ENCODER ENCODER ENCODER
ENCODER ENCODER ENCODER

ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER
ENCODER	ENCODER	ENCODER

6502 ASSEMBLER / DISASSEMBLER

PREMIER PUBLICATIONS - UK101/OHIO QUALITY SOFTWARE

ENCODER is an EPROM-based full feature 6502 assembler/disassembler supporting full mnemonics, labels, hex, dec, binary or ASCII input, etc. As it operates nominally from BASIC workspace, utility EPROMS such as PREMIER TOOLKIT II may be used in conjunction with it to provide sophisticated FIND, AUTO, REPLACE, and other functions. It also ties in directly with the CEGMON full-screen editor.

The manual supplied with ENCODER is designed to give the user all the information necessary to use it successfully. The manual is NOT a complete course in 6502 programming! Newcomers to machine code programming are strongly urged to purchase one of the books listed near the end of this manual.

INITIALISATION

To enter ENCODER, use one of the following methods. Note that in the examples below, xxxx refers to the EPROM hex start address which will be 8000,8800,9000 or 9800.

a/ RESET M xxxxG
b/ from BASIC 5, &GO$xxxx
c/ POKE11,xx:POKE12,xx:X=USR(X)
d/ CALLxxxx if you have new BASIC 1.

Once initialised, the prompt

ENCODER A/D ?

will appear. This is an invitation to Assemble or Disassemble. (See also Options)

DISASSEMBLE

Pressing 'D' will enter the disassembler. The prompt

D$

will appear and you should type the address you wish to start disassembling from, followed by 'RETURN'

Fifteen lines of code will now be displayed on screen. There are now three things you can do

1/ Pressing 'LF' or CTRL J or Up Arrow (depending on which computer you have) will continue with the next fifteen lines.

2/ Dxxxx will restart disassembly from the address 'xxxx'

3/ RETURN will put you into the command mode of the CEGMON monitor, where you can call upon all the facilities that CEGMON offers. Pressing '.U' at any time whilst still in the monitor will re-enter ENCODER at the start up prompt. Pressing RESET (BREAK on OHIOS) at any time means that you must re-initialise ENCODER (see above!).

ASSEMBLE

Pressing 'A' after initialisation causes the prompt

M/L ?

to appear. ENCODER is now asking for assembly from Tape Load or from Memory.

Pressing 'L' will cause ENCODER to assemble a previously recorded source file directly as it reads in from cassette. Make sure that you can hear a clear header tone BEFORE pressing 'L' as a syntax error will cause ENCODER to print an error message and jump to BASIC warm start.

When assembly is finished, the prompt 'M/L ?' will reappear, allowing the use of further options (see later).

In tests, ENCODER has been found to assemble from tape at speeds up to 1200 baud, but this naturally depends on the quality of the audio equipment used and which mod was done to achieve the higher load speed - they don't all work! Higher tape speeds than 1200 baud are theoretically possible.

A test on assembly speed using the 'memory' mode gave a result of a thousand lines in 72 seconds - not bad when it takes BASIC 67 secs just to list that much!

Pressing 'M' will cause ENCODER to assemble source code that has been written in the BASIC workspace.

OPTIONS

Once ENCODER has been initialised, pressing SHIFT P will jump you to the ENCODER A/D ? prompt. Users of TOOLKIT II and ENCODER should note that they use the same I/O vectors and the use of TK II will cancel out the use of SHIFT P.

At the prompt ENCODER A/D ? there are three choices:-

1/ 'RETURN' will jump you back into BASIC
2/ 'A RETURN' will jump you into the monitor command mode
3/ 'D RETURN' will take you back to ENCODER A/D ?

At any time whilst in the monitor, pressing '.U' will jump you to the ENCODER A/D ? prompt. However, should you RESET M this facility will be cancelled.

For printer output of source code, use the standard SAVE:LIST format from BASIC. To output the assembly/disassembly listing, set the SAVE flag, then use either the 'A' or 'D' mode of ENCODER.

It should be noted that when outputting to printer whilst assembling, the assembly speed is greatly reduced.

WRITING SOURCE CODE

The first line of your source code must contain

*$xxxx or *$xxxx$yyyy

where

xxxx is the address at which the source is to be assembled

yyyy is the final destination address of the code

For example

10 *$1000 $A000

will assemble into $1000 code that is finally destined to run at $A000.

If you only specify *$xxxx ENCODER will assemble code to xxxx for use at the same address.

The equivalent of a REM statement can be inserted into your source code by using a ";". This must be inserted immediately after the operand (no spaces), followed by whatever remarks you wish to make.

These will be very useful to you if you later wish to find out exactly what you intended the code to do.

Any remarks will be printed in a listing but do not affect the object code produced during assembly.

20 :SCRN=$D000; define screen label.

All 6502 instruction mnemonics must be followed by a space or a syntax error will be generated. Eg:-

30 LDX #$00

Care should be taken when using a mixture of labels and mnemonics to ensure that a BASIC keyword cannot be formed by a combination of instructions. For example:-

:TRU NOP produces :T<RUN>OP

thus generating an error message at assembly time.

LABELS

These may be up to four alphanumeric characters in length and must be preceded by a colon. Eg; :LABL or :X1. If a label is the first statement in a line, a space must be left after the label. If the label is within a line, do not leave a space. For example:-

50 :LOOP STA :SCRN,X

Labels can be defined at any point in the listing by typing :LABL=$xxxx where xxxx is a hex address and LABL is the label name.

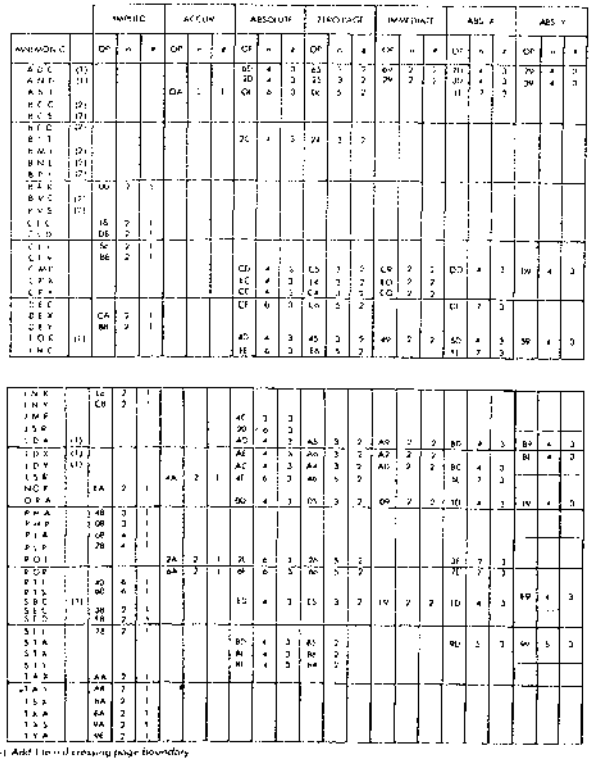
Labels can be zeroed out for future re-definition by typing:-

:LABL=$FFFF

The label does not need to be defined and your source can jump or branch to an as yet undefined label, the true address being inserted by ENCODER when it finally finds it.

ENCODER uses the last two pages of user RAM for its labels store, so you have 512 bytes less free RAM whilst assembling. Only sixty-four labels can be stored in this area, hence the use of :LABL=$FFFF for reusing labels which are no longer required.

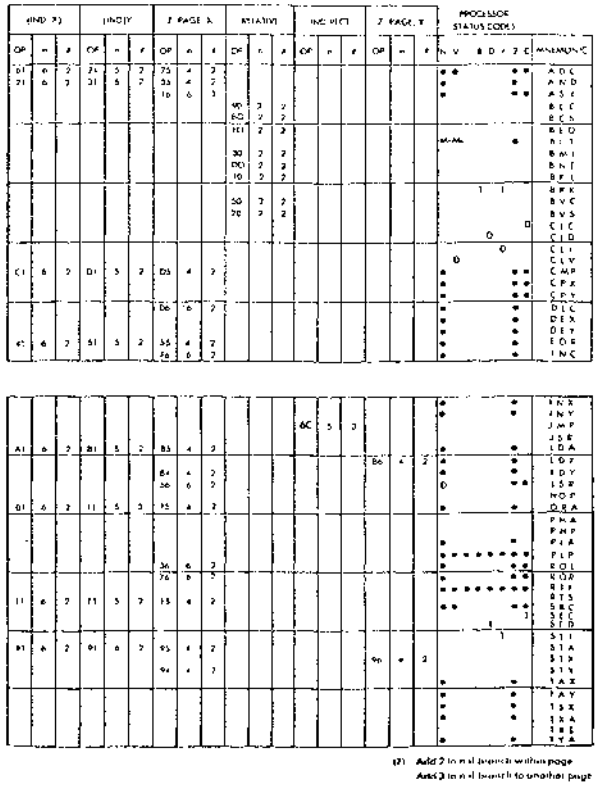
6502—INSTRUCTION SET: HEX AND TIMING



(1) Add 1 if page boundary is crossed.

Here is a sample for you to try:-

```
10 *$1000;                  start address
20 :SCRN=$D000;             define screen label
30 LDX #$00;                zero out X register
40 LDA #$BB;                character 187 into acc.
50 :LOOP STA :SCRN,X;       fill screen + offset
60 INX;                     step up one
70 CPX #$00;                test for 256 steps
80 BNE :LOOP;               if not keep going
90 RTS;                     finished - let's go home
```



(2) Add 1 if branch within page
Add 2 if branch to another page

INSTRUCTIVES

Instructives are the assembler commands which are translated directly into machine code. ENCODER recognises the industry standard mnemonics as follows:-

ADC add memory with carry to the accumulator

AND add memory with accumulator

ASL arithmetic shift left one bit (accumulator or memory)

BCC branch if carry flag clear

BCS branch if carry flag set

BEQ branch if equal to zero

BIT bit test (accumulator and memory)

BMI branch if minus

BNE branch if not equal to zero

BPL branch if plus

BRK break program execution

BVC branch if overflow flag clear

BVS branch if overflow flag set

CLC clear carry flag

CLD clear decimal mode

CLI clear interupt mask (enable interrupts)

CLV clear overflow flag

CMP compare accumulator with memory

CPX compare index register X with memory

CPY compare index register Y with memory

DEC decrement memory contents by one

DEX decrement index register X by one

DEY decrement index register Y by one

EOR exclusive OR accumulator with memory

INC increment memory contents by one

INX increment index register X by one

INY increment index register Y by one

JMP jump to new address

JSR jump to a subroutine

LDA load the accumulator from memory

LDX load the index register X from memory

LDY load the index register Y from memory

LSR logical shift right one bit (accumulator or memory)

NOP no operation

ORA logically OR accumulator with memory

PHA push accumulator contents onto stack

PHP push status register contents onto stack

PLA pull byte from stack - transfer to accumulator

PLP pull byte from stack - transfer to status register

ROL rotate accumulator or memory left one bit (through carry)

ROR rotate accumulator or memory right one bit (through carry)

RTI return from interrupt

RTS return from subroutine

SBC subtract memory with borrow from accumulator

SEC set carry flag

SED set decimal mode

SEI set interrupt mask (disable interrupts)

STA store accumulator contents into memory

STX store index register X contents in memory

STY store index register Y contents in memory

TAX transfer accumulator to index register X

TAY transfer accumulator to index register Y

TSX transfer stack pointer to index register X

TXA transfer index register X to accumulator

TXS transfer index register X to stack pointer

TYA transfer index register Y to accumulator

The 6502 processor offers 11 different addressing modes which can be used on most of the above instructives. They are as follows:-

IMMEDIATE	ABSOLUTE	ZERO PAGE (direct)
IMPLIED	ACCUMULATOR	INDIRECT INDEXED
INDEXED INDIRECT	INDIRECT	ABSOLUTE INDEXED
RELATIVE	ZERO PAGE (indexed)	

Here are a few samples of the different ways these modes are used to affect the LDA instruction, with a BASIC equivalent for those of you who are new to assembly code:-

INSTRUCTION	ADDRESSING MODE	BASIC EQUIVALENT
LDA #$20	immediate	A=32
LDA $20	zero page	A=PEEK(32)
LDA $0220	absolute	A=PEEK(544)
LDA $20,X	zero page indexed	A=PEEK(32+X)
LDA $0220,Y	absolute indexed	A=PEEK(544+Y)
LDA ($20,X)	indirect indexed	A=PEEK(PEEK(32+X))
LDA ($20),Y	indexed indirect	A=PEEK(PEEK(32)+Y)

Zero page addressing means that the address that the processor is accessing is within the range $00 - $FF (or 0-255) only. However, the zero page technique may be applied to addresses greater than 255 by using the indirect mode.

DIRECTIVES

Directives are used to tell the assembler where in memory to put the object code, define labels and set up data stores.

Start Address

The syntax is *$xxxx or *$xxxx$yyyy and is used to tell the assembler where the object code is to be stored. See section on source code for explanation of use.

Label Definition

The syntax is :LABL=$xxxx or JSR :LABL or :LABL JSR :MORE.
All label entries must be preceded by a colon.

Data locations

There are differences in the way that ENCODER and a standard assembler handle this subject. A standard assembler contains the directives .BYTE to allow the entry of blocks of data. However, you need only use the '#' sign (SHIFT3 from the keyboard) in ENCODER.

The syntax is
```
#$FF#$FF#$FF      for hex
#255#255#255      for decimal
#%xxxxxxxx        for binary
```

The length of the data block can be as long as memory allows.
To enter text use the double quote (") Only the starting quote is required.

```
"This is the correct way.
```

If data is required after a text string, a new line must be used starting #$FF or #255 etc, or whatever bytes are required.

Both the data and text entry descriptions above will allow the bytes to be entered immediately into the object code. However, if you require ENCODER to remember where to find the data or text, a label should precede the entry. Eg

```
:TEXT "This is a sample of text insertion
```

Standard assemblers also contain the directives .DBYTE and .WORD and these have the following effects.
.WORD will store the address of a specified label into the current RAM location in the order low byte/high byte.
.DBYTE has the same effect as .WORD except the address is stored in the reverse order - high byte/low byte. An example:-

```
LABL=$AABB
.WORD LABL  stores $BB and $AA in the next two bytes
.DBYTE LABL stores $AA and $BB in the next two bytes
```

ENCODER also allows the use of label address storage but uses a different syntax, as follows.

#:LABL

stores the 16 bit address of the label :LABL (in the form low/high byte) in the next two bytes.

#:LABL>

stores the 16 bit address of the label :LABL (in the form high/low byte) in the next two bytes.

```
:LABL=$1000
#:LABL  will store $00 $10 into the next two bytes
#:LABL> will store $10 $00 into the next two bytes.
```

As can be seen from the above, #:LABL replaces .WORD and #:LABL> replaces .DBYTE.

The statement #$xxxx also replaces .WORD and this function can be used several times on one line. For example:-

```
10 *$1000
20 #$aabb#$ccdd#$eeff#$gghh#$xxyy
```

The above sample would store $bb $aa $dd $cc etc from memory location $1000 onwards so as to form an address look-up table perhaps.

ENCODER also contains the facility for off-setting the label by using the syntax #:LABL+$XX or #:LABL+$XX> where $XX is in the range $00 - $FF, but this can only be a positive offset. For example:-

#:LABL+$XX> will add $XX to the address of :LABL and then store the new address in the form high/low byte. If in the example above, $XX = $80 then $10 $80 would be stored in the next two bytes.

#:LABL+$XX will add $XX to the address of :LABL and store the new address in the form low/high byte. If $XX=80 then $80 $10 would be stored in the next two bytes.

Another facility available allows ENCODER to select the high/low byte of a label address as the operand of an immediate instruction.

```
LDA #:LABL;   will load A with the low byte of the :LABL address
LDA #:LABL>;  will load A with the high byte of the :LABL address
```

The labels may be offset by values from 0 - 255 as discussed above. For example:-

```
10 *$1000;                  start address
20 :JUMP=$0000;             set up warm start jump
30 :STRT LDA #:STRT;        with the
40 STA :JUMP+1;             address of the
50 LDA #:STRT>;             start of
60 STA :JUMP+2;             this routine
70 LDA #$4C
80 STA :JUMP
90 :TEXT "This is a test;   define some text
100 #$00;                   terminated by a null
110 LDY #:TEXT>;            set up A,Y
120 LDA #:TEXT;             with start address
130 JSR $A8C3;              and print it
140 JMP $A274;              do warm start
```


Here are two more samples for you to try out:-

```
10 *$1000$0300;             assembly and destination
20 :X1 "THIS IS A SAMPLE;   text to write to screen
30 #$00;                    Equivalent of .BYTE
40 :SCRN=$D300;             define screen label
50 LDX #$00;                zero out X register
60 :LOOP LDA :X1,X;         get new character from text
70 STA :SCRN,X;             Put to next screen location
80 INX;                     step up one
90 CMP #$00;                test for null (.BYTE CHAR.)
100 BNE :LOOP;              if not keep going
110 RTS;                    finished!
```

```
10 *$1000;                  start address
20 :STOR #$A1#$20#$A1;      look up table named :STOR
30 #$20#$A1#$20#$A1#$20
40 #$A1#$20#$A1#$20#$A1
50 LDX #$00;                zero out X register
60 :LOOP LDA :STOR,X;       get next char from STOR
70 STA $D100,X;             put to next screen location
80 INX;                     step up one
90 CPX #$0D;                test for all done
100 BNE :LOOP;              If not keep going
110 RTS;                    Finished. Let's go home.
```

NB. You are advised that to enable you to use ENCODER to its fullest extent, you purchase a book on 6502 Assembly Language and Programming. If you are a newcomer to Assembly code (or still confused !), we would recommend that you purchase '6502 Software Design' by Leo Scanlon. If you are a more experienced programmer try 'Programming and Interfacing the 6502 by De Jong' (both available from PREMIER).

ERROR MESSAGES

Encoder has full error protection and will produce the following error messages:-

*	No start address (*$xxxx etc.)
L	Invalid Label - either you have used more than the allowed 64 labels or the label contains non-alphanumeric characters.
M	Wrong mode. Mnemonic cannot be used in this mode.
N	Illegal Number. Number too large or using illegal digits; i.e. a decimal number contains non-numeric characters or a binary number contains anything other than 0 or 1.
O	Out of memory. Perhaps ENCODER has just attempted to overwrite its label store with object code. Only 42 undefined labels can be used at once. A tip is to write all subroutines early on in the listing.
R	Out of Range. Trying to branch to a label that is not within +-127 bytes of branch address.
S	SYNTAX error. Spelling mistakes, etc.

ENCODER was written by Dr A Eddleston.

208 CROYDON RD, ANERLEY, LONDON, SE20 7YX

Telephone 01 - 659 - 7131