mm HEADER 3x1	A-B	1593412
or	1	2.54mm HEADER 6x1	A-B	1593415
6	2	JUMPER SHUNT		1632170

KeyPad Module Extra Parts List

Part	Used	Part Type	Designators		
1	1	5mm Red LED	L1		2322131
2	1	680R Resistor 1/8W 5%	R1		1128121
3	1	4 x 4 Keypad	KP1	Altronics	S5383

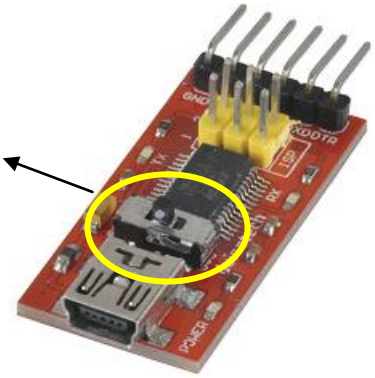
USB Interface

The USB interface allows you to communicate with the EDUC-8 hardware project via the PC USB Port.

The interface is accessed by the menu item [EDUC-8 via USB] after right clicking on the EDUC-8 emulator.

It is important to note that the Interface Window should not be opened or closed unless the hardware is connected to the PC user port, otherwise the PC may hang while it tries to sort out why the USB device has changed state.

The interface unit is a commonly available USB to Serial Converter which can be purchased from many electronic hobbyist outlets. Full instructions and a USB cable are provided with the unit.



Note:

As well as providing communications to and from the PC, this unit also powers the EDUC-8 project. Make sure the interface board switch is to set for 5V Rx/Tx operation before use with the EDUC-8 project.

The interface board may be damaged by static electricity. Please handle the board only by its edges.

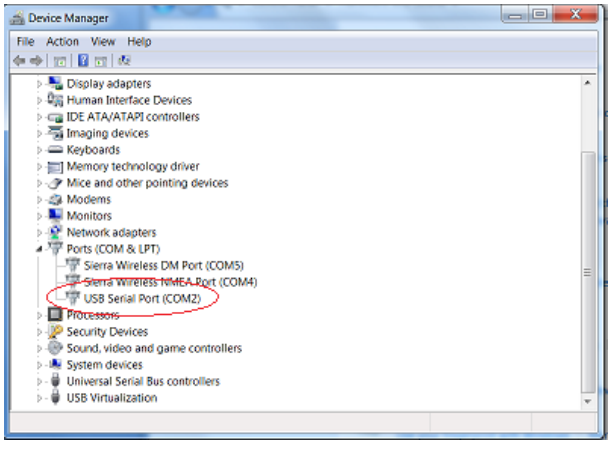
This module can also be used for other projects if required.

Initial Setting Up

Connect the interface board to a free PC USB port and the board should work properly but if there are problems visit http://www.ftdichip.com/ for more details on their drivers.

You need to know the COM port being used for the device.

This can be accessed from the Control Panel – Hardware and Sound – Device Manager – Ports (COM & LPT) – USB Serial Port (COMx) where **x** should be set to a value between **1** and **256**.

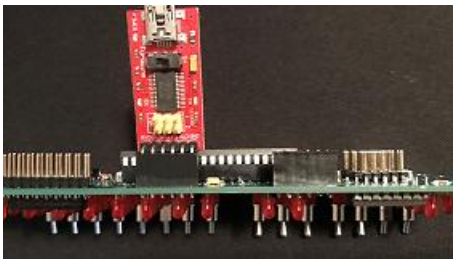


To do this, double click the USB Serial Port entry to open the Property Viewer, click the Port Settings Tab and then click Advanced. You will see a COM Port drop down list where you can set the port number to one of the values described above.

Close all Control Panel windows by clicking on OK.

Disconnect the USB interface from the PC.

Connect the USB interface to the EDUC-8 board USB connector. This is the 6 pin socket located nearest to the Dev 0 Parallel Input socket. The USB interface board should have its components facing towards the PCB top as shown in the picture below.



Connect the USB interface to the PC. On initial connection to USB port, the EDUC-8 may flicker while the PC stabilises the port, but once connected, the EDUC-8 should start to run and the Fetch LED should stay lit.

Run the EDUC8uc.exe program

Right click the computer and select the menu item [EDUC-8 via USB] or press F5.

The Interface Window should open.

If the correct COM Port number was entered, the PC will establish communications with the EDUC-8 project and text similar to below will appear in the list window.

```
EDUC-8 connection query... >OK

Operating Mode                = Normal
Page Zero Mode                = OFF
Slow Deposit/Examine Mode     = OFF
T1/T13 Reset Mode             = T13
Slow Speed Cycle Time         = 500mS Default
Clear registers on reset      = ON
```

The text shows that the connection was established and lists the state of the various operating options.

If an error message appears then you may need to set the correct USB port.

To do this, click on the COM Port radio button and enter the same COM Port number that was set from the Control Panel procedure outlined above.

Once entered, press the ENTER key on the PC keyboard.

The Interface screen allows the user to transfer the memory and programs as well as se or interrogate the various user options.

It also has the ability to upgrade the firmware inside the PIC microcontroller on the EDUC-8 board should this become necessary.

To send data to the EDUC-8 board, use the Write button.

To get data from the EDUC-8 board, use the Read button.

Interface Functions

Note: “Project” refers to the EDUC-8 hardware project.
“Emulator” refers to the EDUC-8 PC emulator.
“Storage” refers to the external memory chip on the EDUC-8 project board

Program Transfers

Memory

Operating Options

Upgrade Driver

COM Port

Reset

Clear

Problems

Program Transfers

Programs can be transferred to and from the PC and project

Directory

This option will list or erase the programs currently stored in the project memory.

Read Lists the stored programs in the view window.
8 programs are for EDUC-8 mode, and 3 are for WIF mode.
The options that are stored with each program are also listed.

```
No  Name                    Pgm   Typ ZPM SED IO

 1  EDUC8 Kit Display       Valid ED8 OFF OFF T13
 2  No Name                 Empty ED8 OFF OFF T13
 3  No Name                 Empty ED8 OFF OFF T13
 4  No Name                 Empty ED8 OFF OFF T13
 5  No Name                 Empty ED8 OFF OFF T13
 6  No Name                 Empty ED8 OFF OFF T13
 7  No Name                 Empty ED8 OFF OFF T13
 8  No Name                 Empty ED8 OFF OFF T13
 1  WIF Kit Display         Valid WIF --- --- AO
 2  No Name                 Empty WIF --- --- AO
 3  No Name                 Empty WIF --- --- AO
```

Erase Opens a window to select the programs to delete.

Programs

Select EDUC8 or WIF program type for access.

Read — Transfers to selected program to the PC.
You can then choose to view the program as disassembled text, transfer it to the PC memory, or save it to a file as disassembled text.
If the PC operating mode does not match the stored program, Ie. Normal or WIF mode, then you will be prompted to change modes.

Write — Transfers the PC program memory to the project memory storage.
You will be prompted for a program name before the transfer starts.
Program options will also be transferred including, the program type, ZPM mode, SED mode and the T1/T13 (BO/AO) reset mode.

Erase — Delete the selected program.

Memory **Write/Read**

This option allows access to the project memory.

The Write process sends either the current PC Emulator memory to the Project memory, or you can fill the Project memory with a value (0 – 255).

The Read process reads the project memory into a PC buffer.

When the read process finishes you will have four options for the data.

List it disassembled on screen
Send it to the PC Emulator memory
Save it as a disassembled text file
Discard it.

During the Read/Write process, the project operating options are synchronised with the EDUC-8 options so that Normal or WIF operating modes match the transferred program.

The Erase process will clear all memory to zero.

Operating Options **Write/Read**

This option accesses the Project operating options.

Normal / WIF mode

Select the operational mode of the project.

Output reset cycle

The EDUC-8 can send an output device reset pulse on either the T1 or T13 cycle. This option tells the PIC code which of these cycles the reset should occur.

Zero Page Mode – EDUC-8 only

A hardware modification was done to allow Zero page mode addressing.
This option will enable or disable that mode of operation.
This option is not available for WIF mode.

See Zero Page explanation

Slow Examine/Deposit Mode – EDUC-8 only

In normal EDUC-8 operation, the computer will start Run mode if either the Deposit or Examine switches are initiated when Slow running is selected. This can be annoying at times so you can enable Fetch or Examine in slow mode by enabling this item.

This option is not available for WIF mode.

Slow speed cycle time

This options give 4 speed choices for the Project slow cycle timing.

125mS
250mS
500mS (Default)
1 second

Clear memory to 0 on reset

This options will clear memory registers to zero on a reset.

Note: If the Zero Page Mode and or the T1/T13 (BO/AO) flags are changed, the current program may not work as expected.

Upgrade Driver **Write**

This option will reprogram the PIC's operating code if the firmware needs to be upgraded.

This updated code will be made available from the project author from the EDUC-8 Emulator website if required.

This HEX code file should be downloaded and stored in the same directory as the EDUC-8 Emulator.

When this option is activated, the HEX file will be loaded, checked and then it will be transferred to the PIC so that it can reprogram itself.

It is important not to interrupt this process once it has started or the PIC may not function properly after a reset.

It is possible that the reprogramming process gets interrupted for a variety of reasons and if this happens the PIC will need to be reprogrammed from a dedicated PIC programmer such as a PICkit3 or similar.

The circuit board for the EDUC-8 project has a port that will allow ICSP connection to a PICkit3 programmer.

COM Port **[Enter]**

Enter a Serial COM Port value.

This value must be in the range 1 to 256.

Press ENTER to change the COM Port.

Press ESC to exit this mode without changing the port.

Reset

This button will force a full reset of the PIC Microcontroller.
The PC will then try to re-establish communications with the EDUC-8 board.

Clear

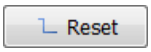
This button will clear the contents of the list window.

Problems

A situation could arise that the PIC firmware or reprogramming has failed for some reason and will not connect properly to the PC via the USB port. If this happens, you may not be able to reprogram the PIC firmware by using the Upgrade Driver option.

Even though the firmware has failed, it is still possible that the internal reprogramming code section, called the “Bootloader”, is still intact.

To see if this is still active, follow the procedure below.

1. Make sure the EDUC-8 is connected to the PC USB port
2. Make sure the EDUC-8 PC Emulator program is running
3. Open the USB Interface Window
4. Verify that a “Communications Failed” message appears
5. Hold the project [Load Address] switch [DOWN]
6. Press and release the [Reset] button from the Interface Window 

7. Release the project [Load Address] switch
8. The following message will appear if the Bootloader is active

```
**********************************************
The EDUC-8 board is operating in Forced Boot Mode
The only functions that will work is to reprogram
the EDUC-8 hardware driver and to do a Reset
**********************************************
```

9. Select [Upgrade Driver] and press the [Write] button to try and reprogram the PIC firmware

If this procedure fails, then a PIC programmer will be required to reprogram the firmware. See ICSP.

PIC SERIAL COMMINCATIONS PROTOCOL

Serial setup: 19200 BAUD 8N1
Data bytes expressed as HEXADECIMAL, Eg $FF
"PIC -> Loop" means PIC will resume its normal code execution
Pgm# means a stored program number in the range 0 - 31

PIC RESET – DTR controls PIC MCLR pin

```
RST1   DTR LOW
RST2   10mS Delay
RST3   DTR HIGH
```

TEST FOR PIC CONNECTION

```
       PC SENDS            PIC RETURNS
TPC1   $A0                 $ED + $E8
```

PROGRAM INTERNAL ROM

Current HEX file loaded, 2 bytes per ROM word. (High Low format is not important)
Data start address = $0x00300 (bytes), any lower will cause a program crash
Data end address = end of ROM data in HEX file.
Ignore EEPROM, ID and CONFIG data from HEX file
Data is padded with $FFFF for total word size (mod 64 = 0)
PIC program buffer is 128 bytes (64 words)
The boot loader program initialises itself ready to receive ROM data

```
       PC SENDS            PIC RETURNS
PIR1   $A1                 $A1    / Jump to boot loader
PIR2   $02, Data+0, Data+1 $02    / Transfer next 2 data words (4 bytes)
            Data+2, Data+3         / Data format is LOW byte HIGH byte
       Continue PIR2 until 64 words (128 bytes) have been sent
PIR3   $03                        / Initiate PIC ROM write
                           $F3    / Internal write not required (same data in ROM)
                           $03    / Internal write completed
       If more ROM data, continue from PIR2
       If no more ROM data, do a PIC MCLR reset to start new code


READ or WRITE Memory - EDUC-8 = up to 256 bytes
                       WIF    = up to 1024 Words HL (2048 bytes) (2 words per xfer)
```

WRITE EDUC-8 MEMORY

```
       PC SENDS            PIC RETURNS
WEM1   $A2                 $A2                 / Initiate EDUC-8 memory write
WEM2   If next byte available
       EDUC-8 Mode
         Byte             $A4                 / while data available
       WIF Mode
         H0 L0 H1 L1       $A4                 / 2 words) while data available
       Continue from WEM2
       Else
       $56                 $56                 / PIC -> Loop
```

READ EDUC-8 MEMORY

Storage is required for 256 bytes

```
       PC SENDS            PIC RETURNS
REM1   $A3                 $A3                 / Initiate EDUC-8 memory read
       EDUC-8 Mode
REM2   $52                 $52, B0, B1, B2, B3 / Read next 4 bytes
       If data count < 256 bytes then continue from REM2
       WIF MODE
REM2   $52                 $52, H0, L0, H2, L2 / Read next 2 words
       If data count < 1024 words then continue from REM2
REM3   $56                 $56                 / PIC -> Loop
```

ERASE EDUC-8 MEMORY

```
       PC SENDS            PIC RETURNS
ERM1   $A4                 $A4                 / Initiate EDUC-8 memory erase
```

```
ERM2   $80                $56                / PIC -> Loop
```

EDUC-8 OPTIONS – 1 BYTE

Operating Mode	bit 0	1 = Normal, 0 = WIF
Spare	bit 1	
Zero Page Mode	bit 2	1=OFF 0=ON
Allow slow exam/deposit	bit 3	1=NO 0=YES
Slow Speed	bits 5 4	00=125mS 01=250mS 10=500mS 11=1sec
T1 T13 Reset	bit 6	1=T13 0=T1
Reset memory to 0 on reset	bit 7	1=No, 0=Yes

WRITE OPERATING MODE

```
       PC SENDS           PIC RETURNS
WIE1   $A5                $A5                / Options access
WIE2   $53, Options       $53                / PIC -> Loop
```

READ OPERATING MODE

```
       PC SENDS           PIC RETURNS
RIE1   $A5                $A5                / Options access
RIE2   $52                $52 Options        / PIC -> Loop
```

PROGRAM STORAGE

EDUC-8	Program Numbers (PN) = 0 – 7	ID = $00	256 bytes each
WIF	Program Numbers (PN) = 0 – 2	ID = $FF	2048 bytes each

Program Options	**Bit**	**Type**	**Value**
	0	Status	Empty = 1 Valid = 0
	1	ZPM mode	NORM = 1 ZPM = 0
	2	SED mode	NORM = 1 SED = 0
	3	Reset mode	T13 (AO) = 1 T1 (BO) = 0

```
  Program Name (NM) –      Data[1]       = Program Options
                           Data [2..24]  = 23 name characters

  Program Storage Address, only High Byte required, Low Byte = 0
                           - EDUC-8  PSAH = EDUC-8 PN
                           - WIF     PSAH = 8 + (WIF PN x 8))
```

READ PROGRAM NAME

```
       PC SENDS           PIC RETURNS
RPD1   $A6                $A6                / Name access
RPD2   $54, PN, ID        $54
RPD3   $52                NM[1..4]           / next 4 characters in name
       If < 24 characters received, continue RPD3
RPD4   $56                $56                / PIC -> Loop
       Program Options in String[1], Program Name is String[2..24]
```

WRITE PROGRAM NAME

```
       PC SENDS           PIC RETURNS
WPN1   $A6                $A6                / Name access
WPN2   $54, PN, ID        $54
WPN2   $53, NM[I]         $53
       If I < 24 characters, continue WPN2, else
WPN3   $56                $56                / PIC -> Loop
```

READ STORED PROGRAM

```
       PC SENDS           PIC RETURNS
RPG1   $A8                $A8                / Read EEPROM access
RPG2   $50, PSAH          $50                / program address
RPG3   $52                $52 + 4 Bytes      / next 4 bytes from memory
       For EDUC-8 – read 256 bytes, WIF – read 2048 bytes
       else
RPG4   $56                $56                / PIC -> Loop
```

WRITE STORED PROGRAM

EEPROM write page size is 32 bytes

```
       PC SENDS            PIC RETURNS
WPG1   $A7                 $A7                / Write EEPROM access
WPG2   $54                 $51                / initialise write buffer
WPG3   $51, Next 4 bytes   $51                / transfer next 4 program bytes
       Repeat WPG3 until 32 bytes sent
WPG4   $53                 $53                / write, returned OK
                           $58                / write, returned FAIL, PIC -> Loop
       If OK and more bytes to be sent, repeat from WPG2
       else
WPG5   $56                 $56                / PIC -> Loop
```

ERASE STORED PROGRAM

```
       PGN = Program Number: EDUC-8 programs 0 - 7
                             WIF    programs 0 - 2

       PC SENDS            PIC RETURNS
EPG1   $A9                 $A9                / Write EEPROM access
EPG2   $54                 $53                / initialise write buffer
EPG3   if EDUC-8
         $53 PGN $00       $53
       If WIF
         $53 PGN $FF       $53
EPG5   $56                 $56                / PIC -> Loop
```

Reprogramming the PIC via ICSP.

Caution: The components used in this project may be damaged by static electricity. Observe proper handling procedures.

Prior to ICSP programming, if the PIC chip has code from another project, erase the chip before ICSP or the code may run and damage then PIC chip and or EDUC-8 circuit.

Remove any power sources and disconnect the USB cable if fitted.

The two links in connector CN3 should be moved so that they connect pins (1 and 2) and pins (4 and 5).

Connector CN2 is used for In Circuit Serial Programming (ICSP) for the PIC chip, using a Microchip® PICkit™ 3 programmer.

Pin 1 of the PICkit3 connects to the connector as shown.

If you are using Microchip's IPE interface with the programmer, then you need to configure it first. Click the [Power] button and make sure the [Power Target Circuit from Tool] item is checked.

This mode requires all operational EDUC-8 LEDs to be off otherwise it may not be able to supply enough power.

To set this up, `see clear memory on reset` - Operating Options

Click the [Operate] button. For the "Source" file, click [Browse] and navigate to the EDUC-8 installation/PIC directory and select "educPIC.hex".

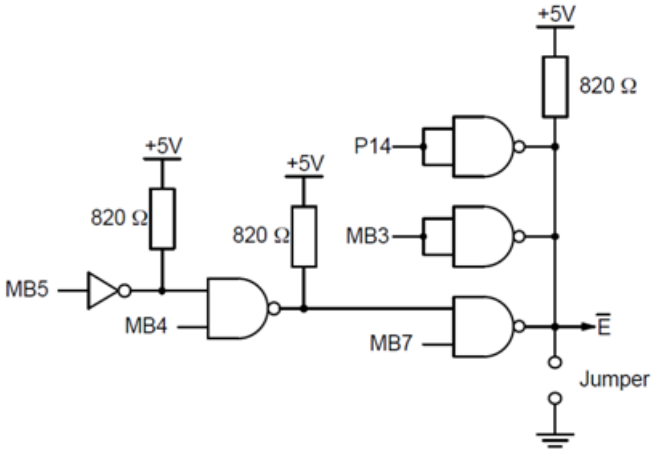
Click [Connect] and follow the prompts.

Click the [Program] button and when PIC programming is completed, remove the PICkit3 and replace the links in CN3 so that they connect pins (2 and 3) and (5 and 6).

Zero Page Mode

The zero page mode of operation was made available by adding a modification to the original EDUC-8 circuit.

The idea is that you can have common variables located in the first 8 bytes of page 0 which can be accessed from all memory pages. That way you do not have to waste memory by adding code to access to them across page boundaries.



The above circuit addition is representative of the following logic function.

```
F.T23.MB3=0.(AND TAD ISZ DCA JMS.MB4)
```

For zero page access to work:

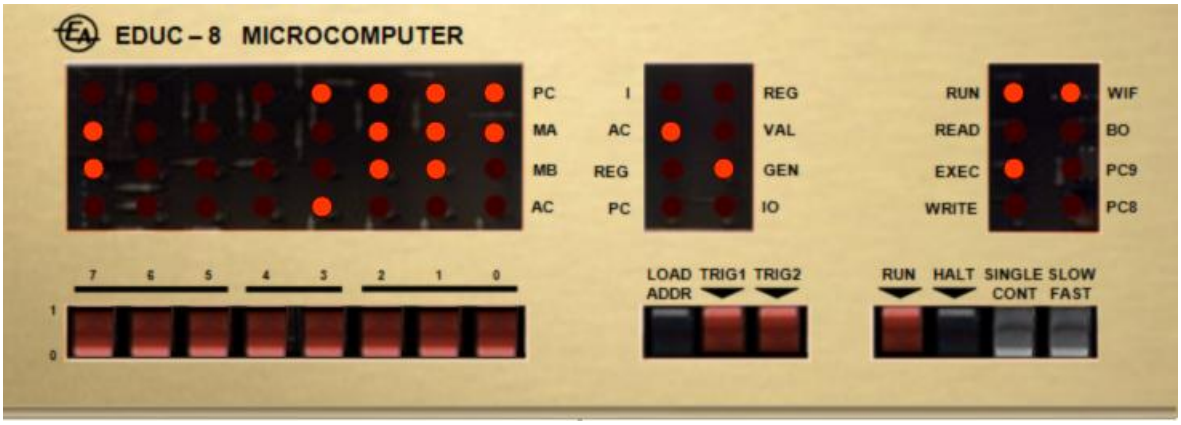
- the condition is sampled on cycle T23
- bit 3 of the address nibble must be zero, (ie. Only lower 8 addresses)
- the instruction has to be AND TAD ISZ DCA or JMS
- if JMS then bit 4 of the instruction must be 1 (ie Indirect)
- the zero page option must be enabled

The assembler file called `EDUC8 PrinterCodes.asm` is an example of using the zero page mode of operation and is available in the installation directory.

The original file with further details can be found at this link.

http://www.sworld.com.au/steven/educ-8/educ8_page_zero.pdf

WIF Mode



What IF mode is an attempt to say:

> “Back then, What IF the EDUC-8 had...”

This mode moderately beefs up the original design to increase its performance and versatility and works with the PC Emulator and the PIC project.

Basic Changes

Memory Size	1K ROM	16 bit
RAM Size	64 bytes	8 bit
EEPROM Size	32 bytes	8 bit
Data paths	Parallel, except for Input / Output.	
Instructions	55	
Indirect Addressing	Yes	
Subroutine Nesting	3 level return stack	

WIF Mode is enabled by checking the [What IF Mode] item from the Options menu.

WIF Front Panel

LED	Function
PC	These indicate the lower 8 bits of the Program Counter.
MA	These are used for instruction data processing
MB	These are used for instruction data processing
AC	These indicate the status of the Accumulator
I	Lit if current instruction accesses Indirect RAM register
AC	Lit if current instruction accesses AC
REG	Lit if current instruction accesses a RAM register
PC	Lit if current instruction accesses the PC
REG	Lit if current instruction is a Register type
VAL	Lit if current instruction is a Value type
GEN	Lit if current instruction is a General type
IO	Lit if current instruction is an Input/Output type
RUN	Program Run indicator
READ	Code execution Read cycle
EXEC	Code execution Exec cycle
WRITE	Code Execution Write cycle
WIF	WIF mode indicator
AO/BO	Output reset pulse indication – AO (After output), BO (Before output)
PC9	Bit 9 of PC address
PC8	Bit 8 of PC address

Switch Functions

The front panel switches work similar to the original EDUC-8.

In WIF mode, the Deposit and Exam switches become TRIG1 and TRIG2.

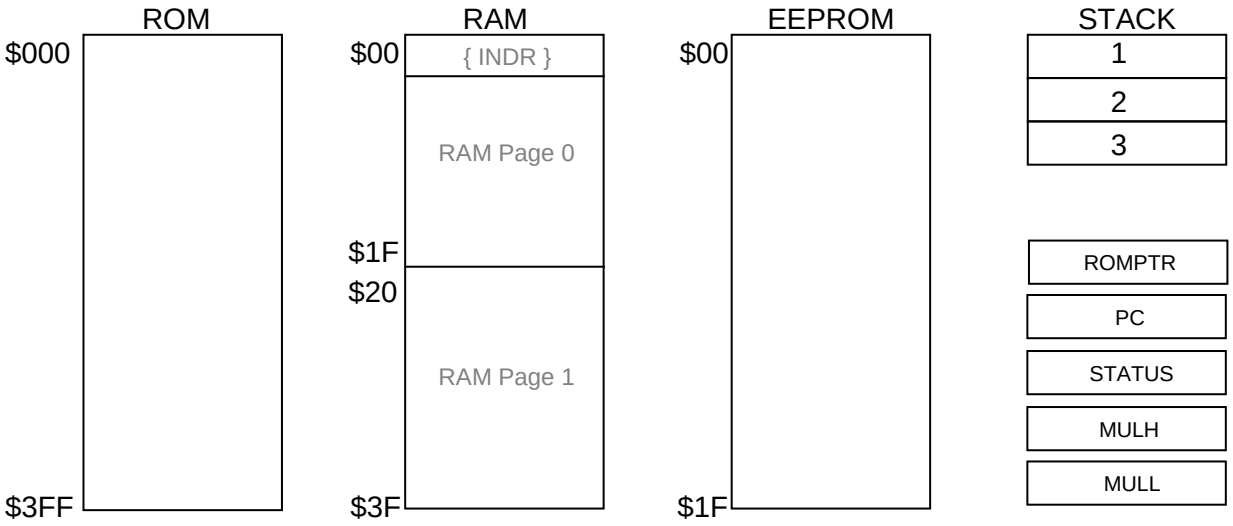
The Load Address range is similar to the EDUC-8, meaning that only ROM addresses 0 – 255 can be set. This means all programs should start somewhere between ROM address $000 and $0FF. It is most likely that WIF programs will begin from ROM address $000 so this shouldn't be a problem.

All modules with the exceptions of the Tape Read, Tape Write and the Magnetic Tape Storage, can be connected to the WIF computer.

There are additional windows available to view RAM and EEPROM contents and you can edit these values by double clicking any row within the grids.

There are also some code examples included in the install directory with the prefix " WIF_ ".

WIF Memory Map



The ROM space consists of 1024 (1K) 16 bit words.
The full ROM space is accessible to all ROM address instructions except for PC as [TGT].

The general memory space consists of 64 bytes of RAM arranged in two banks or 32 bytes. These banks are accessed by setting the `RP` bit in the `STATUS` register.

`INDR` (RAM $00) is used for indirect memory addressing and general purpose RAM.

`MULH` and `MULL` are used to store the result of the 8 bit multiply instruction, `MULTA`.

`MULH` holds the upper 8 bit result.
`MULL` holds the lower 8 bit result.

There are 32 EEPROM bytes for persistent storage and are accessible via the AC register.

The `STATUS` register has 4 active bits - Z, C, DC and RP.

Z is a Zero bit and is set to 1 when the relevant data is zero, or cleared for non zero.

C is a Carry bit and is set or cleared after adding or subtracting values. It can also be manually set or cleared.

DC is a Digit Carry bit and is set or cleared after adding or subtracting values. It is only concerned with a carry from the lower nibble to the upper nibble. It can also be manually set or cleared.

RP is the RAM Page bit. 0 = Page zero access, 1 = Page one access.

The 10 bit Program Counter (PC) covers the full ROM space and can be modified by code.

There are three stack registers that allow for two levels of subroutine nesting.

The GOSUB instruction pushes the current PC + 1 onto the stack.

The RETURN and RETVAL instructions pop the stack into the PC.

WIF Assembler

Assembler Directives

_ISWIMODEON	Alerts the user to set the correct WIF/Normal
_ISWIMODEOFF	mode of operation when assembling code.
_SETPC	Set PC address - ROM range is $000 - $3FF
_ISRESETBO	These directives are same as the T1 T13 flag
_ISRESETAO	for normal operation
_RAM	Specifies an area to define RAM variables
_ENDRAM	End of RAM definitions
_EEP	Specifies an area to define EEPROM variables
_ENDEEP	End of EEPROM definitions
_CONST	Define a constant

The _RAM directive allows entry of up to 64 RAM variables.
The _ENDRAM directive must terminate the end of the RAM variables.

<NAME> <Address (Optional)> <Size (Optional)>

Examples:

```
      _RAM
RAM0                / defines register called RAM0 at address 0
RAM1                / defines register called RAM1 at address 1
Flags #$05          / defines register called Flags at address 5
                    / address pointer will increment from 5 if no other
                    / address directives are found
Buffer :#D10        / defines a 10 byte buffer area starting at $06
Store #$1A :#$05    / defines a 5 byte buffer area starting at $1A
Count               / define count, at address $1F
      _ENDRAM
```

The _EEP directive allows entry of up to 32 EEPROM variables.
The _ENDEEP directive must terminate the end of the EEPROM variables.

<NAME>

Example:

```
      _EEP
CountH       / defines register called CountH at address 0
CountL       / defines register called CountL at address 1
      _ENDEEP
```

A constant can be defined anywhere in the assembler file and non 2's compliment numbers can only be defined as a number between 0 and 1023 ($0000 - $FFFF). See also - Default Constants.

Examples:

```
            _CONST InitVal #$FF
            _CONST ROMaddr #$1002
            _CONST Buffer              / equivalent to RAM #$06 (above)
            _CONST Buffer + #D1        / equivalent to RAM #$07 (above)
```

WIF Instruction Set Summary

Instruction		Code	Flags Affected
NOP	No Operation	0000 00xx xxxx xxxx	
ADDVA	Add value to AC	0001 00xx vvvv vvvv	Z C DC
ANDVA	AND value with AC	0010 00xx vvvv vvvv	Z
IORVA	Inclusive OR value with AC	0011 00xx vvvv vvvv	Z
MOVVA	Move value into AC	0100 00xx vvvv vvvv	
SUBVA	Subtract value from AC	0101 00xx vvvv vvvv	Z C DC
XORVA	Exclusive OR value with AC	0110 00xx vvvv vvvv	Z
SKIPIFA	Test AC, skip if various functions	0111 00md vvvv zzzz	
RDSWA	Read SR<7:0> switches into AC	1000 00xx xxxx xxxx	Z
DAAC	Decimal adjust AC	1001 00xx xxxx xxxx	C
MULTA	Multiply AC with value [BY] [HA] [LA]	1010 00xx vvvv vvvv	
RANDA	Random [0 - 255] into AC	1011 xxxx xxxx xxxx	Z
ADDR	Add to register	0000 01ss tter rrrr	Z C DC
ANDR	AND with register	0001 01ss tter rrrr	Z
CLRR	Clear register	0010 01ss ttxr rrrr	Z
DECR	Decrement register	0011 01ss ttxr rrrr	Z
DECRSZ	Decrement register, Skip if 0	0100 01ss ttxr rrrr	
INCR	Increment register	0101 01ss ttxr rrrr	Z
INCRSZ	Increment register, Skip if 0	0110 01ss ttxr rrrr	
IORR	Inclusive register	0111 01ss tter rrrr	Z
MOVR	Move register	1000 01ss ttxr rrrr	Z
ROTL	Rotate Left register through Carry	1001 01ss ttxr rrrr	C
ROTR	Rotate Right register through Carry	1010 01ss ttxr rrrr	C
SUBR	Subtract register	1011 01ss tter rrrr	Z C DC
XORR	Exclusive register	1100 01ss tter rrrr	Z
EXCHR	Exchange register nibbles	1101 01ss ttxr rrrr	
COMPR	Compliment register	1110 01ss ttxr rrrr	Z
GOSUB	Go to subroutine	0000 10aa aaaa aaaa	
GOTO	Go to address	0001 10aa aaaa aaaa	
RETURN	Subroutine return	0010 10xx xxxx xxxx	
RETVAL	Subroutine return with value in AC	0011 10xx vvvv vvvv	
HALT	Stop program execution	0100 10xx xxxx xxxx	
SKIPSB	Skip if STATUS bit Set/Clear	0101 10xx xxxx mzdc	
SETSTB	Set/Clear STATUS flags	0110 10xx xxxx mzdc	
SETB	Set register bit	0111 10ss bbbr rrrr	
CLRB	Clear register bit	1000 10ss bbbr rrrr	
TOGLB	Toggle register bit	1001 10ss bbbr rrrr	
SKIPBS	Skip if register bit set	1010 10ss bbbr rrrr	
SKIPBC	Skip if register bit clear	1011 10ss bbbr rrrr	
ROMADR	Set ROM access address [H L] [Addr]	1100 10xb vvvv vvvv	
SKIPIF	Skip on input flag	0000 11xx xxvx xxxx	
READID	Read input device	0001 11xx ttvr rrrr	Z
RESETI	Reset input flag	0010 11xx xxvx xxxx	
RDRSTI	Read input device and reset flag	0011 11xx ttvr rrrr	Z
SKIPOF	Skip on output flag	0100 11xx xxvx xxxx	
WRITOD	Load output device	0101 11ss xxvr rrrr	
RESETO	Reset output flag	0110 11xx xxvx xxxx	
WTRSTO	Load output device and reset flag	0111 11ss xxvr rrrr	
SEROUT	AC -> serial port	1000 11xx xxxx xxxx	
SKIPNS	Skip if no serial IN -> AC	1001 11xx xxxx xxxx	Z
SERACC	Serial port access [PC] [PIC]	1010 11xx xxxx xxxp	
ROMACC	ROM access [H L] [R W] ([I] Optional)	1011 11xx xxii iiab	
SKIPTS	Skip on trigger switch [1 2] [H L]	1100 11xx xxxx xxts	
TIMER	Timer Function	1101 11xf vvvv vvvv	
EEPROM	EEPROM Access [0-31] [R] [W] [SKIPWT]	1110 11xs arwe eeee	Z (Read)

x	Don't care
p	Serial port access
v	Input Output device
b	High or Low bit
i	Increment
a	Read / Write access
t	Trigger mode
s	Trigger switch
f	TIMER function
m	STATUS Set/Clear
z	STATUS Z
d	STATUS DC
c	STATUS C
e	Result Destination
r	Mulresa H L

ff	Indr function
tt	Target register [TGT]
ss	Source register [SRC]
bbb	Bit number
zzzz zzzz	Instruction function
r rrrr	RAM address
vvvv vvvv	Value
pppp pppp	Program address H L
aa aaaa aaaa	ROM address
Z	Zero flag
C	Carry flag
md	SKIPA mode
iiii	ROMACC I functions
arwe eeee	EEPROM access, address

WIF Instruction Set – Group 00

NOP — No Operation — Flags affected: None

```
NOP
```

Example:

```
NOP
```

ADDVA — Add value to AC — Flags affected: Z C DC

```
ADDVA [VALUE]
```

Example:

```
ADDVA #$01
AC + #$01 -> AC
```

ANDVA — AND value with AC — Flags affected: Z

```
ANDVA [VALUE]
```

Example:

```
ANDVA CONST_VAL
AC AND #$01 -> AC
```

```
CONST_VAL = #$01
```

IORVA — Inclusive OR value with AC — Flags affected: Z

```
IORVA [VALUE]
```

Example:

```
IORVA #$01
AC OR #$01 -> AC
```

MOVVA — Move value into AC — Flags affected: None

```
MOVVA [VALUE]
```

Example:

```
MOVVA RAM_COUNT
#$00 -> AC
```

RAM_COUNT = RAM 0

SUBVA — Subtract value from AC — Flags affected: Z C DC

```
SUBVA [VALUE] from AC
```

Example:

```
SUBVA #$01
AC - #$01 -> AC
```

XORVA — Exclusive OR value with AC — Flags affected: Z

```
XORVA [VALUE]
```

Example:

```
XORVA #$01
AC XOR #$01 -> AC
```

SKIPIFA — Test AC, skip if function succeeds — Flags affected: None

```
SKIPIFA [POS NEG]
        [<, =, >= (Val)]
```

Examples:

```
SKIPIFA POS      IF 2'S COMP AC = +VE, PC = PC + 1
SKIPIFA >= 5     IF A >= 5,            PC = PC + 1
SKIPIFA = #$45   IF A = $45,           PC = PC + 1
```

RDSWA — Read SR<7:0> switches -> AC — Flags affected: Z

Example:

```
RDSWA
SR<7:0> -> AC
```

DAAC — Decimal adjust AC — Flags affected C

```
DAAC
```

DACC adjusts the 8-bit value in AC, which has the result from an earlier addition of two variables, which were in packed BCD format, and produces a correct packed BCD result.

Example:

```
MOVVA #$48     / packed decimal 4 8 in BCD format
ADDVA #$23     / packed decimal 2 3 in BCD format
AC =   $6B     / Hexadecimal result in AC
DAAC
AC =   $71     / packed decimal 7 1 in BCD format
```

MULTA — Multiply AC with value — Flags affected None

```
MULTA [Func] (VAL)
```

Example:

```
MULTA #$05     / when AC = $34
               / MULH = $01   MULL = $04
```

Example:

```
MULTA HA       / MULTA H result to AC
MULTA LA       / MULTA L result to AC
```

RANDA — Random [0 – 255] into AC — Flags affected Z

```
RANDA
```

Example:

```
RANDA          / Psuedo random value -> AC
```

Note: Randomness will diminish if the RANDA instruction is used in quick succession.

WIF Instruction Set – Group 01

ADDR ADD register
Example:
```
ADDR [SRC] [TGT] [RES]
ADDR 3 A S
Register 3 + AC -> Register 3
```
Flags affected: Z C DC

ANDR AND register
Example:
```
ANDR [SRC] [TGT] [RES]
ANDR A #$01 T
AC AND Register #$01 -> Register #$01
```
Flags affected: Z

CLRR Clear register
Example:
```
CLRR [SRC] [TGT]
CLRR #$01 #$01
Register #$01 = 0 -> Register #$01
```
Flags affected: Z

DECR Decrement register
Example:
```
DECR [SRC] [TGT]
DECR #$01 P
Register #$01 - 1 -> PC
```
Flags affected: Z

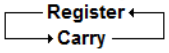
DECRSZ Decrement register, Skip if 0
Example:
```
DECSZ [SRC] [TGT]
DECSZ A A
AC - 1 -> AC (PC + 1 IF = 0)
```
Flags affected: None

INCR Increment register
Example:
```
INCR [SRC] [TGT]
INCR I I
Register (I) + 1 -> Register (I)
```
Flags affected: Z

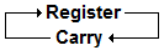
INCRSZ Increment register, Skip if 0
Example:
```
INCSZ [SRC] [TGT]
INCSZ A A
AC + 1 -> AC (PC + 1 IF = 0)
```
Flags affected: None

IORR Inclusive OR register
Example:
```
IORR [SRC] [TGT] [RES]
IORR P A S
PC OR W -> PC
```
Flags affected: Z

MOVR Move register
Example:
```
MOVR [SRC] [TGT]
       MOVR #$01 I
Register #$01 -> Register (I)
```
Flags affected: Z

ROTL Rotate Left register through Carry
Example:
```
ROTL [SRC] [TGT]
ROTL #$01 A
Rotate register #$01 left -> AC
```


Flags affected: C

ROTR Rotate Right register through Carry
Example:
```
ROTR [SRC] [TGT]
ROTR #$01 A
Rotate register #$01 right -> AC
```


Flags affected: C

SUBR Subtract register
Example:
```
SUBR [SRC] from [TGT] [RES]
SUBR P A T
AC - PC -> A
```
Flags affected: Z C DC

XORR Exclusive OR register
Example:
```
XORR [SRC] [TGT] [RES]
XORR I #D4 S
Register INDIRECT XOR Register 4 -> Register INDIRECT
```
Flags affected: Z

EXCHR Exchange register nibbles
Example:
```
EXCHR [SRC] [TGT]
EXCHR COUNT A
Register COUNT exchange nibbles -> AC
COUNT = $A5, after instruction, COUNT = $A5, AC = $5A
```
Flags affected: None

COMPR Compliment register
Example:
```
COMPR [SRC] [TGT]
COMR COUNT A
Compliment register COUNT -> AC
If COUNT = $AA, AC = #55
```
Flags affected: Z

WIF Instruction Set – Group 10

GOSUB	Go to subroutine
	Example:
		`GOSUB [ADDRESS]`
		`GOSUB #$154`
		`PC + 1 -> Stack`
			Flags affected:	None

GOTO	Go to address Example:	`GOTO [ADDRESS]` `GOTO LABELB`	Flags affected:	None

RETURN	Subroutine return
	Example:
		`RETURN`
		`RETURN`
		`Stack -> PC`
			Flags affected:	None

RETVAL	Subroutine return with value in AC
	Example:
		`RETVAL [VALUE]`
		`RETVAL #D34`
		`#D34 -> AC, Stack -> PC`
			Flags affected:	None

HALT	Stop program execution Example:	`HALT` `HALT`	Flags affected:	None

STSKIP	Skip if STATUS bit set/clear
	Example:
		`STSKIP [C DC Z] [SET CLR]`
		`STSKIP C SET`
		`If Carry = Set, PC + 1 -> PC`
			Flags affected:	Z C DC

STFLAG	Set/Clear STATUS flags
	Example:
		`STFLAG [C DC Z] [SET CLR]`
		`STFLAG DC CLR`
		`DC Flag = Clear`
			Flags affected:	Z C DC

SETB	Set register bit
	Examples:
		`SETB [A I P] [BIT]`
		`SETB #$03 7  1 -> RAM $03 bit 7`
		`SETB I 6  1 -> RAM(I) bit 6`
			Flags affected:	None

CLRB	Clear register bit
	Examples:
		`CLRB [A I P] [BIT]`
		`CLRB P 4  0 -> PC bit 4`
		`CLRB A 0  0 -> AC bit 0`
			Flags affected:	None

TOGLB	Toggle register bit
	`Examples:`
		`TOGB [A I P] [BIT]`
		`AC = #B11111111`
		`TOGLB A 4  Toggle AC bit 4  AC = #B11101111`
		`TOGLB A 4  Toggle AC bit 4  AC = #B11111111`
			Flags affected:	None

SKIPBS	Skip if register bit set
	Example:
		`SKIPBS [A I P] [BIT]`
		`SKIPBS A 7`
		`If AC bit 7 = 1 then PC + 1 -> PC`
			Flags affected:	None

SKIPBC	Skip if register bit clear
	Example:
		`SKIPBC [A I P] [BIT]`
		`SKIPBS I 7`
		`If RAM(I) bit 7 = 1 then PC + 1 -> PC`
			Flags affected:	None

ROMPTR	Set ROM pointer register
	Example:
		`ROMPTR [H L] [Address] [Increment]`
		`ROMPTR H #$01`
		`ROMPTR L #$00`
		`ROMPTR register = $100`
			Flags affected:	None

WIF Instruction Set – Group 11

SKIPIF Skip on input flag
Example:

```
SKIPIF  [DEV]                                   Flags affected:   None
SKIPIF 0
If Input flag Device 0 = LO, PC + 1 -> PC
```

READID Read input device
Example:

```
READID  [DEV] [A R I P]                         Flags affected:   Z
READID 0 AC
Input Device 1 data shifted into AC
```

RESETI Reset input flag
Example:

```
RESETI  [DEV]                                   Flags affected:   None
RESETI 1
Input Device 1, Input Flag = HI
```

RDRSTI Read input device and reset flag
Example:

```
RDRSTI  [DEV] [A R I P]                         Flags affected:   Z
RDRSTI 0 I
Input Device 0 data shifted into RAM(I)
Input Device 0, Input Flag = HI
```

SKIPOF Skip on output flag
Example:

```
SKIPOF  [DEV]                                   Flags affected:   None
SKIPOF 1
If Output flag Device 0 = LO, PC + 1 -> PC
```

WRITOD Write output device
Example:

```
WRITOD  [DEV] [A R I P]                         Flags affected:   None
WRITOD 1 A
Output Device 1 data shifted into AC
```

RESETO Reset output flag
Example:

```
RESETO  [DEV]                                   Flags affected:   None
RESETO 0
Output Device 0, Input Flag = HI
```

WTRSTO Write output device and reset flag
Example:

```
WTRSTO  [DEV]   [A R I P]                       Flags affected:   None
WTRSTO 0 #D2
Output Device 0 data shifted from RAM #D2
Output Device 0, Input Flag = HI
```

SEROUT AC -> serial port
Example:

```
SEROUT                                          Flags affected:   None
SEROUT
AC -> Serial Port
```

SKIPNS Skip if no serial IN -> AC
Example

```
SKIPNS                                          Flags affected:   Z
SKIPNS
If no data received to serial port, execute next
If data received, -> AC, skip next instruction
```

SERACC Serial port access
Example:

```
SERACC  [ON]  [OFF]                             Flags affected:   None
SERACC ON
Serial Port Access is turned ON
```

ROMACC Access program ROM
Examples:

```
ROMACC  [R W] [H L] ([I] Optional)              Flags affected:   None
ROMACC W H I+
AC -> ROM[ROMPTR][High Byte] and post increment ROMPTR

ROMACC R L
ROM[ROMPTR][Low Byte] -> AC and no Inc or Dec ROMPTR
```

SKIPTS Skip on Trigger Switch
Examples:

```
SKIPTS  [1 2]  [H L]                            Flags affected:   None
SKIPTS 1 H
If Trigger switch 1 = H, PC + 1 -> PC
SKIPTS 2 L
If Trigger switch 2 = L, PC + 1 -> PC
See IO Port Interfaces
```

TIMER Timer Function
Examples:

```
TIMER [ON][OFF] [SKIPOF] [(1-255)]              Flags affected:   None
TIMER #D20      Timer Delay = 20 * 10 = 200mS
TIMER SKIPOF    If Timer has overflowed PC = PC + 1
See [WIF Clock.asm] for a code example
```

EEPROM EEPROM Access
Examples:

```
EEPROM  [Address] [R] [W] [SKIPWC]              Flags affected:   Z
EEPROM 0        / set EEPROM address pointer = 0
EEPROM R        / EEPROM ADDRESS 0 -> AC (Pointer + 1)
MOVVA #$FF
EEPROM W        / EEPROM ADDRESS 1 = FF  (Pointer + 1)
WAIT,   EEPROM SKIPWC  / wait until EEPROM write completes
GOTO WAIT
```

[TGT] or [SRC] arguments specify where the data for an instruction comes from, and where it is deposited after the instruction executes. These two items can specify a RAM address, or an address code.

Address Codes:

[A]	Accumulator
[R]	Register
[I]	Indirect
[P]	Program Counter

If the [SRC] of an instruction is the Program Counter, then the lower 8 bits of the PC are read.

If the [TGT] of an instruction is the Program Counter register, the final PC address is,

(ROMPTR high byte * 256) + 8 lower bits from the instruction.

If the target [TGT] of an instruction is to be [R], then the [R] is optional.

Example: `decr Count r` is the same as `decr Count`

ADDR ANDR IORR SUBR and XORR instructions have an additional argument which is used to specify if the instruction result goes to the SRC or the TGT register.

```
ADDR A 5 T          / result goes to TGT (RAM address 5)
XORR A TIMER S      / result goes to SRC (AC)
```

Notes:

RESETO, WTRSTO instructions

Output devices can receive a reset pulse before data is shifted out or after data is shifted out. See T1 T13

Serial Port

The serial port is shared between the PC communications and the WIF mode of operation. If the Serial Access is turned ON, then the EDUC-8 PC USB interface should not be connected or it may receive data and interpret it incorrectly. The PIC's operating code will ignore any serial data when the Serial Access is turned ON.

Serial Access is turned OFF when either of the LA switch is used, when transferring a program to memory, when the PIC is reset, or with the instruction SERACC OFF.

The serial port setting is 19200 baud, 8 data bits, 1 stop bit.

The serial port can receive up to two bytes before reading the input buffer. If more bytes are received when the buffer is full, all but the first byte in the buffer can be read. After reading, the buffer will function normally.

Instruction Execution

Most instructions take 4 cycles to complete and transfer data in a parallel mode.

Cycle 1	Initialize
Cycle 2	Read
Cycle 3	Execute
Cycle 4	Write

The instructions `READID`, `RDRSTI`, `WRITOD` and `WTRSTO` take an extra 8 cycles to execute because of the need to serial in or out the 8 bits of data. In addition, the `WTRSTO` instruction takes an extra 2 cycles because of the requirement to reset the output device before or after data transmission.

HIGH LOW Operands

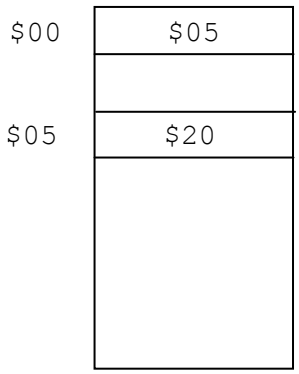
These operands work with most instructions and recover the upper byte (HIGH) or lower byte (LOW) of a value.

Examples:

```
_CONST Count #$1FFF

$1FE: label,      nop               / code is at PC address $1FE
                  movva HIGH label  / AC = #$01
                  movva LOW Count   / AC = #$FF
```

Indirect addressing is invoked when the instruction [TGT] or [SRC] argument is (`I`).

Example:




```
INCR I A    / increment [I] -> AC

Indirect register points to RAM $05
$20 + 1 = $21

AC          = $21
RAM $00     = $05
RAM $05     = $20
```

Indirect addressing can access all bytes in RAM independent of the `STATUS RP` bit.

ROMPTR and ROMACC Instructions

The [ROMPTR] register is a 16 bit register.

The high byte <1:0> is used to compute the PC address if the [TGT] of an instruction is [P].

Example:

```
ROMPTR H #$01       / set ROMPTR = $103
ROMPTR L #$03
```

	High	Low	
[ROMPTR]	1	03	Value = $103 HEX (259 Decimal)

Examples:

AC = $30

```
MOVR A P                    / AC -> PC
```
PC = ROMPTR[H <1:0>] + AC + 1
PC = $131

COUNT = $01

```
INCR COUNT P                / INCR COUNT -> PC
```
$01 + 1 = 2
PC = ROMPTR[H <1:0>] + 2 + 1
PC = $100 + 2 + 1 = $103

COUNT = $FF

```
INCRSZ COUNT P              / INCRSZ COUNT -> PC
```
$FF + 1 = 0, Skip is True, but ignored
PC = ROMPTR[H <1:0>] + Count + 1
PC = $100 + $00 + 1 = $101

The ROMPTR register [High <1:0> - Lo <7:0>] is used as the ROM address for the ROM access instruction (ROMACC).

ROMACC takes either 2 or 3 arguments.

```
W      Write                    R  Read
H      High byte                L  Low byte
Action (Optional)
```

[Action] can have the following types:

```
+I     pre increment ROMPTR     ROMPTR incremented prior to ROMACC execute
I+     post increment ROMPTR    ROMPTR incremented after ROMACC executes
-I     pre decrement ROMPTR     ROMPTR decremented prior to ROMACC execute
I-     post decrement ROMPTR    ROMPTR decremented after ROMACC executes
```

Example:

ROM Memory $103 = $8440 (MOVR A #$00)
ROMPTR = $103

AC = $xx

```
ROMACC R H          / Read ROM High byte -> AC
```
AC = $84

AC = $84

```
ROMACC R L          / Read ROM Low byte -> AC
```
AC = $40

In both cases [Action] was omitted, so ROMPTR stays the same = $103.

Example:

```
       _RAM
Index
       _ENDRAM

$2CF:  Table, movr a p                  / PC address $2CF, instruction = $84C0

              ROMPTR H HIGH Table       / set ROMPTR = $2CF
              ROMPTR L LOW Table

              MOVVA #$85                / value $85
              ROMACC W H                / write into ROM[ROMPTR H]
              MOVVA #$C0                / value $C0
              ROMACC W L I+             / write into ROM[ROMPTR L], Post Increment

$2CF:  Table, movr Index p              / PC address $2CF = new instruction $85C0

              ROMaddr = $2D0
```

Table access can be accomplished by using the [P] argument in an instruction.

Example:

```
$2CF:  Table, movr Index p
              retval #$00
              retval #$01
              retval #$02
              retval #$03
```

As the [P] argument uses the high byte of the ROMPTR register to compute the final ROM address, it must be set to the correct ROM page.

If the above code executes at address $2CF when ROMPTR is set to $0000, the final PC address will be $0D0 instead of the required $2D0.

Example:

```
$2CF:  Table, ROMPTR H High Table / match ROMPTR with [Table] start
              movr Index p
              retval #$00
              retval #$01
              retval #$02
              retval #$03
```

The entire data table must be in the same ROM page unless special code adjusts the ROMPTR register.

Example:

```
$1FA:  Table, ROMPTR H High Table / match ROMPTR with Table start
$1FB:         movr Index p
$1FC:         retval #$00
$1FD:         retval #$01
$1FE:         retval #$02
$1FF:         retval #$03
$200:         retval #$04         / table cannot be accessed from here
$201:         retval #$05         / when [Index] is >= 4
$202:         retval #$06
```

This concept can be seen in the following example code as a key code jump table.

WIF Clock.asm

Z C DC FLAGS

SUBR [SRC] from [TGT]

```
T > S       C=1   Z=0
T = S       C=1   Z=1
T < S       C=0   Z=0
```

SUBVA [VAL] from [AC]

```
AC > V      C=1   Z=0
AC = V      C=1   Z=1
AC < V      C=0   Z=0
```

The [DC] bit is set when there is a carry or borrow generated from the low nibble (BITS <3:0>) after the appropriate instruction executes.

Examples:

```
MOVVA #$0F
ADDVA #$00        / Z = 0, C = 0, DC = 0

MOVVA #$0F
ADDVA #$01        / Z = 0, C = 0, DC = 1
```

SUBR [SRC] from [TGT] <3:0>

```
T > S       C=1
T = S       C=1
T < S       C=0
```

SUBVA [VAL] from [AC] <3:0>

```
AC > V      C=1
AC = V      C=1
AC < V      C=0
```

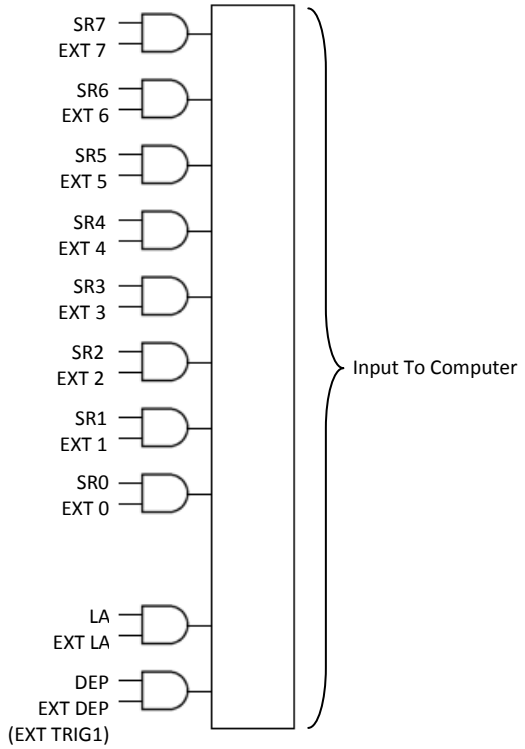
DAAC Instruction Operation

```
if Carry = 0 prior to DACC then
  after DAAC if previous Wreg was < $99, then
    C=0
  else
    C=1

$21 + $12 -> AC = $33       $66 + $45 -> AC = $AB

DAAC       -> AC = $33      DAAC       -> AC = $11
C=0                         C=1
```

IO Port Interfaces

The SR<7:0> and the LA and DEP switches are effectively ANDed with the external SR<7:0>, and the LA and DP inputs.

When the parallel Input Dev0 is used, the SR<7:0> and LA and DP switches must be set to 1, otherwise the computer will not recognise changes from the external inputs.

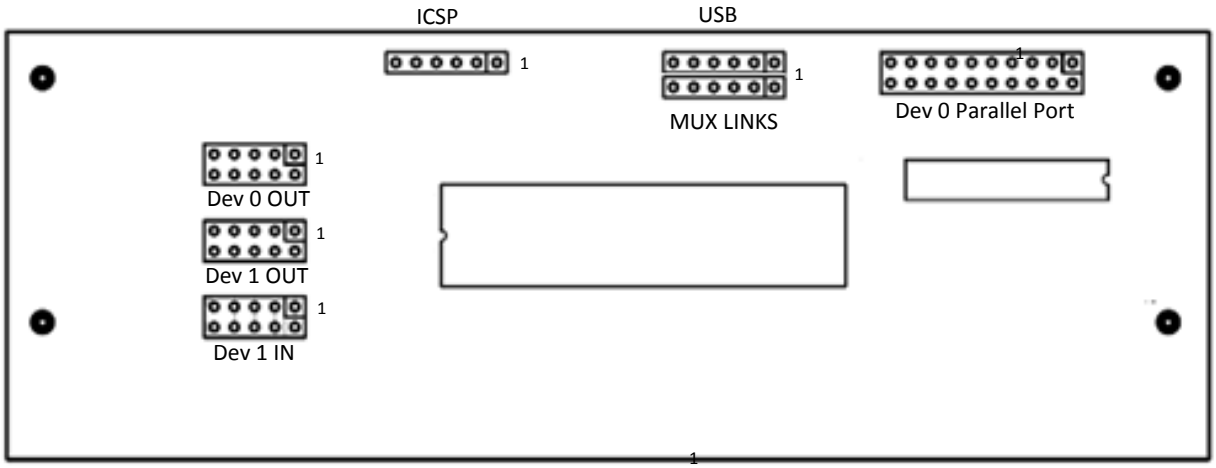


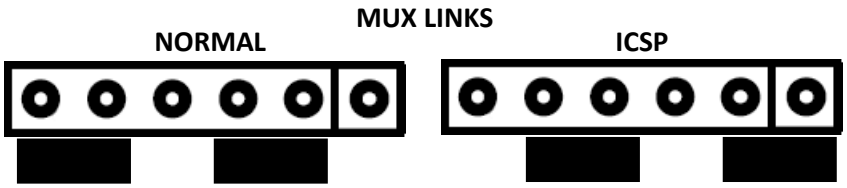
In WIF mode, the DEP and EXAM switches become TRIG1 and TRIG2.

These switches can now provide triggers to modify code execution via the instruction `SKIPTS`.

TRIG1 can also be accessed externally from the Dev0 Parallel Port.

Port Connections





ICSP	
1	N/C
2	RB6i
3	RB7i
4	GND
5	5V
6	MCLR

USB	
1	MCLR
2	PIC TX
3	PIC RX
4	5V
5	N/C
6	GND

MUX	
1	RB6i
2	PIC RB6
3	DEP 0
4	RB7i
5	PIC RB7
6	LA 0

DEV 0 OUT	
1	CLOCK OUT
2	CLOCK OUT
3	RESET OUT
4	RESET OUT
5	RESET IN
6	RESET IN
7	DATA OUT
8	DATA OUT
9	5V
10	GND

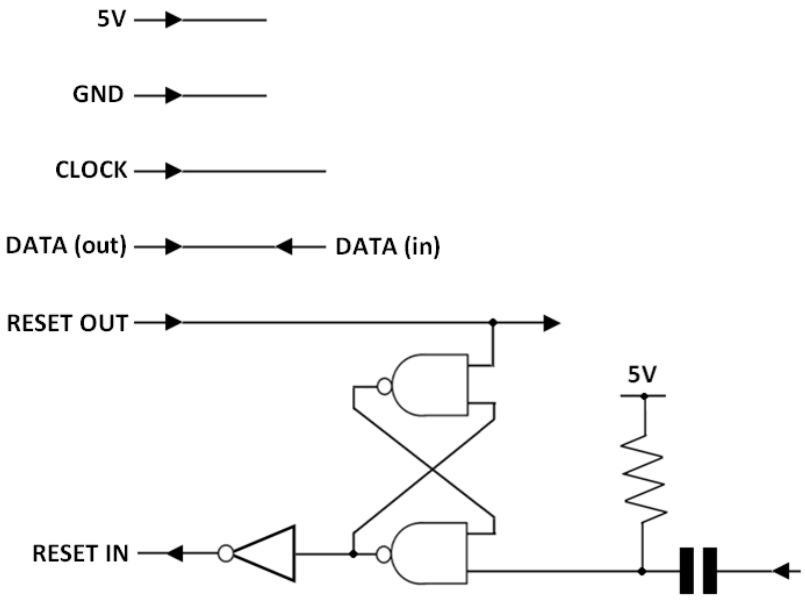
DEV 1 OUT	
1	CLOCK OUT
2	CLOCK OUT
3	RESET OUT
4	RESET OUT
5	RESET IN
6	RESET IN
7	DATA OUT
8	DATA OUT
9	5V
10	GND

DEV 1 IN	
1	CLOCK OUT
2	CLOCK OUT
3	RESET OUT
4	RESET OUT
5	RESET IN
6	RESET IN
7	DATA IN
8	DATA IN
9	5V
10	GND

DEV 0 IN – PARALLEL PORT

1	GND	**2**	GND
3	5V	**4**	EXT SR0
5	CLOCK OUT	**6**	EXT SR1
7	DATA IN	**8**	EXT SR2
9	RESET IN	**10**	EXT SR3
11	N/C	**12**	EXT SR4
13	N/C	**14**	EXT SR5
15	RESET OUT	**16**	EXT SR6
17	EXT DEPOSIT (TRIG1)	**18**	EXT SR7
19	EXT LOAD ADDRESS	**20**	N/C

Typical Serial Input / Output Interface Circuit Concept



The NAND gate flip flop circuit that controls the interface RESET could also be controlled by using two microprocessor pins if it is part of the interface circuit.

The RESET IN pin should be Logic 0 when the device is ready to accept input data or ready to transmit output data.

The RESET OUT pin is pulsed Logic 0 to reset the flip flop on the T1 (BO) or T13 (AO) clock cycle – RESET IN goes to Logic 1. Data is shifted in or out during cycles T2 – T9.

Input Example – Keyboard.

- The keyboard interface is connected and resets itself when powered up.
- `Reset In` is HI.
- This tells the EDUC-8 code that the keypad is not ready to send data.
- The user presses a key on the keypad.
- The interface sets `Reset In` to LO.
- The EDUC-8 code sees the LO and starts to send 8 `Clock` pulses.
- On each HI going clock pulse the interface sets the `Data` line HI or LO for the EDUC-8 to read on the next LO going `Clock` signal.
- After 8 clock pulses, the EDUC-8 then briefly sets `Reset Out` LO.
- The keypad interface then sets `Reset In` HI and gets ready to accept a new key press.

Output Example – LED Display

- The LED interface is connected and resets itself when powered up.
- `Reset In` is HI.
- This tells the EDUC-8 code that the display is not ready to accept data.
- The LED interface is configured so that after a delay timer expires, `Reset In` goes LO.
- The EDUC-8 code sees the LO and starts to send 8 `Clock` pulses.
- It will set the data out pin HI or LO when the `Clock` signal is HIGH.
- The LED interface will accept HI or LO `Data` when `Clock` goes LO.
- After 8 clock pulses, the EDUC-8 then briefly sets `Reset Out` LO.
- The LED interface then sets `Reset In` HI, resets the timer and then displays the received data.

It is possible for an output device to require a reset pulse prior to accepting data, (to wake it up), or after receiving data, (to clean up afterwards).

As the data is clocked out on the T2-T9 cycles, the Reset on T1 cycle or reset on T13 cycle gives that functionality.

WIF mode has BO (Before Output) or AO (After Output).

This input and output example is available to try out by using the `EDUC8 Keypad.asm` and the `WIF Keypad.asm` code examples.

There is also PIC code available for a PIC16F18345 chip to provide a keyboard and LED display interface. This can also be used with the LED and KEY interface circuits.

Default Constants

There are some predefined constants available. These can be used directly in code.

```
0                   0           / basic numerals
1                   1
2                   2
3                   3
4                   4
5                   5
6                   6
7                   7
8                   8
9                   9
INDR                0           / indirect register $00
DEV0                0           / IO devices
DEV1                1
DEV0_IN             0
DEV1_IN             1
DEV0_OUT            0
DEV1_OUT            1
TRIG1               1           / trigger switches
TRIG2               2

/ 10 digit display

D10_IDX             $40
D10_OFF             $50
D10_ON              $5F
D10_ERR             $60
D10_INI             $70

D10_0      0        D10_NEG    10       D10_F      20       D10_P      30
D10_1      1        D10_COL    11       D10_G      21       D10_Lq     31
D10_2      2        D10_DP     12       D10_H      22       D10_Lr     32
D10_3      3        D10_BK     13       D10_Lh     23       D10_S      33
D10_4      4        D10_A      14       D10_I      24       D10_Lt     34
D10_5      5        D10_Lb     15       D10_Li     25       D10_U      35
D10_6      6        D10_C      16       D10_J      26       D10_Lu     36
D10_7      7        D10_Lc     17       D10_L      27       D10_Y      37
D10_8      8        D10_Ld     18       D10_Ln     28
D10_9      9        D10_E      19       D10_Lo     29

/ Alphanumeric display

AL_IDX      0                   AL_INI      128
AL_RED      16                  AL_CLR      129
AL_GRN      17                  AL_CL1      130
AL_YEL      18                  AL_CL2      131
AL_BLU      19                  AL_TLN      132
                                AL_BLN      133
                                AL_OFF      134
                                AL_ON       135
```