5

ADDRESSING MODES

So far we have met two addressing modes. One of these is absolute addressing as in

```
LDA addr
```

which, when executed, loads the accumulator with the contents of the location whose address is 'addr'. The other is immediate addressing as in

```
LDA #&81
```

which, when executed, loads the accumulator with the actual value &81.

However, other addressing modes exist and one of the most important, 'indexed addressing', is introduced here prior to a summary exposition of all the addressing modes available to the 6502 processor.

5.1 Indexed addressing

In this addressing mode one of the index registers (X or Y) is added to the address as an offset which gives the precise location for the stored data. For example, we can write:

```
LDA addr, X
```

If X contains zero this instruction will behave just like 'LDA addr'. However, if X contains 1 it will load the accumulator with the contents of 'one location further on from addr'. Since X can contain any value from 0 to 255, the instruction 'LDA addr,X' gives you access to 256 different memory locations. If you are familiar with BASIC's byte vectors you can think of 'addr' as the base of a vector, and of X as containing the subscript, e.g.

```
addr?7 = 12
```

is equivalent to

```
LDA #12
LDX #7
STA addr, X
```

5.2 String types

Two examples of the use of indexed addressing are given below, both involving strings. There are two string types available for use in BASIC and assembler; ATOM strings and Microsoft strings. An ATOM string is a string of characters terminated by a RETURN character. The name which identifies the string is preceded by a dollar ($) sign and the strings can be easily set up in BASIC, e.g.

```
$name = "Fred"
```

ATOM strings must have an area of memory set aside for them. This can be done, as in the examples, by using a DIM statement. The characters making up the string are then stored in the location identified by the name of the string. This is very useful as the address of each character is then also known.

A Microsoft string is a string of characters preceded by a byte which gives the length of the string. In this case, the name of the string has a dollar ($) sign after it. It is more flexible than the ATOM string because it can contain RETURN characters. Its disadvantage is that all the characters making up the string are stored in locations chosen by BASIC, hence the addresses of these are not known.

Example - print inverted-case string

The following program uses indexed addressing to

print out a string of characters terminated by a carriage return (which is represented in the memory by &D), swapping case as it prints out each character.

```
 10 DIM string 256, code 100
 20 oswrch = &FFEE
 30 FOR pass = 0 TO 3 STEP 3
 40 P% = code
 50[OPT pass
 60.enter
 70 LDX #0             Set index to zero
 80.loop
 90   LDA string,X     Get characters from string
100   CMP #&D          Is it end of string ?
110   BEQ return       If so, end
120   EOR #&20         Else invert case bit
130   JSR oswrch       Print it
140   INX              Increment index
150   BNE Loop         If string Longer than 256
160.return
170   RTS              then end anyway
180]
190 NEXT pass
200 END
```

Assemble the program by typing RUN, and then try the program by entering:

```
$string = "Test String"
CALL enter
```

Example-- index subroutine

Another useful operation, easily performed in a machine-code routine, is looking up a character in a string and returning its position in that string. The following subroutine reads in a character, using a call to the OSRDCH read-character routine, and saves in ' ?found' the position of the first occurrence of that character in ' $target'.This is exactly the same as the BASIC ' ?found =INSTR("ABCDEFGH",GET$)' .

```
 0 REM Index Rooutine
10 DIM target 25,P% 100
20 osrdch=&FFE0 : $target="ABCDEFGH" : found = &70
```

```
 30[
 40.enter
 50 JSR osrdch              Get character
 60 LDX#(LEN($target)-1)    Length of string
 70. Loop
 80 CMP target,X            Compare character
 90 BEQ match               Got a match
100 DEX                     Try again
110 CPX #255                Until end of string
120 BNE loop
130.match
140 INX                     The position in the
150 STX found               string is stored
160 RTS                     Return
170]
180 END
```

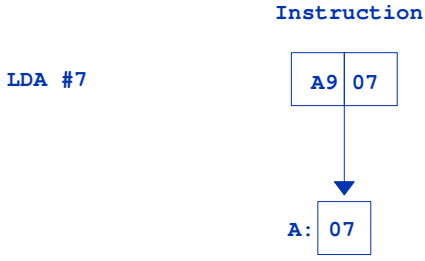
The routine is entered at ' .enter'and as it stands it looks for one of the letters A to H.

5.3 Summary of addressing modes

The following sections summarise all addressing modes that are available on the 6502, some of which have been met already.

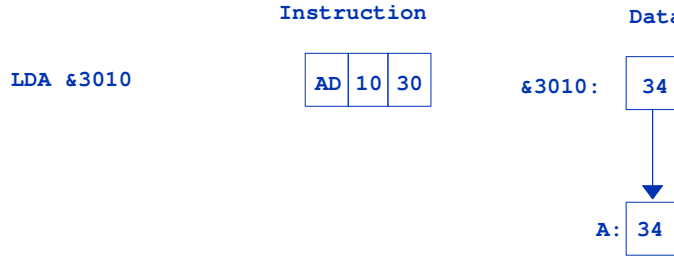
Immediate addressing

Use immediate addressing when the data for an instruction is known at the time of writing the program. In this mode the second byte of the instruction contains the actual eight-bit data to be used by the instruction. The ' #'symbol denotes an immediate operand.



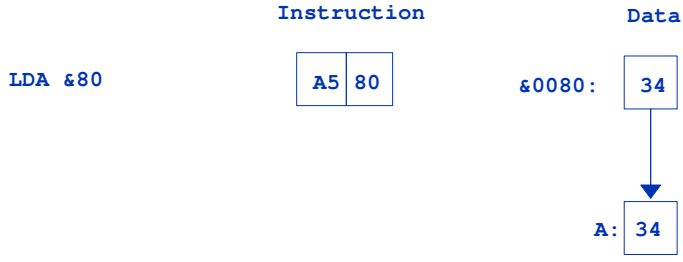

```
Examples: LDA #value
CPY #flag + 2
```

Absolute addressing
Use absolute addressing when the effective address, to be used by the instruction, is known at the time the program is being written. In this mode the two bytes following the op-code contain the 16-bit effective address to be used by the instruction, the low byte being given first, followed by the high byte.




```
Example: LDA address
```

Zero page addressing
Zero page addressing is a subset of absolute addressing. They are similar in that the instruction specifies the effective address to be used; the difference between them is that in absolute addressing the address used can be anywhere, whereas in zero page addressing the address is in zero page, i.e. from &0000 to &00FF. Hence this address is only one byte rather than two. The assembler will automatically produce zero-page instructions.

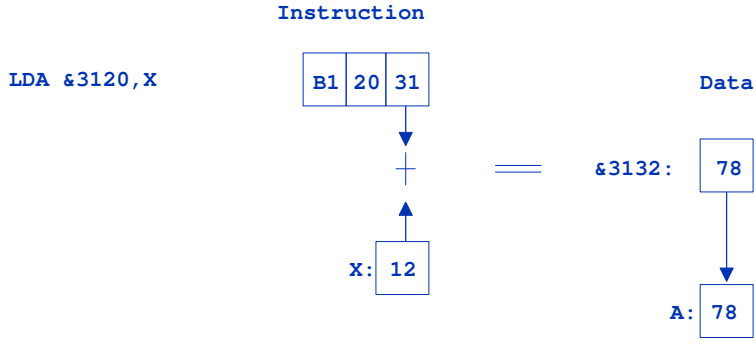



```
Examples: JSR Loop
ASL &9A
```

Indexed addressing
Indexed addressing is used to access a table of memory locations by specifying them in terms of an offset from a base address. The base address is known at the time that the program is written; the offset, which is provided in one of the index registers, can be calculated by the program.

In all indexed addressing modes one of the eight-bit index registers, X and Y, is used in order to calculate the effective address to be used by the instruction. Five different indexed addressing modes are available, and are listed below.

Absolute indexed addressing
The simplest indexed addressing mode is absolute indexed addressing. In this mode the two bytes following the instruction specify a 16-bit address which is to be added to one of the index registers to form the effective address to be used by the instruction.

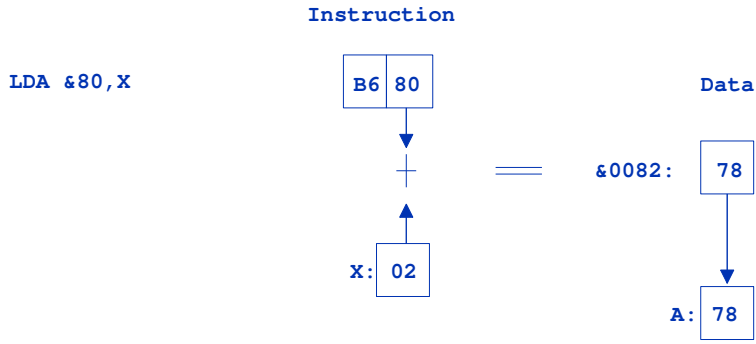



```
Examples: LDA tabLe,X
LDX palette,Y
INC score,X
```

Zero,X indexed addressing
Here, the second byte of the instruction specifies an eight-bit address, which is added to the X-register to

give a zero-page address to be used by the instruction.

Note that in the case of the LDX instruction a ' zero,Y'addressing mode is provided instead of the ' zero,X' mode.




```
Examples: LSR &80,X
LDX addr,Y (where addr+Y is in zero page)
```

Indirect addressing
It is sometimes necessary to use an address which is actually computed when the program runs, rather than being an offset from a base address or a constant address. In this case indirect addressing is used.

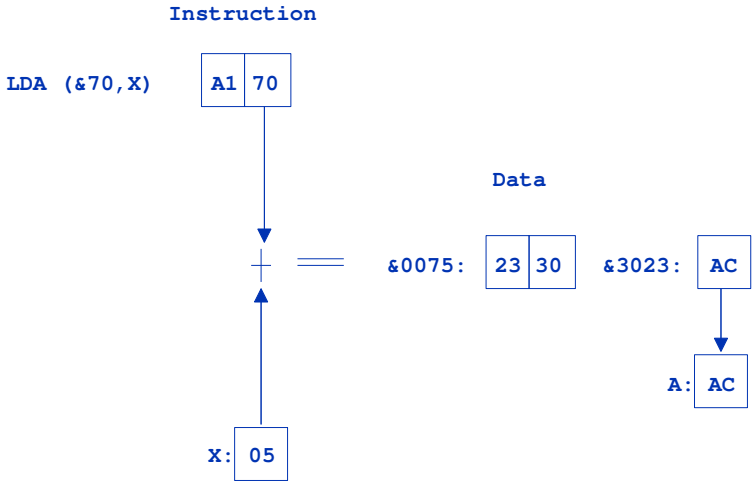
Indirect addressing is distinct from direct addressing (i.e. absolute, indexed, etc) in that the address specified after the mnemonic is used to refer to a location where the final address will be found. Thus the machine does not go directly to the address, but instead it goes indirectly, via the address given.

The indirect mode of addressing is available for the JMP instruction. Thus control can be transferred to an address calculated at the time the program is run.

```
Examples: JMP (&2800)
JMP (addr)
```

For the dual-operand instructions ADC, AND, CMP, EOR, LDA, ORA, SBC and STA, two different modes of indirect addressing are provided: preindexed indirect, and post-indexed indirect. Pure indirect addressing can be obtained, using either mode, by first setting the respective index register to zero.

Pre-indexed indirect addressing



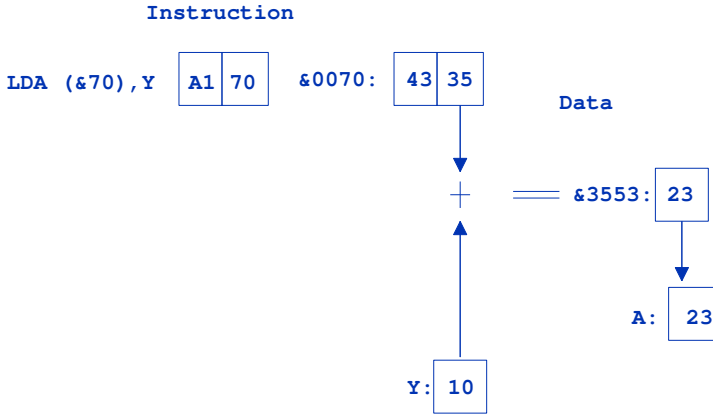
Examples: STA (zerotable,X)
EOR (&60,X)

This mode of addressing is used when a table of effective addresses is provided in zero page; the X index register is used as a pointer to select one of these addresses from the table.

In pre-indexed indirect addressing the second byte of the instruction is added to the X register to give an address in zero page. The two bytes at this zero-page address are then used as the effective address for the instruction.

Post-indexed indirect addressing
This indexed addressing mode is like the absolute,X or absolute,Y indexed addressing modes, except that in this case the base address of the table is provided in zero page, rather than in the bytes following the instruction.

In post-indexed indirect addressing the second byte of the instruction specifies a zero-page address. The two bytes at this address are added to the Y index register to give a 16-bit address which is then used as the effective address for the instruction.



Examples: ADC (&66),Y
CMP (pointer),Y

This last addressing mode is very useful. An example of its use is given in the program below, which will only work on a machine without a second processor attached. It clears the screen.

```
10 DIM MC% 100
20 addr=&70
```

```
 30 FOR pass =0 TO 2 STEP 2
 40 P%=MC%
 50[OPT pass
 60.cls
 70 LDA #&58        High byte of start address
 80 STA addr+1
 90 LDY #0          Low byte of address,
100 STY addr        and value to write to screen
110 TYA             Put zero (black) into A
120.clsloop
130 STA (addr),Y    Store zero
140 INY
150 BNE clsloop     do 256 times
160 INC addr+1      Increment hi byte of adress
170 LDX addr+1      Set status register to addr
180 BPL clsloop     Compare with top of RAM
190 RTS
200]
210 NEXT pass
220 MODE 4
230 COLOUR 129
240 CLS             White out screen
250 A=GET           Wait for a key
260 CALL cls        Black out screen
270 COLOUR 128
280 END
```