Servicing the BBC Micro

6502 Assembly Language

Assembly language

The 6502 microprocessor

The 6502 is the 'brains' of the computer containing all the logic required to recognise and execute the list of instructions called the program. All the time the machine is switched on the microprocessor is busy, reading numbers from memory, interpreting them as instructions and then carrying out the operations specified by these instructions. To help it with this task there are a number of special memory locations, called registers, on the microprocessor chip itself. These are identified by name rather than number, i.e. they are not part of the so-called Memory Map.
The registers of the 6502 are indicated in *Fig. 1.*

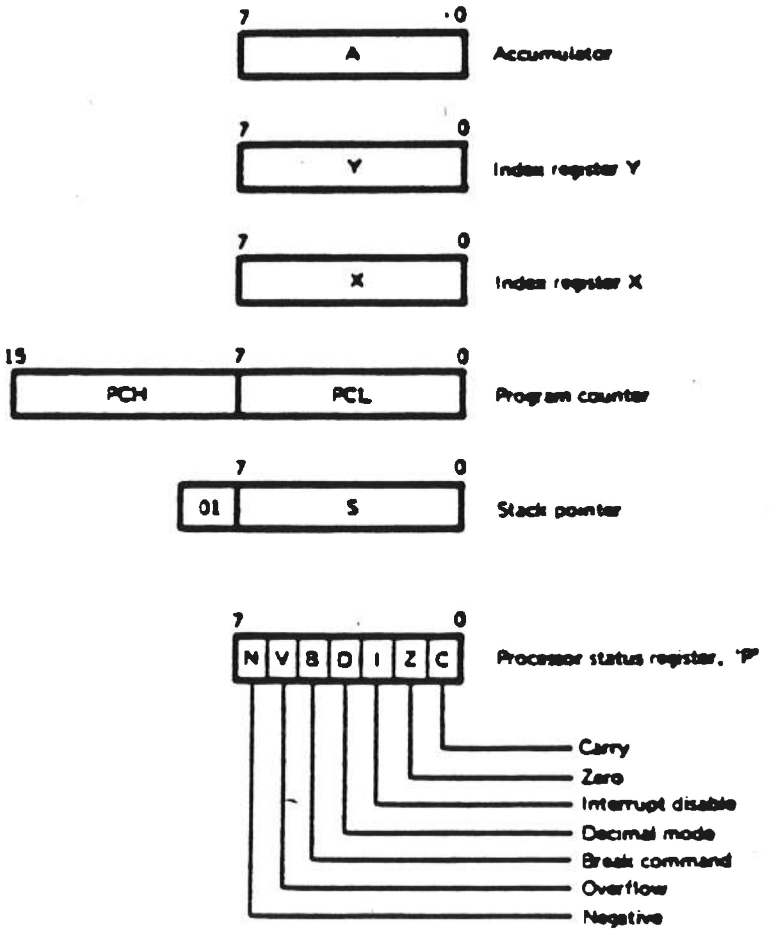


Fig. 1.

The accumulator A is the register involved in most of the mathematical and logical functions because of its greater power than other registers and memory locations. The X and Y registers are used to store values for counting, timing and indexing (identify an address or sequence of addresses referenced to some base address and particularly useful in scanning tables of values with the minimum of programming). The program counter, PC, registers the current address; the stack pointer keeps records of information put aside when the microprocessor is temporarily diverted from its main task, and the status register is a collection of individual bits identifying features of the previous instruction.

The program followed by the microprocessor bears little resemblance to BASIC. The only language the processor understands is the language of 0s and 1s, or MACHINE CODE. For example, the set of binary numbers below forms a short machine code program that stores the number 21 (hex) in memory location 1600 (hex):

Binary	*In Hex Representation*
10101001	A9
00100001	21
10001101	8D
00000000	00
00010110	16

Some of these numbers are called OPERATION CODES or 'OP CODES', and tell the processor what it has to do. In the example, A9 tells the processor to load its accumulator with the next number, namely 21. The next code, 8D, tells it to store the contents of the accumulator in location 1600 (hex), the memory location defined by the next two numbers.

Although it is possible to write programs directly in machine code (in some early microcomputers it was the only method, e.g. KIM), it is a slow process, prone to error, requiring the programmer to make continuous reference to instruction tables similar to those shown in Appendix B . An alternative approach is to write programs in a more 'human friendly' format called ASSEMBLY LANGUAGE. This language uses alphabetic abbreviations for each type of instruction rather than binary or hex OP codes. For example, abbreviations such as LDA and STA are used to represent the operations LoaD the Accumulator and STore the Accumulator. These abbreviations are often called 'mnemonics' because they are more easily remembered than OP codes. Written in mnemonics, the example program becomes:

```
LDA #21
STA 1600
```

The question now arises, 'How does this assembly language program become the machine code program stored in the computer's memory?' The answer is to use a special program called an ASSEMBLER which translates the 'easily understood by humans' assembly program into the language of the processor, machine code.

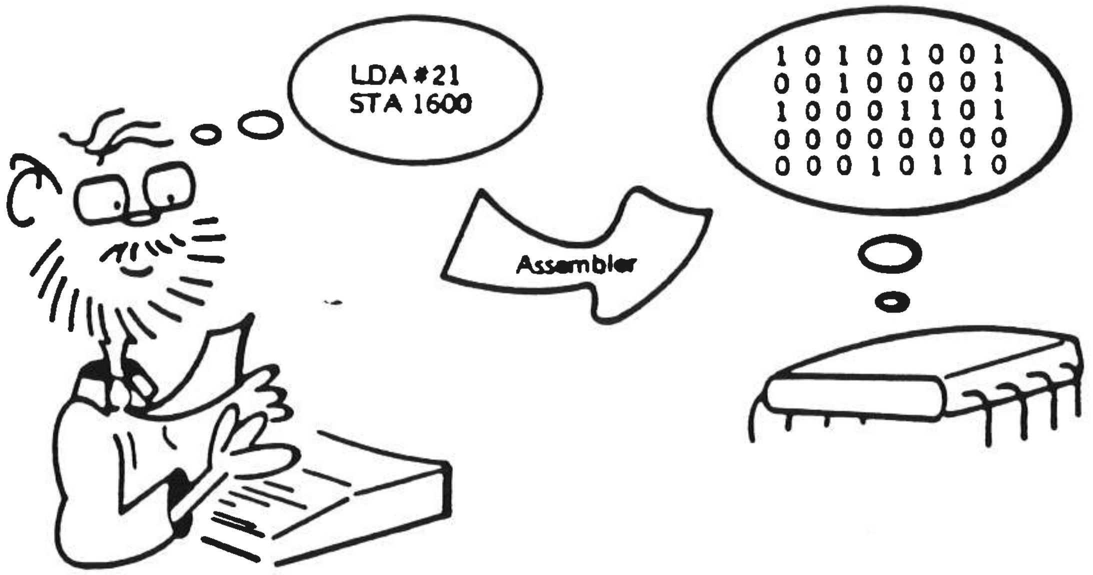


Fig. 2. Assembler

Using the assembler

Entering an assembly language program on the BBC micro is similar to entering a BASIC program. There are some extra instructions required that can best be explained with the aid of an example (*Fig. 3*).

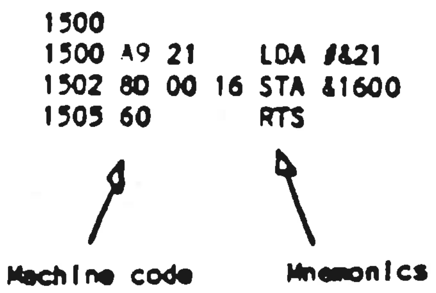
```
10 P% = &1500

20 [

30 LDA #&21

40 STA &1600

50 RTS

60 ]
```

Fig. 3.

—Line 10 acts as an 'origin' statement for the program, telling the assembler where the machine code has to be positioned in the computer's memory. The integer variable P% is used for this task.
—Lines 20 and 60 contain square brackets (they appear as arrows in MODE 7) that enclose the Assembly Language program.

—Line 30 contains the instruction to LoaD the Accumulator with 21 (hex).
—Line 50 contains a ReTurn from Subroutine instruction which returns control to BASIC on completion of the machine code program.

The RUN command will assemble the program, placing the machine code in memory. An assembler listing of the mnemonics and the machine code will also be sent to the display.



```
1500
1500 A9 21     LDA #&21
1502 8D 00 16 STA &1600
1505 60        RTS
```

Fig. 4.

Examination of memory location &1600 will show that the program has not yet been executed. Try:

PRINT ~ ? &1600

i.e. it is unlikely that it contains &21.

To execute the machine code program we use the CALL statement followed by the starting address of the routine, i.e. type:

CALL &1500

The computer will execute the program and return to BASIC, displaying the '>' prompt. Check memory location &1600 again. It should now contain &21.

The CALL statement can be included at the end of the program, see *Fig. 5.*

```
10 P% = &1500

20 [

30 LDA #&21

40 STA &1600

50 RTS

60 ]

70 CALL &1500
```

Fig. 5.

On the command RUN the program will be assembled and then executed.

Comments and labels

Documentation considerably improves a program, making it easier to read. In BASIC, comments are added using REM statements. Unfortunately REM statements are not allowed within the assembler program and comments must be attached to assembler statements using a semi-colon or backslash (\)

e.g. 40 STA &1600; THIS IS A COMMENT

Variable names can be used to represent memory addresses or memory contents. However, they must be defined outside the square brackets holding the assembler program

```
e.g. NUMBER=&21
     STORE  =&1600
```

An exception to this rule is the definition of program addresses. If we wished to label the starting address of a program 'START' it could be done as follows.

```
 30  START LDA #&21
```

The label is prefixed by a full stop (period) '.' and separated from the assembler mnemonic by at least one space.

Using comments and labels the previous program becomes that listed in *Fig. 6.*

```
 10 REM ************************
 20 REM DEMO ASSEMBLER PROGRAMME
 30 REM ************************
 40 P%=&1500
 50 NUMBER=&21
 60 STORE=&1600
 70 [
 80 .START LDA #NUMBER ;GET NUMBER
 90        STA STORE ;STORE NUMBER
100        RTS  ;BACK TO BASIC
110  ]
120  CALL START
```

Fig. 9.6.

Finally a word of caution on the choice of labels. The same restrictions are placed on variable names as in BASIC. In particular, BASIC 'keywords' like PRINT, NEXT, REPEAT, END, etc. are definitely NOT ALLOWED. However, their lower case equivalents are permitted.

Operating systems subroutines

The BBC micro's operating system ROM contains many useful machine code subroutines that can be included in your assembler program. Three routines of particular interest are:

1. OSRDCH—OPERATING SYSTEM READ CHARACTER Address— &FFEO:
   Reads a character from 'the input channel', normally the keyboard, placing it in the accumulator.
2. OSWRCH—OPERATING SYSTEM WRITE CHARACTER Address—&FFEE:
   Writes a character in the accumulator 'down the output channel', normally to the screen.
3. OSASCI—OPERATING SYSTEM ASCII Address—&FFE3:
   As for OSWRCH except that a line feed is automatically inserted with a carriage return.
   (Note: The X and Y registers are not affected by any of these routines.)

The example in *Fig.* 7 turns the computer into an electronic typewriter that will ignore all BASIC keywords and merely display depressed keys on the screen. The subroutine OSRDCH is used to obtain the ASCII code of any depressed key. OSASCI then transfers this code from the accumulator to the display. The JMP START instruction sends the processor back to the keyboard to look for another depressed key.

```
 10REM ************************
 20REM USING THE OS SUBROUTINES
 30REM ************************
 35 DIM SPACE 100
 40 OSASCI=&FFE3
 50 OSRDCH=&FFE0
 60 P%=SPACE
 70 [
 80 .START JSR OSRDCH ;GET CHART. FROM KB
 90        JSR OSASCI ;PLACE ON SCREEN
100        JMP START ;REPEAT
110  ]
120  CALL START


0F22
0F22 20 E0 FF .START JSR OSRDCH ;GET CHART. FROM KB
0F25 20 E3 FF JSR OSASCI ;PLACE ON SCREEN
0F28 4C OF JMP START ;REPEAT
```

Fig. 7.

In this example we have used a different technique to instruct the assembler where to position the machine code. Rather than defining uniquely where the code has to be placed, the DIM statement in line

35 reserves 101 bytes and places the machine code in this reserved space at the end of the BASIC program. This technique has the advantage that the assembler will ensure that the machine code program is positioned in a safe place within the system memory and will not corrupt either the original BASIC program or memory locations allocated to the screen.

Two pass assembly

The BBC assembler uses the full stop to define labels within an assembly program. However, problems arise when a label is referred to before it is defined. This situation is illustrated in the example in *Fig. 8* where the 'typewriter' program has been modified to return control to BASIC whenever the asterisk key is depressed:

```
 35 DIM SPACE 100
 40 OSASCI=&FFE3
 50 OSRDCB=&FFE0
 60 P%=SPACE
 70 [
 80.START JSR OSRDCH  ;GET CHART. FROM KB
 85       CMP #ASC"*" ;IS IT AN "*"
 87       BEQ FINI    ;IF SO QUIT
 90       JSR OSASCI  ;PLACE ON SCREEN
100       JMP START   ;REPEAT
105.FINI RTS         ;RETURN TO BASIC
110 ]
120 CALL START
```

Fig. 8.

The label FINI appears in line 87 but is not defined until line 105. An attempt to assemble the program would give the result shown in *Fig. 9*

```
0F28
0F28 20 E0 FF .START JSR OSRDCH ;GET CHART. FROM KB
0F2B C9 2A     CMP #ASC"*" ;IS IT AN "*"

No such variable at line 87
```

Fig. 9.

—namely the assembler stops, displaying an error message. The solution is to allow the assembler to pass through the source program twice. In the first pass the assembler establishes a table of labels and their addresses. In the second pass it uses these addresses to construct the final machine code program. Due to the assembler being 'embedded' in BASIC, the extra programming required to initiate the two pass assembly is relatively simple, requiring only a FOR . . . NEXT loop to sent the assembler through the program twice, see *Fig. 10.*

```
    FOR N = 1 TO 2

    P% = SPACE

  [
```

```
  ]

   NEXT N
```

Fig. 10.

However two points are worth noting:

(i) The equate statement for the origin P% must be enclosed within the loop so that it is reset to the correct value at the start of each pass.

(ii) A 'OPT' statement is required to suppress error messages and prevent the assembler halting during the first pass. A number between 0 and 3 is used with this statement to give the following options during assembly:

OPT0	assembler errors suppressed, no listing
OPT1	assembler errors suppressed, listing
OPT2	assembler errors reported, no listing
OPT3	assembler errors reported, listing

A reasonable choice of options might be OPT1 during the first pass to suppress error messages and OPT3 during the second pass to display any errors still remaining. This is achieved in the example in *Fig. 11* by placing the counter N after the OPT statement in line 70 and assigning it the values 1 and 3 in line 55:

```
 35 DIM SPACE 100
 40 OSASCI=&FFE3
 50 OSRDCH=&FFE0
 55 FOR N=1 TO 3 STEP 2
 60   P%=SPACE
 70  [OPTN
 80  .START JSR OSRDCH ;GET CHART. FROM KB
 85         CMP #ASC"*" ;IS IT AN "*"
 87         BEQ FINI    ;IF SO QUIT
 90         JSR OSASCI  ;PLACE ON SCREEN
100         JMP START   ;REPEAT
105    .FINI RTS        ;RETURN TO BASIC
110    ]
115     NEXT N
120  CALL START
```

Fig. 11.

With these options the assembler produces a listing on the display during each pass. However the relative displacement of 06 at &0F4C is only resolved in the second listing.

```
0F46           OPTN
0F46 20 E0 FF .START JSR OSRDCH ;GET CHART. FROM KB
0F49 C9 2A     CMP #ASC"*" ;IS IT AN "*"
0F4B F0 FE     BEQ FINI    ;IF SO QUIT
0F4D 20 E3 FF JSR OSASCI ;PLACE ON SCREEN
0F50 4C 46 0F JMP START  ;REPEAT
0F53 00        .FINI RTS       ;RETURN TO BASIC
0F46           OPTN
0F46 20 E0 FF .START JSR OSRDCH ;GET CHART. FROM KB
0F49 C9 2A     CMP #ASC"*" ;IS IT AN "*"
0F4B F0 06     BEQ FINI    ;IF SO QUIT
0F4D 20 E3 FF JSR OSASCI ;PLACE ON SCREEN
0F50 4C 46 0F JMP START  ;REPEAT
0F53 60        .FINI RTS       ;RETURN TO BASIC
```

Fig. 12.

A final point worth noting is the use of the statements VDU 14 and VDU 15 to switch the 'PAGE MODE' on and off. This can prove useful when assembling large programs to examine the assembly listing a page or 'screenful' at a time.

Mixing machine code and BASIC

Two statements allow control to pass to a machine code routine from BASIC—'CALL' and 'USR'. With both statements the processors A, X, and Y registers are initialised to the least significant bytes of the integer variables A%, X% and Y% and the carry flag is set to the least significant bit of the variable C% on entry to the machine code routine.

e.g.

```
10  A%=&41
20  CALL &FFEE
30  END
```

```
10  A%=&41
20  Z=USR(&FFEE)
30  END
```

Each of these programs would send 41 hex or ASCII 'A' to the display routine OSWRCH at FFEE hex.

On completion of the machine code routine USR return a 32 bit number made up of the processor's status, Y, X and A registers. For example, see *Fig. 13.*

The CALL statement offers greater flexibility, allowing program variables of all types to be passed to and from a machine code subroutine. To aid this transfer a 'parameter block', starting at &0600,

```
 10REM ***************************
 20REM DEMO OF THE "USR" FUNCTION
 30REM ***************************
 40 DIM SPACE 100
 50 P%=SPACE
 60 [
 70 LDA #&AA ;SET A=&AA
 80 LDX #&BB ;SET X=&BB
 90 LDY #&CC ;SET Y=&CC
100 SEC      ;SET CARRY =1
110 RTS      ;RETURN TO BASIC
120 ]
130 RESULT=USR(SPACE)
140 PRINT~RESULT
```

```
0F35
0F35 A9 AA   LDA #&AA ;SET A=&AA
0F37 A2 BB   LDX #&BB ;SET X=&BB
0F39 A0 CC   LDY #&CC ;SET Y=&CC
0F3B 38      SEC      ;SET CARRY =1
0F3C 60      RTS      ;RETURN TO BASIC
  B1CCBBAA
  ‿ ‿ ‿ ‿
 /  \ \  \
P   Y X   A
```

Fig. 13.

contains details of the number, location and type of variables to be passed. The block has the following structure:

0600—number of parameters
0601—low byte of address 1st parameter
0602—high byte of address 1st parameter
0603—code defining parameter type
0604—low byte of address 2nd parameter
0605—high byte of address 2nd parameter
0606—code defining parameter type
etc.

The codes used to define parameter types are:

0—8 bit byte (e.g. ?X)
4—32 bit integer variable (e.g. X%)
5—40 bit floating point number (e.g. T)
128—A string at a defined address (e.g. $X)
129—A string variable (e.g. A$)

In the example in *Fig. 14* the structure of the parameter block is illustrated by passing two variables, B% and C%, in the subroutine CALL to &FFEE in line 60:

```
10B%=&00112233
20C%=&44556677
30A%=&55
60 CALL&FFEE,B%,C%
70 FOR N=&0600 TO &0610
80 PRINT~?N
90 NEXT N
```

Fig. 14.

Lines 70 to 90 print details of the start of the parameter block in hex:

Address 0600 contains 02 – number of variables

0601	,,	08	ADL 1st variable
0602	,,	04	ADH 1st variable
0603	,,	04	– code for integer variable
0604	,,	0C	ADL 2nd variable
0605	,,	04	ADH 2nd variable
0606	,,	04	– code for integer variable
		FF	
		etc.	

Investigating &0408 and &040C, we find the values of the two variables B% and C% (*Fig. 15*).

```
100 FOR N=&0408 TO &040F
110 PRINT~?N
120 NEXT N


 33 )
 22 }       B%
 11 }
  0 )
 77 )
 66 }
 55 }       C%
 44 )
```

Fig. 15.

The final example illustrates a machine code program that will convert and display decimal numbers between 0 and 255 as binary. The program begins by assembling and inserting the machine code program. A small BASIC program then calls the conversion utility, passing the parameter NUMBER % which is then displayed in binary.

```
   10 DIM BINARY 100 ,TEMP 4
   20 OSASCI=&FFE3
   30  FOR N=0 TO 2 STEP 2
   40   P%=BINARY
   50  [OPTN
   60      LDY #00 ;SETY=0
   70      LDA &0601 ;TRANSFET POINTERS TO ZERO PAGE
   80      STA &80
   90      LDA &0602
  100      STA &81
  110     LDA (&80),Y ;GET NUMBER
  120     STA TEMP ;PLACE IN TEMPORARY STORE
  130    LDX #08 ;USE COUNTER TO EXAMINE 8 BITS
  140   .START BIT TEMP ;"1" OR "0"
  150       BPL ZERO ;BRANCH IF ITS A ZERO
  160       LDA #ASC"1" ;PRINT "1'
  170       JSR OSASCI   ;TO DISPLAY
  180       JMP ROTATE
  190   .ZERO LDA #ASC"0" ;PRINT "0"
  200       JSR OSASCI
  210   .ROTATE ROL TEMP ;ROTATE BYTE LEFT
  220       DEX    ;NEXT BIT
  230     BNE START
  240     LDA #0D ;C-RETURN TO DISPLAY
  250     JSR OSASCI
  260     RTS       ;BACK TO BASIC
  270 ]
  280 NEXT N
  290 REM ***************
  300 REM BASIC PROGRAMME
  310 REM TO GENERATE
  320 REM BINARY NUMBERS
  330 REM ***************
  340  FOR NUMBER%=0 TO 16
  350    CALL BINARY,NUMBER%
  360       NEXT NUMBER%
  370  END
  >RUN
00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
00010000
```

Fig. 16.

Driving graphics from machine code

All the BASIC keywords used to control the display have their equivalent VDU statement, e.g:

PRINT "A" is the same as VDU 65
MODE 5 is the same as VDU 22,5
COLOUR 3 is the same as VDU 17,3

etc.

The link between the BASIC VDU statement and operating system display routine OSWRCH is easily understood if the keyword 'VDU' is interpreted as 'SEND THE FOLLOWING BYTE (S) TO OSWRCH', i.e. the assembly language equivalent of

(a) PRINT "A" or VDU 65 is:

```
LDA #65
JSR OSWRCH
```

(b) MODE 5 or VDU 22,5 is:

```
LDA #22
JSR OSWRCH
LDA #5
JSR OSWRCH
```

The VDU statements use all 32 ASCII control codes (i.e. ASCII codes not used as symbols or alphanumeric characters). The first byte after the VDU statement, i.e. the first byte sent to OSWRCH, selects the desired display function. The operating system then knows how many more bytes are required to complete the instruction, e.g. MODE selection only requires one byte after the code, whereas redefining the shape of a display character requires 9.

The example program in *Fig. 17* selects the display mode.

```
 10 REM **********************
 20 REM SELECTING SCREEN MODE
 30 REM FROM AN ASSEMBLY
 40 REM LANGUAGE ROUTINE
 50 REM **********************
 60 OSWRCH=&FFEE
 70 DIM SPACE 100
 80 INPUT"WHICH MODE"M
 90 P%=SPACE
100 [
110 LDA #22     ;CONTROL CODE FOR MODE SELECT
120   JSR OSWRCH ;DOWN OUTPUT CHANNEL
130   LDA #M     ;SELECT MODE
140   JSR OSWRCH ;DOWN OUTPUT CHANNEL
150   RTS       ;BACK TO BASIC
160  ]
170  CALL SPACE
180  PRINT "THIS IS MODE";M
190  END
```

Fig. 17.

Using control codes as a means of selecting and driving different display functions adds greatly to the BBC micro's flexibility. It can be adapted as a colour graphics terminal communicating through either its RS423 serial port to a larger mainframe computer, or through its own system bus, called the 'Tube', to a second processor option.

Appendix A

MOS TECHNOLOGY MCS6502 SERIES

The MOS Technology 6502 series microprocessors are *n*MOS 8-bit types, of which the 6502 is probably the most commonly found. Other processors in this series are mainly simplified variants designed to fit into smaller packages.

In many respects the basic design philosophy of the 6502 follows the same lines as that of the Motorola 6800 series. The 6502 is a slightly less complex processor in terms of its architecture, but it can in some respects be considered as an enhanced version of the 6800, particularly in its comprehensive range of addressing modes.

Because of the similar hardware design, the bus systems for the 6502 and 6800 appear to be the same, but in fact they are not directly compatible. Generally the support chips for the 6800 can readily be used with a 6502 CPU and the reverse is also true, although in some cases additional external logic may be required. The instruction sets may also appear to be similar, but are totally incompatible as far as machine code is concerned.

Some of the wide popularity of the 6502 series can be attributed to their use in such popular personal computer systems as the CBM PET and the Apple II.

Prime manufacturer

MOS Technology Inc., which is a subsidiary of Commodore Business Machines (CBM).

Devices available

MCS6502	Basic type 65k address on-chip clock
MCS6512	As 6502 but external clock
MCS6503	4k address range on-chip clock
MCS6504	8k address range on-chip clock, no NMI
MCS6505	4k address range on-chip clock, no NMI
MCS6506	4k address range on-chip clock, no NMI
MCS6507	8k address range on-chip clock, no interrupts
MCS6513	As 6503 but external clock
MCS6514	As 6504 but external clock
MCS6515	As 6505 but external clock

Alternative source devices

Rockwell

R6502, R6503, R6504, R6505, R6506, R6507
R6512, R6513, R6514, R6515

Synertek

SY6502, SY6503, SY6504, SY6505, SY6506, SY6507
SY6512, SY6513, SY6514, SY6515

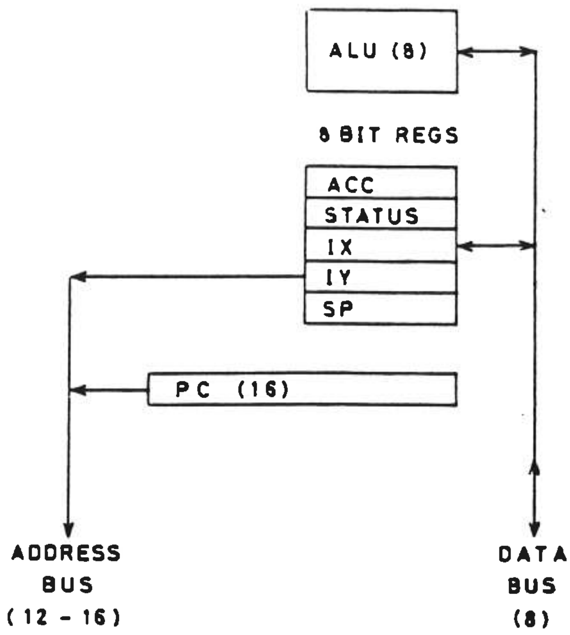
EMM-Semi

6502, 6503, 6504, 6505, 6506, 6507
6512, 6513, 6514, 6515

Note that all of the 6502 series types are available with various clock speed options, with versions for 1 MHz, 2 MHz and 3 MHz maximum clock frequency.

Architecture

If the architecture diagram for the 6502 (fig. 3.17) is compared with that of the 6800 it will be seen that the 6502 is similar in design to the 6800, though rather less complex.



Only one 8-bit accumulator is provided, compared with the two accumulators of the 6800, and this handles all arithmetic and logic operations via the ALU. Although slightly less flexible when dealing with 16-bit numbers, the single 8-bit accumulator is perfectly adequate for all normal computing requirements.

An 8-bit status register privides flags for zero, minus, carry and overflow results of operations, and for the interrupt, break and decimal modes.

Unlike the 6800 the 6502 has two 8-bit index registers rather than a single 16-bit index register. This limits the index range to 256 but provides much greater flexibility in dealing with data tables.

The stack pointer of the 6502 has only 8 bits and the stack is always located within page 1 of the memory map. It is possible to have any stack length up to 256 bytes and a number of separate stacks may be set up within page 1. This is slightly less flexible than the 6800, where the stacks may be set up anywhere in memory, but is perfectly adequate.

As with the 6800 there are no general purpose registers provided in the 6502, since it uses general memory locations for this purpose. Similarly all input–output devices will be treated simply as memory locations by the processor.

In the 6502 and 6512 the program counter register is 16 bits wide, allowing up to 65k of memory to be addressed. In other devices of the series the program counter length is cut to 12 or 13 bits, allowing either 4k or 8k of address space.

Like the 6800 the bus system of the 6502 comprises an 8-bit bidirectional data bus, a 16-bit address bus and some control signals. All operations are controlled by a 2-phase clock, and memory access is made on phase 2 of each cycle of the clock. Internal operations occur during phase 1.

The basic memory map for the 6502 is:

FFFF	
FFFA	Vectors for int. and reset
FFF9	
	Main user space
0200	
01FF	
	Stack area
0100	
00FF	
	Zero page
0000	

Package

The 6502 and 6512 are supplied in 40-pin dual in line
All other types use a 28-pin dual in line package
All types use a plastic encapsulation

Pin connections

6502 and 6512

1	V_{ss}	21	V_{ss}
2	RDY	22	AB12
3	ϕ1	23	AB13
4	IRQ	24	AB14
5	No conn. (6502) V_{ss} (6512)	25	AB15
6	NMI	26	DB7
7	SYNC	27	DB6
8	V_{cc}	28	DB5
9	AB0	29	DB4
10	AB1	30	DB3
11	AB2	31	DB2
12	AB3	32	DB1
13	AB4	33	DB0
14	AB5	34	R/W
15	AB6	35	No conn.
16	AB7	36	No conn. (6502) DBE (6512)
17	AB8	37	ϕ0 (6502) ϕ2 (6512)
18	AB9	38	S.O.
19	AB10	39	ϕ2 OUT
20	AB11	40	RESET

6503 28 pin

1	RESET	15	AB9
2	V_{ss}	16	AB10
3	IRQ	17	AB11
4	NMI	18	DB7
5	V_{cc}	19	DB6
6	AB0	20	DB5
7	AB1	21	DB4
8	AB2	22	DB3
9	AB3	23	DB2
10	AB4	24	DB1
11	AB5	25	DB0
12	AB6	26	R/W
13	AB7	27	ϕ0 IN
14	AB8	28	ϕ2 OUT

6504/6507 28 pin

1	RESET	15	AB10
2	V_{ss}	16	AB11
3	IRQ (6504) RDY (6507)	17	AB12
4	V_{cc}	18	DB7
5	AB0	19	DB6
6	AB1	20	DB5
7	AB2	21	DB4
8	AB3	22	DB3
9	AB4	23	DB2
10	AB5	24	DB1
11	AB6	25	DB0
12	AB7	26	R/W
13	AB8	27	ϕ0 IN
14	AB9	28	ϕ2 OUT

Other 28-pin types have same AB and DB connections as above, according to whether they have 12 or 13-bit address. Other pins are different and manufacturer's data sheets should be consulted.

Signal functions

DB0 – DB7	Bidirectional data bus
AB0 – AB15	Address bus (output)
V_{ss}, V_{cc}	Power supplies
R/W	Read–write (low = write)
IRQ, NMI	Interrupt req. inputs active low
RDY	Ready input used to halt CPU
SYNC	Output (1 during instruction fetch)
RESET	Reset input (active low)
S.O.	Set overflow input
ϕ0, ϕ1, ϕ2	Clock signals
DBE	Data bus enable (active high)

Power requirements

$V_{ss} = 0$ V
$V_{cc} = +5$ V $\pm 5\%$
Power dissipation 700 – 800 mW

Signal levels

Inputs are TTL compatible 300 μA loading
Outputs will drive one TTL load
Data bus is tri-state

Input–output

The 6502 series treat all input–output as memory locations, data being presented or accepted via the data bus.

Interrupt facilities

The 6502 provides both maskable (IRQ) and non-maskable (NMI) interrupts. There is also a software interrupt facility using the BRK instruction. On an interrupt execution the program counter and status register are pushed to the stack. These are restored by the RTI instruction at the end of the interrupt routine. BRK is the same as IRQ, but not maskable and sets a flag bit in the status register. Interrupt vector addresses are stored at the top of memory as shown:

FFFF	IRQ vector	(MSB)
FFFE	IRQ vector	(LSB)
FFFD	Reset vector	(MSB)
FFFC	Reset vector	(LSB)
FFFB	NMI vector	(MSB)
FFFA	NMI vector	(LSB)

Reset causes a reset sequence within the CPU and the instruction address is obtained from FFFC/FFFD.

Multilevel interrupt operation is readily achieved and priorities may be dealt with either by polling software or by external hardware.

Instruction set

The 6502 instruction set contains 52 different instructions, and at first sight may appear to be very similar to that for the 6800 series microprocessors. Instructions may have one, two or three bytes.

Arithmetic and logic

Addition and subtraction with carry or borrow are provided using the 8-bit accumulator. A decimal mode also allows addition and subtraction of BCD format numbers. There are no complement or negate instructions and the accumulator cannot be directly incremented or decremented, although memory locations can.

AND, OR and EXCLUSIVE OR operations can be carried out between accumulator and memory. There are also shift and rotate left and right instructions for both memory and accumulator.

Branch and jump

A useful series of conditional branch instructions is provided, although this is not as extensive as those on the 6800. Status register bits may be set and reset by program. Tests for zero, negative, carry and overflow are provided.

Only unconditional jump and jump to subroutine are available. A subroutine jump automatically stores the return address on the stack.

Register and transfer operations

Data can readily be transferred between the A accumulator, the X and Y index registers and the stack pointer, or memory. Push and pull instructions allow data from the accumulator or status register to be transferred to the stack. Both index registers may be incremented or decremented.

Memory or accumulator words may be tested bit by bit if desired.

Timing

Like the 6800, the 6502 series uses a 2-phase processor clock, and all memory access is carried out during $\phi 2$ clock cycles. Most instructions take 2, 3 or 4 clock cycles and may use 1, 2 or 3 bytes of machine code.

The standard parts operate with a 1 MHz clock, giving instruction execution times of some 2 – 4 μs. Special high speed parts are available with clock frequencies of 2 or 3 MHz. These are coded with suffix A (6502A) for 2 MHz operation or suffix B (6502B) for 3 MHz operation.

Types 6502 to 6507 have on-chip clock phase generators but need an external crystal oscillator to provide the $\phi 0$ input, whilst types 6512 to 6515 require an external 2-phase non-overlapping clock signal applied to the $\phi 1$ and $\phi 2$ clock inputs.

Support devices

A wide range of support devices is available for the 6502 series microprocessors. Some of these are:

6520 PIA	Two 8-bit bidirectional programmable ports (identical to 6820)
6522	Versatile interface adapter (VIA) 2 × 8-bit ports as in 6520, plus 2 × 16-bit interval timers and a serial I/O facility
6530	1k ROM, 64-byte RAM, 2 × 8-bit parallel ports plus an 8-bit interval timer.
6531	2k ROM, 128 byte RAM, 2 × 8-bit parallel I/O, serial I/O and a 16-bit timer/counter
6532	128-byte RAM, 2 × 8-bit parallel I/O ports, 8-bit timer
6541	Keyboard/display controller
6545	Raster scan CRT controller
6551	Asynchronous serial I/O
6591	Floppy disk controller

The 6502 series may also be used with most of the 6800 series support devices. Some care may be needed with address decoding, however, since the 6500 has its lower address byte in the lower memory location whilst the 6800 stores its addresses in memory with the high address byte first.

Development aids

MOS Technology

KIM1	Stand alone board with keypad and LED displays

Appendix

6502 instruction set

ADC

Add memory to accumulator with carry

Operation: A+M+C—A, C

N Z C I D V
/ / / - - /

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	ADC	#Oper	69	2	2
Zero Page	ADC	Oper	65	2	3
Zero Page, X	ADC	Oper, X	75	2	4
Absolute	ADC	Oper	6D	3	4
Absolute, X	ADC	Oper, X	7D	3	4*
Absolute, Y	ADC	Oper, Y	79	3	4*
(Indirect, X)	ADC	(Oper, X)	61	2	6
(Indirect), Y	ADC	(Oper), Y	71	2	5*

* Add 1 if page boundary is crossed.

AND

AND memory with accumulator

Operation: A∧M—A

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	AND	#Oper	29	2	2
Zero Page	AND	Oper	25	2	3
Zero Page, X	AND	Oper, X	35	2	4
Absolute	AND	Oper	2D	3	4
Absolute, X	AND	Oper, X	3D	3	4*
Absolute, Y	AND	Oper, Y	39	3	4*
(Indirect, X)	AND	(Oper, X)	21	2	6
(Indirect), Y	AND	(Oper), Y	31	2	5*

* Add 1 if page boundary is crossed.

ASL

Shift Left One Bit (Memory or Accumulator)

Operation: C—76543210—Ø

N Z C I D V
/ / / - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Accumulator	ASL	A	ØA	1	2
Zero Page	ASL	Oper	Ø6	2	5
Zero Page, X	ASL	Oper, X	16	2	6
Absolute	ASL	Oper	ØE	3	6
Absolute, X	ASL	Oper, X	1E	3	7

BCC

Branch on Carry Clear

Operation: Branch on C=Ø

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BCC	Oper	9Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BCS

Branch on carry set

Operation: Branch on C=1

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BCS	Oper	BØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to next page.

BEQ

Branch on result zero

Operation: Branch on Z=1

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BEQ	Oper	FØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to next page.

BIT
Test bits in memory with accumulator

Operation: $a \wedge M$, M_7—N, M_6—V

N Z C I D V
M_7 / - - - M_6

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Zero Page	BIT	Oper	24	2	3
Absolute	BIT	Oper	2C	3	4

BMI
Branch on result minus

Operation: Branch on N=1

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BMI	Oper	3Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BNE
Branch on result not zero

Operation: Branch on Z=O

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BNE	Oper	DØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BPL
Branch on result plus

Operation: Branch on N=Ø

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BPL	Oper	1Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BRK
Force Break

Operation: Forced Interrupt PC+2|P|

N Z C I D V
- - - * - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	BRK	ØØ	1	7

* A BRK command cannot be masked by setting I.

BVC
Branch on overflow clear

Operation: Branch on V=O

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BVC	Oper	5Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BVS
Branch on overflow set

Operation: Branch on V=1

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Relative	BVS	Oper	7Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

CLC
Clear carry flag

Operation: Ø—C

N Z C I D V
- - Ø - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	CLC	18	1	2

CLD
Clear decimal mode

Operation: Ø—D

N Z C I D V
- - - - Ø -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	CLD	D8	1	2

CLI

Clear interrupt disable bit

Operation: Ø—I

N Z C I D V
- - - Ø - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	CLI	58	1	2

CPX

Compare memory and index X

Operation: X—M

N Z C I D V
/ / / - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Immediate	CPX #Oper	EØ	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

CPY

Compare memory and index Y

Operation: Y—M

N Z C I D V
/ / / - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Immediate	CPY #Oper	CØ	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

CLV

Clear overflow flag

Operation: Ø—V

N Z C I D V
- - - - - Ø

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

CMP

Compare memory and accumulator

Operation: A—M

N Z C I D V
/ / / - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Immediate	CMP #Oper	C9	2	2
Zero Page	CMP Oper	C5	2	3
Zero Page, X	CMP Oper, X	D5	2	4
Absolute	CMP Oper	CD	3	4
Absolute, X	CMP Oper, X	DD	3	4*
Absolute, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper), Y	D1	2	5*

* Add 1 if page boundary is crossed.

DEC

Decrement memory by one

Operation: M-1—M

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

DEX

Decrement index X by one

Operation: X-1X

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

DEY

Decrement index Y by one

Operation: Y-1—Y

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	DEY	88	1	2

EOR

Exclusive—Or memory with accumulator

Operation: A-M—A

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	EOR	#Oper	49	2	2
Zero Page	EOR	Oper	45	2	3
Zero Page, X	EOR	Oper, X	55	2	4
Absolute	EOR	Oper	4D	3	4
Absolute, X	EOR	Oper, X	5D	3	4*
Absolute, Y	EOR	Oper, Y	59	3	4*
(Indirect, X)	EOR	(Oper, X)	41	2	6
(Indirect), Y	EOR	(Oper), Y	51	2	5*

* Add 1 if page boundary is crossed.

INC

Increment memory by one

Operation: M+1—M

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Zero Page	INC	Oper	E6	2	5
Zero Page, X	INC	Oper, X	F6	2	6
Absolute	INC	Oper	EE	3	6
Absolute, X	INC	Oper, X	FE	3	7

INX

Increment index X by one

Operation: X+1—X

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	INX	E8	1	2

INY

Increment index Y by one

Operation: Y+1—Y

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	INY	C8	1	2

JMP

Jump to new location

Operation: (PC+1)—PCL
(PC+2)—PCH

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Absolute	JMP	Oper	4C	3	3
Indirect	JMP	(Oper)	6C	3	5

JSR

Jump to new location saving return address

Operation: PC+2↓, (PC+1)—PCL
(PC+2)—PCH

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Absolute	JSR	Oper	20	3	6

LDA

Load accumulator with memory

Operation: M—A

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	LDA	#Oper	A9	2	2
Zero Page	LDA	Oper	A5	2	3
Zero Page, X	LDA	Oper, X	B5	2	4
Absolute	LDA	Oper	AD	3	4
Absolute, X	LDA	Oper, X	BD	3	4*
Absolute, Y	LDA	Oper, Y	B9	3	4*
(Indirect, X)	LDA	(Oper, X)	A1	2	6
(Indirect), Y	LDA	(Oper), Y	B1	2	5*

* Add 1 if page boundary is crossed.

LDX

Load index X with memory

Operation: M—X

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	LDX	#Oper	A2	2	2
Zero Page	LDX	Oper	A6	2	3
Zero Page, Y	LDX	Oper, Y	B6	2	4
Absolute	LDX	Oper	AE	3	4
Absolute, Y	LDX	Oper, Y	BE	3	4*

* Add 1 when page boundary is crossed.

LDY

Load index Y with memory

Operation: M—Y

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Immediate	LDY #Oper	AØ	2	2
Zero Page	LDY Oper	A4	2	3
Zero Page, X	LDY Oper, X	B4	2	4
Absolute	LDY Oper	AC	3	4
Absolute, X	LDY Oper, X	BC	3	4*

* Add 1 when page boundary is crossed.

ORA

OR memory with accumulator

Operation: AVM—A

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Immediate	ORA #Oper	Ø9	2	2
Zero Page	ORA Oper	Ø5	2	3
Zero Page, X	ORA Oper, X	15	2	4
Absolute	ORA Oper	ØD	3	4
Absoulte, X	ORA Oper, X	1D	3	4*
Absolute, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	Ø1	2	6
(Indirect), Y	ORA (Oper), Y	11	2	5*

* Add 1 on page crossing

PHA

Push accumulator on stack

Operation: A↓

N Z C I D V
- - - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	PHA	48	1	3

LSR

Shift right one bit (memory or accumulator)

Operation: Ø—7 6 5 4 3 2 1 Ø—C

N Z C I D V
Ø / / - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

NOP

No operation

Operation: No operation (2 cycles)

N Z C I D V
- - - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

PHP

Push processor status on stack

Operation: P↓

N Z C I D V
- - - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	PHP	Ø8	1	3

PLA

Pull accumulator from stack

Operation: A↑

N Z C I D V
/ / - - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	PLA	68	1	4

PLP

Pull processor status from stack

Operation: P↑

N Z C I D V
From Stack

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Implied	PLP	28	1	4

ROL

Rotate one bit left (memory or accumulator)

M or A

Operation: 7 6 5 4 3 2 1 Ø—C

N Z C I D V
/ / / - - -

Addressing mode	Assembly language form	OP Code	No. Bytes	No. Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

ROR

Rotate one bit right (memory or accumulator)

M or A

Operation: C→7 6 5 4 3 2 1 0

N Z C I D V
/ / / - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Accumulator	ROR	A	6A	1	2
Zero Page	ROR	Oper	66	2	5
Zero Page, X	ROR	Oper, X	76	2	6
Absolute	ROR	Oper	6E	3	6
Absolute, X	ROR	Oper, X	7E	3	7

RTI

Return from interrupt

Operation: P↑ PC↑

N Z C I D V
From Stack

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	RTI	40	1	6

RTS

Return from subroutine

Operation: PC↑, PC+1→PC

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	RTS	60	1	6

SBC

Subtract memory from accumulator with borrow

Operation: A − M − C→A
Note: C = Borrow

N Z C I D V
/ / / - - /

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Immediate	SBC	#Oper	E9	2	2
Zero Page	SBC	Oper	E5	2	3
Zero Page, X	SBC	Oper, X	F5	2	4
Absolute	SBC	Oper	ED	3	4
Absolute, X	SBC	Oper, X	FD	3	4*
Absolute, Y	SBC	Oper, Y	F9	3	4*
(Indirect, X)	SBC	(Oper, X)	E1	2	6
(Indirect), Y	SBC	(Oper), Y	F1	2	5*

* Add 1 when page boundary is crossed.

SEC

Set carry flag

Operation: 1→C

N Z C I D V
- - 1 - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	SEC	38	1	2

SED

Set decimal mode

Operation: 1→D

N Z C I D V
- - - - 1 -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	SED	F8	1	2

SEI

Set interrupt disable status

Operation: 1→I

N Z C I D V
- - - 1 - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	SEI	78	1	2

STA

Store accumulator in memory

)peration: A→M

N Z C I D V
- - - - - -

ddressing ode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
ero Page	STA	Oper	85	2	3
ero Page, X	STA	Oper, X	95	2	4
bsolute	STA	Oper	8D	3	4
bsolute, X	STA	Oper, X	9D	3	5
bsolute, Y	STA	Oper, Y	99	3	5
ndirect, X)	STA	(Oper, X)	81	2	6
ndirect), Y	STA	(Oper), Y	91	2	6

STX

Store index X in memory

Operation: X—M

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Zero Page	STX	Oper	86	2	3
Zero Page, Y	STX	Oper, Y	96	2	4
Absolute	STX	Oper	8E	3	4

STY

Store index Y in memory

Operation: Y—M

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*		*OP Code*	*No. Bytes*	*No. Cycles*
Zero Page	STY	Oper	84	2	3
Zero Page, X	STY	Oper, X	94	2	4
Absolute	STY	Oper	8C	3	4

TAX

Transfer accumulator to index X

Operation: A—X

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TAX	AA	1	2

TAY

Transfer accumulator to index Y

Operation: A—Y

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TAY	A8	1	2

TYA

Transfer index Y to accumulator

Operation: Y—A

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TYA	98	1	2

TSX

Transfer stack pointer to index X

Operation: S—X

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TSX	BA	1	2

TXA

Transfer index X to accumulator

Operation: X—A

N Z C I D V
/ / - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TXA	8A	1	2

TXS

Transfer index X to stack pointer

Operation: X—S

N Z C I D V
- - - - - -

Addressing mode	*Assembly language form*	*OP Code*	*No. Bytes*	*No. Cycles*
Implied	TXS	9A	1	2

ØØ –BRK
Ø1 –ORA—(Indirect, X)
Ø2 –Future Expansion
Ø3 –Future Expansion
Ø4 –Future Expansion
Ø5 –ORA—Zero Page
Ø6 –ASL—Zero Page
Ø7 –Future Expansion
Ø8 –PHP
Ø9 –ORA—Immediate
ØA –ASL—Accumulator
ØB –Future Expansion
ØC –Future Expansion
ØD –ORA—Absolute
ØE –ASL—Absolute
ØF –Future Expansion
1Ø –BPL
11 –ORA—(Indirect), Y
12 –Future Expansion
13 –Future Expansion
14 –Future Expansion
15 –ORA—Zero Page, X
16 –ASL—Zero Page, X
17 –Future Expansion
18 –CLC
19 –ORA—Absolute, Y
1A –Future Expansion
1B –Future Expansion
1C –Future Expansion
1D –ORA—Absolute, X
1E –ASL—Absolute, X
1F –Future Expansion
2Ø –JSR
21 –AND—(Indirect, X)
22 –Future Expansion
23 –Future Expansion
24 –BIT—Zero Page
25 –AND—Zero Page

26 -ROL—Zero Page
27 -Future Expansion
28 -PLP
29 -AND—Immediate
2A -ROL—Accumulator
2B -Future Expansion
2C -BIT—Absolute
2D -AND—Absolute
2E -ROL—Absolute
2F -Future Expansion
3Ø -BMI
31 -AND—(Indirect), Y
32 -Future Expansion
33 -Future Expansion
34 -Future Expansion
35 -AND—Zero Page, X
36 -ROL—Zero Page, X
37 -Future Expansion
38 -SEC
39 -AND—Absolute, Y
3A -Future Expansion
3B -Future Expansion
3C -Future Expansion
3D -AND—Absolute, X
3E -ROL—Absolute, X
3F -Future Expansion
4Ø -RTI
41 -EOR—(Indirect, X)
42 -Future Expansion
43 -Future Expansion
44 -Future Expansion
45 -EOR—Zero Page
46 -LSR—Zero Page
47 -Future Expansion
48 -PHA
49 -EOR—Immediate
4A -LSR—Accumulator
4B -Future Expansion
4C -JMP—Absolute
4D -EOR—Absolute
4E -LSR—Absolute
4F -Future Expansion
5Ø -BVC
51 -EOR—(Indirect), Y
52 -Future Expansion
53 -Future Expansion
54 -Future Expansion
55 -EOR—Zero Page, X
56 -LSR—Zero Page, X
57 -Future Expansion
58 -CLI
59 -EOR—Absolute, Y
5A -Future Expansion
5B -Future Expansion
5C -Future Expansion
5D -EOR—Absolute, X
5E -LSR—Absolute, X
5F -Future Expansion
6Ø -RTS
61 -ADC—(Indirect, X)
62 -Future Expansion
63 -Future Expansion
64 -Future Expansion
65 -ADC—Zero Page
66 -ROR—Zero Page
67 -Future Expansion
68 -PLA
69 -ADC—Immediate
6A -ROR—Accumulator
6B -Future Expansion
6C -JMP—Indirect
6D -ADC—Absolute
6E -ROR—Absolute
6F -Future Expansion
7Ø -BVS
71 -ADC—(Indirect), Y
72 -Future Expansion
73 -Future Expansion
74 -Future Expansion
75 -ADC—Zero Page, X
76 -ROR—Zero Page, X
77 -Future Expansion
78 -SEI
79 -ADC—Absolute, Y
7A -Future Expansion
7B -Future Expansion
7C -Future Expansion
7D -ADC—Absolute, X
7E -ROR—Absolute, X
7F -Future Expansion
8Ø -Future Expansion
81 -STA—(Indirect, X)
82 -Future Expansion
83 -Future Expansion
84 -STY—Zero Page
85 -STA—Zero Page
86 -STX—Zero Page
87 -Future Expansion
88 -DEY
89 -Future Expansion
8A -TXA
8B -Future Expansion
8C -STY—Absolute
8D -STA—Absolute
8E -STX—Absolute
8F -Future Expansion
9Ø -BCC
91 -STA—(Indirect), Y
92 -Future Expansion
93 -Future Expansion
94 -STY—Zero Page, X
95 -STA—Zero Page, X
96 -STX—Zero Page, Y
97 -Future Expansion
98 -TYA
99 -STA—Absolute, Y
9A -TXS
9B -Future Expansion
9C -Future Expansion
9D -STA—Absolute, X
9E -Future Expansion
9F -Future Expansion
AØ -LDY—Immediate
A1 -LDA—(Indirect, X)
A2 -LDX—Immediate
A3 -Future Expansion
A4 -LDY—Zero Page
A5 -LDA—Zero Page
A6 -LDX—Zero Page
A7 -Future Expansion
A8 -TAY
A9 -LDA—Immediate
AA -TAX
AB -Future Expansion
AC -LDY—Absolute
AD -LDA—Absolute
AE -LDX—Absolute
AF -Future Expansion
BØ -BCS
B1 -LDA—(Indirect), Y
B2 -Future Expansion
B3 -Future Expansion
B4 -LDT—Zero Page, X
B5 -LDA—Zero Page, X
B6 -LDX—Zero Page, Y
B7 -Future Expansion
B8 -CLV
B9 -LDA—Absolute, Y
BA -TSX
BB -Future Expansion
BC -LDY—Absolute, X
BD -LDA—Absolute, X
BE -LDX—Absolute, Y
BF -Future Expansion
CØ -CPY—Immediate
C1 -CMP—(Indirect, X)
C2 -Future Expansion
C3 -Future Expansion
C4 -CPY—Zero Page
C5 -CMP—Zero Page
C6 -DEC—Zero Page
C7 -Future Expansion
C8 -INY
C9 -CMP—Immediate
CA -DEX
CB -Future Expansion
CC -CPY—Absolute
CD -CMP—Absolute
CE -DEC—Absolute
CF -Future Expansion
DØ -BNE
D1 -CMP—(Indirect), Y
D2 -Future Expansion
D3 -Future Expansion
D4 -Future Expansion
D5 -CMP—Zero Page, X

D6 –DEC—Zero Page, X
D7 –Future Expansion
D8 –CLD
D9 –CMP—Absolute, Y
DA–Future Expansion
DB–Future Expansion
DC–Future Expansion
DD–CMP—Absolute, X
DE–DEC—Absolute, X
DF–Future Expansion
EØ –CPX—Immediate
E1 –SBC—(Indirect, X)
E2 –Future Expansion
E3 –Future Expansion
E4 –CPX—Zero Page
E5 –SBC—Zero Page
E6 –INC—Zero Page
E7 –Future Expansion
E8 –INX
E9 –SBC—Immediate
EA–NOP
EB –Future Expansion
EC–CPX—Absolute
ED–SBC—Absolute
EE –INC—Absolute
EF –Future Expansion
FØ –BEQ
F1 –SBC—(Indirect), Y
F2 –Future Expansion
F3 –Future Expansion
F4 –Future Expansion
F5 –SBC—Zero Page, X
F6 –INC—Zero Page, X
F7 –Future Expansion
F8 –SED
F9 –SBC—Absolute, Y
FA –Future Expansion
FB –Future Expansion
FC –Future Expansion
FD–SBC—Absolute, X
FE –INC—Absolute, X
FF –Future Expansion