```
0E 8D 01
4E 60 44
52
```

meaning: opc 0E, which takes two data bytes; then opc 4E, which also takes two data bytes; then opc 52, which isn't a legal opc, causing the equivalent of a syntax error in the processor. This demonstrates how an initial misunderstanding generates a series of gross logical errors in the program execution.

This also clearly shows some other important points about machine code: it really *is* unfriendly to the user (at least, in the beginning) in that it is difficult to read and write; it is highly sequential with nothing but the order differentiating one instruction from the next; and it is literal as only a machine can be, obeying wrong instructions as readily as it will correct ones, and rejecting only syntax errors.

Some of the unfriendliness can be avoided by writing alphabetic mnemonics instead of the numeric op-codes while the program is being developed, and only resorting to op-codes when the program is actually loaded into memory. These mnemonics constitute the processor Assembly language, and translating them into numerical op-codes is called assembly or assembling. Notice that there is a direct one-to-one correspondence between the set of Assembly language mnemonics, and the set of op-codes: although Assembly language is a higher-level language than machine code, the difference is minimal.

If we rewrite the machine code fragment above in 6502 Assembly language, then it looks like this:

```
0000 A9 0E     LDA #$0E
0002 8D 01 4E  STA $4E01
0005 60        RTS
```

while the same sequence of operations in Z80 Assembly language looks like this:

```
0000 3E 0E     LD A,$0E
0002 32 01 4E  LD ($4E01),A
0005 C9        RET
```

The first column shows the hex address in memory of the first byte of the line — the opc A9 in the 6502 list, for example, is in byte0; the page byte 4E in both lists is in byte4, and so on. The next column may contain one, two or three bytes, and shows the machine code listing. The third column starts with an Assembly language mnemonic, and shows the Assembly language version of the machine code. Don't bother trying to puzzle it all out now, it's enough that you've seen an Assembly language list, and can observe the differences and similarities between the Z80 and 6502 versions. You might also notice that the address in the second line appears in conventional lo-hi form in machine code, but 'normal' hi-lo form in Assembly language.

In the next instalment of the course we'll start to examine op-codes in detail, and take a look at the architecture of the microprocessor.

Hexadecimal Convertor

To convert the Mempeek program of page 59 so that byte contents are displayed in hexadecimal rather than decimal, make the following changes:

BBC Micro

Add:

```
3000 DEF PROCHXPRINT(DECNUM)
3100 LOCAL X$
3200 X$="0123456789ABCDEF"
3300 HB=INT(DECNUM/16):LB=DECNUM-HB*16
3400 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)+"   "
3500 PRINT B$;
3600 ENDPROC
```

and change line 600 to:

```
600 PROCHXPRINT(PK%)
```

Spectrum

Add:

```
10 LET X$="0123456789ABCDEF"
3000 REM******HEX BYTE S/R*******
3100 LET HB=INT (PK/16): LET LB=PK-HB*16
3200 LET B$=X$(HB+1)+X$(LB+1)+"   "
3300 PRINT B$;
3400 RETURN
```

and change line 600 to:

```
600 GOSUB 3000
```

Commodore 64

Add:

```
10 LET X$="0123456789ABCDEF"
3000 REM******HEX BYTE S/R*******
3100 HB=INT(PK/16):LB=PK-HB*16
3200 B$=MID$(X$,HB+1,1)+MID$(X$,LB+1,1)+"   "
3300 PRINT B$;
3400 RETURN
```

and change line 600 to:

```
600 GOSUB 3000
```

These changes will cause the contents of the memory to be displayed in hexadecimal. The start address and number of bytes should still be entered in decimal