6502 编程大奥秘（前三章汇编部分）

修正版 维京猎人

6502寄存器

FC 技术书籍 2008-10-06 00:01 阅读 85 评论 0

本文来自文曲星的《6502 编程大奥秘.chm》

6502 兼容 6527，

任天堂的 FC 和步步高的文曲星用的是同一种 CPU:6502 或 6527

//6502 寄存器

1.累加寄存器 A

这是个 8 位寄存器,既然是 8 位,那么说明该寄存器中只能存储一个(00-FF)之间的立即数.它与算术逻辑运算单元一起完成各种算术逻辑运算,它既可存放操作前的初始数据,也可存放操作结果,所以称为累加器.

在 6502 汇编中,这个寄存器应该算是用的最多的

大家也不要管那么多,只要知道有这个寄存器,该寄存器可以存放一个 00-FF 之间的立即数就可以了.

2.变址寄存器 X

也是 8 位寄存器,它在编程中常被当作一个计数器来使用.它可以由指令控制而被置成一个常数并能方便的用加 1,减 1,比较操作来修改和测试其内容,以使得程序能够方便灵活的处理数据块,表格等问题.

3.变址寄存器 Y

用法和 变址寄存器 X 一样,只不过在有些情况下,比如程序中要同时处理两个以上的数据块时,一个变址

寄存器显得不够,所以 6502 中有两个用于变址的寄存器 X 和 Y.

4.程序计数器 PC

它是 6502 中唯一的 16 位寄存器,PC 是用来存放指令地址码的寄存器,由于程序的执行一般为顺序执行方式,每

取出一个指令字节后 PC 即自动加 1,为取下一个指令字节做好准备,所以程序计数器 PC 中的内容往往是指向下一个

指令字节地址,但在执行转移指令时,PC 中将被放进要转移的目标地址.

5.堆栈指针 S

它是用来指示堆栈栈顶位置的寄存器,由于 6502 规定堆栈设在 第 1 页存储器中,所以堆栈指针 S 也是 8 位寄存器

只用来指出堆栈位置的低 8 位 地址.S 具有数据进栈时自动减 1,出栈时自动加 1 的功能.

6.标志寄存器 P

这也是 8 位的寄存器,但是只用了其中的 7 位,第 5 位空着不用.

每条指令在执行之后往往会发生进位溢出,结果为 0,或是结果为 负数(大于 7F 的数叫负数)的情况.指令执行完后常常

要保留这些情况作为条件分支的依据,标志寄存器 P 就是为了适应这需要而设计的,在寄存器 P 中有以下 7 个标志位,不过

我这里只介绍其中的 5 位

7	6	5	4	3	2	1	0
N	V		B	D	I	Z	C

C--进位标志.指令执行完毕后的最高进位状态,若最高位有进位则使 C = 1,若最高位无进位则使 C = 0

N--零标志. 指令执行完毕后结果为 0,那么 Z = 1;否则 Z = 0.

I--中断标志.此位置 0 表示允许中断,置 1 表示禁止中断,但非屏蔽中断不受次约束

V--溢出标志.指令执行后若产生溢出,则次标志位被置 1

N--负数标志.指令执行完毕后,若结果最高位 为 1,则该位置 1

下面我们举例来说明:

例: 两个正数 61,4A 相加

0110 0001

\+ 0100 1010

= 1010 1011

两个正数相加,为什么结果变为负数呢?这是也 61 + 4A = AB,超过了八位寄存器所能表示的最大正数 7F,而产生了溢出

那么这时 V = 1,结果不是 0,那么 Z = 0,结果最高位为 1,那么 N = 1,结果最高位没有进位,那么 C = 0

标志位常常在执行条件转移指令时做为条件判断的依据.这在后面的指令系统中会讲到.

//

6502寻址

FC 技术书籍 2008-10-06 00:03 阅读 54 评论 0

本文来自文曲星的《6502 编程大奥秘.chm》

6502 兼容 6527，

任天堂的 FC 和步步高的文曲星用的是同一种 CPU:6502 或 6527

//6502 寻址

1.立即寻址

2 字节指令.

指令的操作数部分给出的不是操作数地址而是操作数本身,我们称为立即数(00-FF 之间的任意数)

寻址方式的指令格式:

操作码	第一字节
操作数	第二字节

例如指令 LDA #$30,这里"#"表示后面的是立即数，"$"表示是十六进制表示

这条指令就是 立即寻址,这条指令的功能是将立即数 30 送寄存器 A.

例如指令 ADC #$30 //寄存器 A 的内容与立即数 30 和进位 C 相加,这里操作数 30 直接给出,所以是立即寻址

```
SUB #$30

LDX #$30    //把立即数 30 送寄存器 X

LDY #$30    //把立即数 30 送寄存器 Y

AND #$30    //寄存器 A 的内容和立即数 30 进行逻辑与运算
```

说明:

1.立即寻址一般用来设置初始数据

2.立即寻址的指令,执行速度很快

2.直接寻址

三字节指令.

指令的操作数给出的是操作数在存储器中的有效地址,所以称为直接寻址.指令格式:

操作码	第一字节
操作数地址低字节	第二字节
操作数地址高字节	第三字节

由于操作数地址是两个字节,所以它可以是整个内存中的任何一个地址,这种指令表示成机器码时操作数地址是低字节在前,高字节在后.例如指令 LDA $3000,表示成机器码为:AD 00 30,而不是AD 30 00,初学者比较容易混淆这一点.

例如指令 LDA $3000,该指令的功能是将 地址 3000 中的内容送寄存器 A

例如指令 STA $3001,该指令的功能是将寄存器 A 的内容送地址 3001

我们可在 NCTOOLS 做这样一个实验,证明一下

输入 A 2000,然后输入以下代码:

2000:LDA $3000

2003:STA $3001

2006:RTS

然后 E C 3000,输入一个数据,比如 40,然后 G 2000,然后 D 3001,看看地址 3001 的内容是不是 40 呢?

3.零页寻址

2 字节指令.

先说说什么是零页,地址 00-地址 FF 就叫做零页地址

零页寻址和直接寻址的区别在于零页寻址方式中操作数的地址仅限于存储器的零页范围(00-FF)

指令格式如下:

操作码　　第一字节
操作数　　第二字节

例如指令 LDA $F0,功能是将地址 F0 的内容送寄存器 A,这里 F0 属于零页范围

有一点需要说清楚,可以用零页寻址的指令,一般就可用直接寻址.

例如指令 LDA $F0 和指令 LDA $00F0,功能是完全一样的,不过我们不应该用直接寻址,为什么呢?因为直接寻址占 3 个

字节,而零页寻址仅仅 2 个字节,而且零页寻址执行速度快些,所以可以用零页寻址的指令就不应该用直接寻址.

4.累加器寻址

单字节指令.

指令操作所需要的操作数就存在于寄存器 A 中,所以无须操作数,该指令仅仅占一个字节.

例如指令 LSR,默认就是将寄存器 A 的内容逻辑右移一位,而无须在操作数中指出操作数,默认操作数就是寄存器 A 的内容.

指令 PHA, 默认就是将寄存器 A 的内容压入堆栈.所以也是累加器寻址.

5.隐含寻址

单字节指令

隐含寻址和累加器寻址的区别在于隐含寻址的操作数地址是除寄存器 A 外的其他寄存器,例如寄存器 X,Y,S 或 P

所以说,累加器寻址其实也可以叫隐含寻址.

例如指令 INX ;寄存器 X 的内容加 1,这里隐含规定操作数就是寄存器 X 的内容,所以叫隐含寻址

INY ;寄存器 Y 的内容加 1,这里隐含规定操作数就是寄存器 Y 的内容,所以叫隐含寻址

DEX ;.....

DEY ;.....

6.使用寄存器 X 的直接变址

为了方便起见,我们称该寻址方式为 直接 X 变址

三字节指令

这种寻址方式是将一个 16 位的直接地址作为基地址,然后和寄存器 X 的内容相加,结果就是真正的有效地址,指令格式:

操作码	第一字节
基地址低字节	第二字节
基地址高字节	第三字节

例如指令 LDA $3000,X 它的寻址过程是这样的:

假使此时寄存器 X 的内容为 03,即(X) = 03,地址 3003 的内容为 40,即(3003) = 40

先确定基地址 3000

把基地址 3000 + (X) = 3000 + 03 = 3003,计算出有效地址为 3003

然后把地址 3003 的内容送寄存器 A

这里我们可以发现,有效地址是随寄存器 X 的内容发生变化的,所以叫直接 X 变址.

7.使用寄存器 Y 的直接变址

该寻址方式和上面是一样的,只不过把寄存器 X 换成寄存器 Y 而已.

为了方便起见,我们称该寻址方式为 直接 Y 变址

三字节指令

这种寻址方式是将一个 16 位的直接地址作为基地址,然后和寄存器 Y 的内容相加,结果就是真正的有效地址,指令格式:

操作码	第一字节
基地址低字节	第二字节
基地址高字节	第三字节

例如指令 LDA $3000,Y 它的寻址过程是这样的:

假使此时寄存器 Y 的内容为 03,即(Y)=03,地址 3003 的内容为 40,即(3003)=40

先确定基地址 3000

把基地址 3000 + (Y) = 3000 + 03 = 3003,计算出有效地址为 3003

然后把地址 3003 的内容送寄存器 A

这里我们可以发现,有效地址是随寄存器 Y 的内容发生变化的,所以叫直接 Y 变址.

8.使用寄存器 X 的零页变址

现在我们应该不看我的下面的说明就能知道这种寻址方式的用法了吧,其实和上面的寻址方式几乎是一样的

只不过这里的基地址仅仅限于零页地址罢了,指令格式如下:

操作码	第一字节
零页基地址	第二字节

例如指令 LDA $F0,X 寻址过程如下:

设(X) = 03,(F3) = 40

基地址 F0 + (X) = F0 + 03 = F3

这条指令的功能就是将地址 F3 的内容送寄存器 A

9.使用寄存器 Y 的零页变址

和上面的"使用寄存器 X 的零页寻址"不同之处仅仅在于把寄存器 X 换为寄存器 Y.

操作码	第一字节
零页基地址	第二字节

例如指令 LDX $F0,Y 寻址过程如下:

设(Y)=03,(F3)=40

基地址 F0+(Y)=F0+03=F3

这条指令的功能就是将地址 F3 的内容送寄存器 X

10.间接寻址

在 6502 中,仅仅用于无条件跳转指令 JMP 这条指令

三字节指令.

该寻址方式中,操作数给出的是间接地址,间接地址是指存放操作数有效地址的地址,指令格式:

操作码	第一字节
间接地址低字节	第二字节
间接地址高字节	第三字节

由于操作数有效地址是 16 位的,而每一存储单元内容仅仅 8 位,所以要通过两次间接寻址才能得到有效地址

我们还是举例子说明吧

这里我们设 (3000) = 23,(3001) = 30

指令 JMP ($3000)的寻址过程是这样的:

先对地址 3000 间接寻址得到有效地址低 8 位 23

再对地址 3001 间接寻址得到有效地址高 8 位 30

这样,再把两次结果合在一起就得到有效地址=3023

执行该指令后,程序就无条件跳转到地址 3023

11.先变址 X 后间接寻址

两字节指令

指令格式:

操作码	第一字节
零页基地址	第二字节

这种寻址方式是先以 X 作为变址寄存器和零页基地址 IND 相加 IND+X,不过这个变址计算得到的只是一个间接地址,还必须

经过两次间接寻址才得到有效地址

第一次对 IND + X 间址得到有效地址低 8 位

第二次对 IND + X + 1 间址得到有效地址高 8 位

然后把两次的结果合起来,就得到有效地址.

我们看一个例子:

指令 LDA ($F0,X) 的寻址过程如下:

这里设 (X) = 02,(F2) = 30,(F3) = 40

那么先得到间接地址 = F0 + (X) = F0 + 02 = F2

第一次对地址 F0 + (X) = F2 间址得到有效地址低 8 位 = 30

第二次对地址 F0 + (X) + 1 = F3 间址得到有效地址高 8 位 = 40

那么有效地址就是地址 4030 了,该指令功能就是将地址 4030 的内容送寄存器 A,大家可以在 NCTOOLS 中试一下

12.后变址 Y 间接寻址

两字节指令

指令格式:

操作码	第一字节
零页间接地址	第二字节

这种寻址方式是对 IND 部分所指出的零页地址先做一次间接寻址,得到一个低 8 位地址

再对 IND + 1 作一次间接寻址,得到一个高 8 位地址

最后把这高,低两部分地址合起来作为 16 的基地址,和寄存器 Y 进行变址计算得到操作数的有效地址,注意的是这里 IND 是零页地址

看一个例子:

例如指令 LDA ($F0),Y

我们看看寻址过程:

设 (F0)=20,(F1)=30,(Y)=03

先对地址 F0 间址得到低 8 位地址 20

再对地址 F0+1 间址得到高 8 位地址 30

把两次结果合起来得到 16 位的基地址 3020

然后再把地址 3020 和寄存器 Y 进行变址,得到有效地址 3020+(Y)=3020+03=3023

所以该指令的功能是将地址 3023 的内容送寄存器 A

13.相对寻址

该寻址仅用于条件转移指令,指令长度为 2 个字节.第 1 字节为操作码,第 2 字节为条件转移指令的跳转步长.又叫偏移量 D.偏移量可正可负,D 若为负用补码表示.

指令格式:

操作码 第 1 字节

偏移量 D 第 2 字节

相对寻址是用本条指令的第 1 个字节所在地址和偏移量 D 相加得到有效地址.

由于在实际中,你是用汇编写程序,所以没有必要搞懂其寻址方式,如果你想用机器码写程序那么你必须搞懂,下面的东西你就必须看.

①负偏移的计算

例如下面的程序

```
2000:A2  9C        LDA  #$9C
2002:BD  00  00    LDA  $000,X
2005:9D  BF  02    STA  $02BF,X
2008:CA            DEX
2009:D0  ??        BNE  $2002
200B:60            RTS
```

这里在地址 2009 那里的??就是偏移量,这里我们要跳到地址 2002,那么怎么计算出偏移量呢?

方法是: A = (2009 + 1) - 2002 + 1 = 2009 - 2002 + 2 =9

然后 B = 256 - 9 = 247

然后化为 16 进制形式 B = F7

所以这里 ?? = F7

②正偏移的计算

例如下面的程序

```
2000: A2 00     LDX #$00
2002: BD 00 00  LDA $0000,X
2005: 9D C0 02  STA $02C0,X
2008: E0 9B     CPX #$9B
200A: F0 ??     BEQ $2010
200C: E8        INX
200D: 4C 02 20  JMP $2002
2010: 60        RTS
```

这里??的计算方法是

A = 2010 - 200A -2 = 4

//

6502汇编指令 (之一)

FC 技术书籍 2008-10-06 00:06 阅读 214 评论 0

本文来自文曲星的《6502 编程大奥秘.chm》
任天堂的 FC 和步步高的文曲星用的是同一种 CPU:6502 或 6527

6502 汇编指令。6502 及 6527（后者为台湾盗版，内核及指令系统完全一样，仅引脚排列顺序不同）两个芯片曾用于中华学习机、小霸王、电子词典等上，交流学习，互通有。

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

一、数据传送指令

//LDA--由存储器取数送入累加器 M→A

符号码格式	指令操作码	寻址方式
LDA ($addr,X)	A1	先变址 X 后间址
LDA $addr	A5	零页寻址
LDA #$data	A9	立即寻址
LDA $addr16	AD	绝对寻址
LDA ($addr),Y	B1	后变址 Y 间址
LDA $addr,X	B5	零页 X 变址
LDA $addr16,Y	B9	绝对 Y 变址
LDA $addr16,X	BD	绝对 X 变址

//LDX--由存储器取数送入累加器 M→X

符号码格式	指令操作码	寻址方式
LDX #$data	A2	立即寻址
LDX $addr	A6	零页寻址
LDX $addr16	AE	绝对寻址
LDX $addr,Y	B6	零页 Y 变址
LDX $addr16,Y	BE	绝对 Y 变址

//LDY--由存储器取数送入累加器 M→Y

符号码格式	指令操作码	寻址方式
LDY #$data	A0	立即寻址
LDY $addr	A4	零页寻址
LDY $addr16	AC	绝对寻址
LDY $addr,X	B4	零页 X 变址

LDY $addr16,X	BC	绝对 X 变址

//STA--将累加器的内容送入存储器 A--M

符号码格式	指令操作码	寻址方式
STA ($addr,X)	81	先变址 X 后间址
STA $addr	85	零页寻址
STA $addr16	8D	绝对寻址
STA ($addr),Y	91	后变址 Y 间址
STA $addr,X	95	零页 X 变址
STA $addr16,Y	99	绝对 Y 变址
STA $addr16,X	9D	绝对 X 变址

//STX--将寄存器 X 的内容送入存储器 X--M

符号码格式	指令操作码	寻址方式
STX $addr	86	零页寻址
STX $addr16	8E	绝对寻址
STX $addr,Y	96	零页 Y 变址

//STY--将寄存器 Y 的内容送入存储器 Y--M

符号码格式	指令操作码	寻址方式
STY $addr	84	零页寻址
STY $addr16	8C	绝对寻址
STY $addr,X	94	零页 X 变址

//寄存器和寄存器之间的传送

符号码格式	指令操作码	寻址方式	指令作用
TAX	AA	寄存器寻址	将累加器 A 的内容送入变址寄存器 X
TXA	8A	寄存器寻址	将变址寄存器 X 的内容送入累加器 A
TAY	A8	寄存器寻址	将累加器 A 的内容送入变址寄存器 Y
TYA	98	寄存器寻址	将变址寄存器 Y 的内容送入累加器 A
TSX	BA	寄存器寻址	将堆栈指针 S 的内容送入变址寄存器 X
TXS	9A	寄存器寻址	将变址寄存器 X 的内容送入堆栈指针 S

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

二、[算术运算指令]

1. ADC--累加器,存储器,进位标志 C 相加,结果送累加器 A A+M+C→A

符号格式	指令码操作码	寻址方式	周期
ADC ($addr,X)	61	先变址 X 后间址	
ADC $addr	65	零页寻址	
ADC #$data	69	立即寻址	
ADC $addr16	6D	绝对寻址	
ADC ($addr),Y	71	后变址 Y 间址	
ADC $addr,X	75	零页 X 变址	
ADC $addr16,Y	79	绝对 Y 变址	
ADC $addr16,X	7D	绝对 X 变址	

注意:由于进位标志 C 页会参加运算,所以在做加法运算时,一般要在前面加指令 CLC,清除进位标志

例 1: //两个 8 位数加法运算演示,目的是将寄存器 A 的内容加上地址 2100 的内容,结果送寄存器 A

```
2000:LDA  #$20      //立即数 21 送寄存器 A
2002:STA  $2100     //地址 2100 的内容为 20
2005:LDA  #$21      //寄存器 A 的内容为 21
2007:CLC            //清除进位标志 C,注意:在做加法运算前一定要加该指令
2008:ADC  $2100     //寄存器 A 的内容和地址 2100 的内容相加
200B:RTS
```

大家可以进入 NCTOOLS，输入 A 2000，然后输入上面代码，然后 输入 G 2000，然后按 R 看寄存器 A 的值，是不是等于 41 呢？

例 2：//两个 16 位数加法运算，这里是将 0194+01BA，大家注意看看程序与上面有什么不同.

在这个程序里,先将要相加的数分别放地址 2100,2101,2102,2103,具体是这样的:2100:94 01 BA 01

然后先将地址 2100 的内容+地址 2102 的内容,结果送地址 2104

再将地址 2101 的内+地址 2103 的内容,结果送地址 2105

那么地址 2104,2105 的结果,就是两个 16 位数相加的结果,这里是 034E

```
2000:LDA #$94        //立即数 94 送寄存器 A
2002:STA $2100       //寄存器 A 的内容送地址 2100
2005:LDA #$01        //立即数 01 送寄存器 A
2007:STA $2101       //寄存器 A 的内容送地址 2101
200A:LDA #$BA        //立即数 BA 送寄存器 A
200C:STA $2102       //寄存器 A 的内容送地址 2102
200F:LDA #$01        //立即数 01 送寄存器 A
2011:STA $2103       //寄存器 A 的内容送地址 2103
2014:CLC             //清除进位标志,注意,开始做加法程序前,一定要清除进位标志
2015:LDA $2100       //地址 2100 的值送寄存器 A
2018:ADC $2102       //寄存器 A 的内容 + 地址 2102 的内容 + C → A,这里 C=0
201B:STA $2104       //寄存器 A 的内容送地址 2104
201E:LDA $2101       //地址 2101 的内容送寄存器 A
2021:ADC $2103       //寄存器 A 的内容 + 地址 2103 的内容 + C → A,这里 C=1,请注意,这里没有用 CLC 指令
2024:STA $2105       //寄存器 A 的内容送地址 2105
2027:RTS             //程序结束
```

在这个程序里,最重要的就是为什么后面的加法指令前面没有用 CLC 指令,不是说在做加法前要用 CLC 指令吗?

但这里就不一样,由于在前面已经用了 CLC 指令,将 C=0,做了一次加法运算后,会影响标志位 C

如果做完第一次加法运算后,C=0,那么说明没有产生进位,C 还是等于 0,所以没有必要再用 CLC 指令

如果做完第一次加法运算后,C=1,那么说明产生了进位,如果再做第二次加法运算前,还使用 CLC 指令,那么计算的结果

就是错误的,我们来看 第一次运算后,(2104)=4E,如果第二次运算前使用 CLC 指令,那么 01+01+00=02,最后的结果是这样的

(2104)=4E,(2105)=02,那么说明 0194+01BA=024E,我们口算都能知道这个结果是错误的,实际上由于第一次运算后,产生了进位

所以我们要把进位保留即 01+01+(C)=01+01+01=03,所以实际的结果是(2104)=4E,(2105)=03

2. SBC--从累加器减去存储器和进位标志 C,结果送累加器 A-M-C→A

符号码格式	指令操作码	寻址方式
SBC ($addr,X)	E1	先变址 X 后间址
SBC $addr	E5	零页寻址
SBC #$data	E9	立即寻址
SBC $addr16	ED	绝对寻址
SBC ($addr),Y	F1	后变址 Y 间址
SBC $addr,X	F5	零页 X 变址
SBC $addr16,Y	F9	绝对 Y 变址
SBC $addr16,X	FD	绝对 X 变址

注意:由于在做减法运算时,进位标志 C 会参与运算，所以在做减法前要先加指令 SEC，置进位标志

例： //减法指令演示,目的是将寄存器 A 的内容减去地址 2100 的内容,结果送寄存器 A

```
2000: LDA #$21          //立即数 21 送寄存器 A
2002:STA $2100          //寄存器 A 的内容送地址 2100
2005: LDA #$22          //立即数 22 送寄存器 A
2007:SEC                //置位标志位 C,注意:在做减法运算前一定要加该指令
2008:SBC $2100          //寄存器 A 的内容减去地址 2100 的内容
```

200B: RTS

大家可以进入 NCTOOLS，输入 A 2000，然后输入上面代码，然后 输入 G 2000，然后按 R 看寄存器 A 的值，是不是等于 01 呢？

例 2: //求二进制数的平方根

程序目的:将地址 2100 的内容求得其平方根后,结果送地址 2101,为方便计算,这里假设地址 2100 的内容为整数,方根也取整数

例如 若(2100)=19 (十进制的 25)

结果(2101)=05

若(2101)=65 (十进制的 101)

结果(2101)=0A

求方根方法:我们知道任何整数都有如下性质:

1 * 1 = 1

2 * 2 = 1 + 3

3 * 3 = 1 + 3 + 5

4 * 4 = 1 + 3 + 5 + 7

.....

N * N=1 + 3 + 5 + ... + (2N-1)

那么,我们可以把一个正整数 X = N * N 连续减去奇数 1, 3, 5, 7.....(2N-1)直到结果为 0 或者不够减为止,所减去奇数的个数

就是这个正整数的整数平方根.

在下面的程序里,我们将目标数连续减去地址 F0 的内容

2000:LDA#$00 //初始化地址 F0 的内容为 00

2002:STA $F0

2004:LDA $2100 //取目标数到寄存器 A 中

2007:SEC

2008:SBC $F0 //目标数减去地址 F0 的内容

200A:BCC $2016 //不够减转结束,结果在地址 F0 中

200C:INC $F0 //地址 F0 的内容加 1,(06)为已减的奇数个数

200E:SBC $F0 //和前面的减法指令合起来正好减去了奇整数

2010:BEQ $2016 //减结果为 0,转结束,结果在地址 F0 中

2012:BCS $2008 //减结果>0,继续减

2014:DEC $F0 //减结果<0,则从结果中扣除 1

2016:LDA $F0

2018:STA $2101 //把最后结果送地址 2101

201B:RTS

3. INC--存储器单元内容增 1 M+1→M

符号码格式	指令操作码	寻址方式
INC $addr	E6	零页寻址
INC $addr16	EE	绝对寻址
INC $addr,X	F6	零页 X 变址
INC $addr16,X	FE	绝对 X 变址

这个指令应该很好理解吧,就是把某个地址的内容加 1,当然我们也可以用加法指令把某个地址加 1,但大家可以看到,这里的指令所占用字节少

执行速度也较快.请看下面的比较:

用加法指令把地址 2100 的内容加 1

2000:LDA $2100

2003:CLC

2004:ADC #$01

2006:STA $2100

2009:RTS

所占字节数:10 个

用上面的指令把地址 2100 的内容加 1

2000:INC $2100

2003:RTS

所占字节数:4 个

一比较,大家就能发现谁占了上风吧!在对某个地址进行加 1 运算时,应该用 INC 指令.

4. DEC--存储器单元内容减 1 M-1→M

符号码格式	指令操作码	寻址方式
DEC $addr	C6	零页寻址
DEC $addr16	CE	绝对寻址
DEC $addr,X	D6	零页 X 变址
DEC $addr16,X	DE	绝对 X 变址

5. 寄存器 X,Y 加 1 减 1

符号码格式	指令操作码	寻址方式
INX	E8	隐含寻址
DEX	CA	隐含寻址
INY	C8	隐含寻址
DEY	88	隐含寻址

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

三、[逻辑运算指令]

1.AND--寄存器与累加器相与,结果送累加器 A∧M→A

符号码格式	指令操作码	寻址方式
AND ($addr,X)	21	先变址 X 后间址
AND $addr	25	零页寻址
AND #$data	29	立即寻址
AND $addr16	2D	绝对寻址
AND ($addr),Y	31	后变址 Y 间址
AND $addr,X	35	零页 X 变址
AND $addr16,Y	39	绝对 Y 变址
AND $addr16,X	3D	绝对 X 变址

逻辑与的主要功能是对目的操作数的某些位置 0,或测试某位的状态

例 1: 屏蔽地址 2100 的高四位,即将地址 2100 的高四位清零

2000:LDA $2100

2003:AND #$7F (7F 转化为二进制:0000 1111)

2005:STA $2100

2008:RTS

所以我们可以知道,我们要使地址 2100 的高四为为 0,只要使操作数的高四位为 0,低四位为 1,即操作数为 0000 1111

然后再把地址 2100 的内容送寄存器 A,把 A 的内容和 0000 1111 进行逻辑与运算,那么寄存器 A 的高四位自然就为 0,由于操作数

低四位为 1,所以寄存器 A 的低四位不变.

例 2: 测试地址 2100 的第 2 位的状态,如果第 2 位=0,那么把 00 送寄存器 X,否则把 01 送寄存器 X

2000:LDA $2100

2003:AND #$04 (04 转化为二进制:0000 0100)

2005:CMP #$04

2007:BNE $200C

2009:LDX #$01

200B:RTS

200C:LDX #$00

200E:RTS

在这个程序中,我们要测试第 2 位的状态,只要让它与 0000 0100 进行逻辑与运算,若第 2 位为 0,那么结果一定为 0

如果第二位为 1,那么由于操作数的第 2 位也为 1,所以结果为 0000 0100,即十六进制的 04,所以我们可以判断结果是不是 04

如果是 04,那么说明第 2 位为 1,否则为 0.

随便说个笑话,我们数字电路的老师告诉我们这样的口诀记住逻辑与的用法,那就是 "见 0 出 0,全 1 出 1"

2.ORA--寄存器与累加器相或,结果送累加器 A∨M→A

符号码格式	指令操作码	寻址方式
ORA ($addr,X)	01	先变址 X 后间址
ORA $addr	05	零页寻址
ORA #$data	09	立即寻址
ORA $addr16	0D	绝对寻址
ORA ($addr),Y	11	后变址 Y 间址
ORA $addr,X	15	零页 X 变址
ORA $addr16,Y	19	绝对 Y 变址
ORA $addr16,X	1D	绝对 X 变址

先告诉大家哪个不登大雅之堂的口诀 "见 1 出 1,全 0 出 0"

逻辑或的功能主要是对目的操作数的某些位置 1

例 1: 使地址 2100 的高 4 位置 1

2000:LDA $2100

2003:ORA #$F0 (F0 的二进制:1111 0000)

2005:STA $2100

2008:RTS

这里我就不解释为什么这样做了,大家可以想一下.

3.EOR--寄存器与累加器相异或,结果送累加器 A⊻M→A

符号码格式	指令操作码	寻址方式
EOR ($addr,X)	41	先变址 X 后间址
EOR $addr	45	零页寻址
EOR #$data	49	立即寻址
EOR $addr16	4D	绝对寻址
EOR ($addr),Y	51	后变址 Y 间址
EOR $addr,X	55	零页 X 变址
EOR $addr16,Y	59	绝对 Y 变址
EOR $addr16,X	5D	绝对 X 变址

先告诉大家哪个不登大雅之堂的口诀 "相同出 0,不同出 1"

异或的功能主要就是求补码,加密等.

例 1: 求地址 2100 的内容的反码

2000:LDA $2100

2003:EOR #$FF

2005:STA $2100

2008:RTS

先说明下异或是怎么回事,什么是 "相同出 0,不同出 1"

这里我们要把立即数 7F,40 进行异或运算,过程是这样的

1.先把 7F,40 转化为二进制形式

2.然后如果相同的位的值都不相同,那么该位为 1,否则为 0

(HEX) 7F (BIN) 0 1 1 1 1 1 1 1 (EOR)

(HEX) 40 (BIN) 0 1 0 0 0 0 0 0

(HEX) 3F (BIN) 0 0 1 1 1 1 1 1

所以我们要求反码,只要将该数和 FF 进行异或运算就可以了.

例 2: 把地址 3000-30FF 的数据加密与解密

先说明下为什么异或运算可以对数据进行加密,异或运算有一个特性:

一个操作数(这里设为 D1),和另外一个操作数(这里设为 D2)进行异或运算,结果为 D3

表达式是这样的 D1 EOR D2 = D3

然后如果我们将 D3 再和 D2 进行异或运算,结果一定为 D1,这就是加密的依据.

所以我们要对某段数据进行加密时,只要使改段数据均和某个数进行异或运算,解密时再把加密的数据再和该数进行

异或运算即可,这里的某个数我们成为密匙,也就是说,一个人要解密,他必须得到密匙才能解密.

当然实际运用时,我们可以搞的稍微复杂些,就比如下面的例子,以寄存器 X 的内容作为对象进行异或.

加密程序如下:

```
2000:LDX  #$50          //这里 50 就是密匙,不过这里的密匙不是不变的,每异或一次就加 1
2002:LDY  #$00          //计数器 Y 初始化为 00
2004:INX                //寄存器 X 的内容加 1,即每异或一次,寄存器 X 的内容就加 1
2005:TXA                //寄存器 X 的内容发送给寄存器 A
2007:EOR  $3000,Y       //寄存器 A 的内容和地址[3000+Y]的内容进行异或运算
200A:STA  $3000,Y       //结果仍然送回原地址
200D:INY                //计数器 Y 的值加 1
200E:BNE  $2004         //当全部加密完,程序结束,否则继续
```

2010:RTS

解密程序如下:

2100:LDX #$50 //解密时,必须知道加密时 X 寄存器的初值,否则无法解密

2102:LDY #$00 //计数器 Y 初始化为 00

2104:INX //寄存器 X 的内容加 1,即每异或一次,寄存器 X 的内容就加 1

2105:LDA $3000,Y //地址[3000+Y]的内容送寄存器 A

2108:STX $F0 //X 寄存器的内容送地址 F0,为下面的指令做好准备,因为没有和寄存器 X 进行异或运算的指令

210A:EOR $F0 //寄存器 A 的内容和地址 F0 即寄存器 X 的内容进行异或

210C:STA $3000,Y //结果送回原地址

210F:INY //计数器 Y 的值加 1

2110:BNE $2104 //当全部解密完,程序结束,否则继续

2112:RTS

由上面的程序可以知道,加密程序的密匙是可变的,但是变化规律很明显,就是一直加 1,解密程序和加密程序几乎是一样的.

当然,这样的加密程序是不管用的,讲这个程序的目的只是抛砖引玉,为了说明 EOR 在加密中的运用.

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

四、[置标志位指令]

1. CLC--清除进位标志 0→C 机器码 18 √

2. SEC--置进位标志 C 1→C 机器码 38 √

3. CLD--清除十进制运算标志 D 0→D 机器码 D8 ×

4. SED--置十进制运算标志 D　1→D　机器码 F8　×

5. CLV--清除溢出标志 V　0→V　机器码 B8

6. CLI--清除中断禁止指令 I　0→I　机器码 58　√

7. SEI--置位中断禁止标志 I　1→I　机器码 78　√

说明:上面的指令中,用的比较多的是指令 CLC,SEC,CLI,SEI,这些指令也没有什么好讲的

只是有两点要注意:

1.如果你在一个程序中用了 SEI 指令,那么程序结束前一定要用 CLI 指令,否则会死机.

2.指令 SED 在文曲星中似乎不能用,我每次用都会死机.也不知道是怎么回事,大家暂时也别用,CLD 倒是可以用,不过好象没有用

的必要,所以这两条指令我看可以去掉.

6502汇编指令 (之二)

FC 技术书籍 2008-10-06 00:09 阅读 214 评论 0

本文来自文曲星的《6502 编程大奥秘.chm》
任天堂的 FC 和步步高的文曲星用的是同一种 CPU:6502 或 6527

6502 汇编指令。6502 及 6527（后者为台湾盗版，内核及指令系统完全一样，仅引脚排列顺序不同）两个芯片曾用于中华学习机、小霸王、电子词典等上，交流学习，互通有。

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

五、比较指令

1. CMP--累加器和存储器比较

符号码格式	指令操作码	寻址方式
CMP ($addr,X)	C1	先变址 X 后间址
CMP $addr	C5	零页寻址
CMP #$data	C9	立即寻址
CMP $addr16	CD	绝对寻址
CMP ($addr),Y	D1	后变址 Y 间址
CMP $addr,X	D5	零页 X 变址
CMP $addr16,Y	D9	绝对 Y 变址
CMP $addr16,X	DD	绝对 X 变址

该指令也是做减法操作,将寄存器的内容减去存储器的内容,但它和减法指令有 2 点区别:

一是借位标志 C 不参加运算,所以在用 CMP 指令不必加指令 SEC

二是减法的结果不送入寄存器 A

该指令运行后,会影响标志位 C，Z，N.我们在实际中尤其要注意它是如何影响标志位 C 和标志位 Z

若执行指令 CMP 后,C=1 表示无借位,即 A》M

若执行指令 CMP 后,C=0 表示有借位,即 A<M

若执行指令 CMP 后,Z=1 表示 A=M

从上面我们可以判断出 A 和 M 谁大谁小,或者 A 和 M 是不是相等

例：//比较指令演示,演示如何判断 A 和 M 的大小

2000:LDA $2100 //地址 2100 的内容送寄存器 A

2003:CMP $2101 //寄存器 A 的内容和地址 2101 的内容相比较

2006:BEQ $2016 //若标志位 Z=1,那么程序就跳转到地址 2016

2008:BCC $2010 //若标志位 C=0,那么程序就跳转到地址 2010

200A:LDA #$02 //若进位标志 C=1,程序不跳转,顺序执行

200C:STA $2102 //若程序执行到这里，说明 C=1,那么将立即数 02 送地址 2102，作为地址 2100 的值>地址 2101 的值的标志

200F:RTS

2010:LDA #$01 //若标志位 C=0，那么将立即数 01 送地址 2102，作为地址 2100 的值<地址 2101 的值的标志

2012:STA $2102

2015:RTS

2016:LDA #$00 //若标志位 Z=1，那么将立即数 00 送地址 2102，作为地址 2100 的值=地址 2101 的值的标志

2018:STA $2102

201B:RTS

进入 NCTOOLS,输入 A 2000,然后输入上面的程序,先用 E 命令 E 2100,输入 0102,然后 G 2000,然后看地址 2102 的值

这里因为(2100)=01,(2101)=02,显然 01>02 那么程序执行后,(2102)=02,你可以用 D 2102,看看是不是这样.

你也可以输入别的数值,看看地址 2102 的内容是不是和预期的一样.

2. CPX--寄存器 X 的内容和存储器比较

符号码格式	指令操作码	寻址方式
CPX #$data	E0	立即寻址
CPX $addr	E4	零页寻址
CPX $addr16	EC	绝对寻址

这些指令和 CMP 指令相似,不过前者是寄存器 A,后者是寄存器 X,另外寻址方式也比较少.

这条指令用的比较多,特别是在循环时

例： //CPX 在循环程序中的运用,该程序实现了将地址 3000-30FF 的内容发送到地址 3100-31FF

2000:LDX #$00 //初始化寄存器 X 的值,一开始（X）=0

2002:LDA $3000,X //地址[3000+X]的内容送寄存器 A

2005:STA $3100,X //寄存器 A 的内容送地址[3100+X]

2008:INX //寄存器 X 的内容加 1

2009:CPX #$00 *//如果寄存器 X 的内容=00,那么说明数据已经发送完了.注意：FF+01=00*

200B:BNE 2002 //程序跳转到地址 2002 继续发送直到寄存器 X 的内容=00

200D:RTS

这里我们来为初学者分析一下代码运行过程：

第 1 次循环 （X）=00 地址[3000+00]=3000 的内容送地址[3100+00]=3100

第 2 次循环 (X) =01 地址[3000+01]=3001 的内容送地址[3100+01]=3101

第 3 次循环 (X) =02 地址[3000+02]=3002 的内容送地址[3100+02]=3102

....

....

第 256 次循环 (X) =FF 地址[3000+FF]=30FF 的内容送地址[3100+FF]=31FF

3. CPY--寄存器 Y 的内容和存储器比较

符号码格式	指令操作码	寻址方式
CPY #$data	C0	立即寻址
CPY $addr	C4	零页寻址
CPY $addr16	CC	绝对寻址

这些指令和 CPX 指令相似,不过前者是寄存器 X,后者是寄存器 Y.

4. BIT--位测试指令

符号码格式	指令操作码	寻址方式
BIT $addr	24	零页寻址
BIT $addr16	2C	绝对寻址

这条指令的功能和 AND 指令有相同之处,那就是把累加器 A 同存储器单元相与,但和 AND 指令不同的是相与的结果不送入累加器 A

另外该指令对标志位的影响也和 AND 指令不同

若 结果=0，那么 Z=1

若 结果<>0,那么 Z=0

N=M 的第 7 位

V=M 的第 6 位

所以执行该指令后 N，V 两标志位的状态就是参加与操作的存储单元的最高两位状态

这些指令在通讯程序中用的相当多，大家要给予足够的重视，是很有用的指令

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

六、移位指令

1. 算术左移指令 ASL

符号码格式	指令操作码	寻址方式
ASL	0A	累加器寻址
ASL $data	06	零页寻址
ASL $addr16	0E	绝对寻址
ASL $addr,X	16	零页 X 变址
ASL $addr16,X	1E	绝对 X 变址

ASL 移位功能是将字节内各位依次向左移 1 位，最高位移进标志位 C 中，最底位补 0

ASL 执行结果相当于把移位前的数乘 2

例如 //ASL 的应用

```
2000:LDA  #$20          //把立即数 20 送累加器 A
2002:ASL                //累加器 A 的内容算术左移
2003:STA  $2100         //把累加器 A 的内容送地址 2100
2006:ASL  $2100         //地址 2100 的内容算术左移
2009:LDA  $2100         //地址 2100 的内容送累加器 A
200C:RTS                //程序结束
```

2. 逻辑右移指令 LSR

符号码格式	指令操作码	寻址方式
LSR	4A	累加器寻址
LSR $data	46	零页寻址
LSR $addr16	4E	绝对寻址
LSR $addr,X	56	零页 X 变址
LSR $addr16,X	5E	绝对 X 变址

该指令功能是将字节内各位依次向右移 1 位，最低位移进标志位 C，最高位补 0.

该操作对于无符号数和正数相当于乘 1/2

例： //拆字程序，将地址 2100 单元的高四位送地址 2101 的低四位，将地址 2100 单元的低四位送地址 2102 的底四位

并且清除地址 2101 和地址 2102 的高四位

```
2000:LDA  $2100         //地址 2100 的内容送 A
2003:AND  #$0F          //A 和 0F 进行逻辑与运算，屏蔽了 A 的高四位
2005:STA  $2102         //结果送地址 2102
2008:LDA  $2100
200B:LSR                //将 A 的高四位挪到低四位，高四位补 0
200C:LSR
```

200D:LSR

200E:LSR

200F:STA $2101 //结果送地址 2101

2012:RTS

3. 循环左移指令 ROL

符号码格式	指令操作码	寻址方式
ROL	2A	累加器寻址
ROL $data	26	零页寻址
ROL $addr16	2E	绝对寻址
ROL $addr,X	36	零页 X 变址
ROL $addr16,X	3E	绝对 X 变址

ROL 的移位功能是将字节内容连同进位 C 一起依次向左移 1 位

4. 循环右移指令 ROR

符号码格式	指令操作码	寻址方式
ROR	6A	累加器寻址
ROR $data	66	零页寻址
ROR $addr16	6E	绝对寻址
ROR $addr,X	76	零页 X 变址
ROR $addr16,X	7E	绝对 X 变址

ROR 的移位功能是将字节内容连同进位 C 一起依次向右移 1 位

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

七、堆栈操作指令

1. 累加器进栈指令 PHA

PHA 是隐含寻址方式的单字节指令，操作码是 48

功能是把累加器 A 的内容按堆栈指针 S 所指示的位置送入堆栈，然后堆栈指针减 1

该指令不影响标志寄存器 P 的状态

2. 累加器出栈指令 PLA

PLA 是隐含寻址方式的单字节指令，操作码是 68

功能是先让堆栈指针 S+1，然后取加过 1 的 S 所指向的单元的内容，把它送累加器 A

该指令影响标志寄存器 P 中的 N，Z 两标志位

3. 标志寄存器 P 进栈指令 PHP

PHP 是隐含寻址方式的单字节指令，操作码是 08

功能是把标志寄存器 P 的内容按堆栈指针 S 所指示的位置送入堆栈，然后堆栈指针减 1

该指令不影响标志寄存器 P 的状态

4. 标志寄存器 P 出栈指令 PLP

PLP 是隐含寻址方式的单字节指令，操作码是 28

功能是先让堆栈指针 S+1，然后取加过 1 的 S 所指向的单元的内容，把它送标志寄存器 P

5. 堆栈用法举例

堆栈是一个存储区域,用来存放调用子程序或响应中断时的主程序断点,以及其他寄存器或存储器的内容.

当主程序需要调用子程序时,有一组中间结果及标志位的状态需分别保留在寄存器和标志寄存器中

但被调用的子程序执行时,也需要占用这些寄存器并影响标志寄存器,这样除了在执行调用指令时将断点

(调用指令后紧接着的一条指令地址)保存在堆栈中外,还必须将原主程序中保留在寄存器中中间结果和

标志位的状态保留在堆栈中,直到子程序结束,返回主程序时,再将这些中间结果及标志位状态送回寄存器

和标志寄存器中.

6502 的堆栈地址是 0100-01FF,但由于实际上系统也占用了堆栈,所以堆栈指针并不是指向栈底,我们可以

来测试一下.

进入 NCTOOLS 下,A 2000

2000: TSX 堆栈指针低 8 位送寄存器 X

2001: RTS

然后 G 2000,按 R 查看寄存器状态,结果发现 (X) = B4

这里说明堆栈指针的值是 01B4,但这是可变的

下面我们来看看堆栈指针随进栈和出栈的变化情况:

A 2000

2000: TSX ;堆栈初始地址低 8 位送寄存器 X

2001: STX $3000

2004: PHA ;寄存器 A 的数据压入堆栈

2005: TSX ;把这时堆栈地址低 8 位送寄存器 X

2006: STX #3001 ;结果送地址 3001

2009: PHA ;寄存器 A 的数据压入堆栈

200A: TSX ;把这时堆栈地址低 8 位送寄存器 X

200B: STX $3002 ;结果送地址 3002

200E: PLA ;从堆栈中弹出一个数据

200F: TSX ;把这时堆栈地址低 8 位送寄存器 X

2010: STX $3003 ;结果送地址 3003

2013: PLA ;从堆栈中弹出一个数据

2014: TSX ;把这时堆栈地址低 8 位送寄存器 X

2015: STX $3004 ;结果送地址 3004

2018: RTS

然后我们 G 2000

然后 V 3000

看见地址 3000-3004 的内容分别是 B4 B3 B2 B3 B4

说明堆栈指针地址是 01B4 01B3 01B2 01B3 01B4

大家可见,当把一个数据进栈后,堆栈指针地址减 1 了,最开始堆栈指针地址是 01B4,后来不就是 01B3 了

但把一个数据出栈后,是把最近压入堆栈的数据先出栈,大家可以看到,当又压入一个数据进栈后堆栈指针地址为

01B2,但是当我们弹出一个数据后,堆栈指针为 01B3,说明把最近的一个数据弹出了.

所以 6502 堆栈是遵循 "先进后出"的原则,就好象我们把书一本一本的层叠,但我们拿书的时候,拿的却是最上面

的那本.

我举个例子:

```
A 2000

2000: LDA $00       ;地址 00 的内容压入堆栈

2002: PHA

2003: LDA $0A       ;地址 0A 的内容压入堆栈

2005: PHA

2006: LDA $0D       ;地址 0D 的内容压入堆栈

2008: PHA

2009: PLA           ;注意:出栈时,先出的是地址 0D 的内容,所以是把结果送地址 0D,不是地址 00

200A: STA $0D

200C: PLA

200D: STA $0A

200F: PLA

2010: STA $00

2012: RTS
```

当调用子程序时,系统自动将子程序后一指令的地址 - 1 送堆栈,我们举例说明:

子程序从地址 2100 开始:

```
A 2100

2100: LDY #$00

2102: LDA #$01        ;堆栈指针地址高 8 位 送地址 46

2104: STA $46

2106: TSX             ;堆栈指针地址低 8 位 送寄存器 X

2107: INX             ;寄存器 X 内容加 1

2108: STX $45         ;送地址 45

210A: LDA ($45), Y    ;读取目标地址值送地址 3000

210C: STA $3000

210F: INC $45         ;堆栈指针 低 8 位加 1

2111: LDA ($45), Y    ;读取目标地址值送地址 3001

2113: STA $3001

2116: RTS
```

主程序从地址 2000 开始:

```
A 2000

2000: JSR $2100       ;调用子程序 $2100

2003: RTS
```

这里我们 G 2000,V 3000,发现(3000) = 02,(3001) = 20

执行完 JSR $2100 后,应该执行 地址 2003 的指令,为什么是 2002 呢?

因为返回后,程序计数器还会自动加 1.

程序在调用 子程序时,会保存下一指令地址 -1,到堆栈,保存的顺序是 先存地址 高 8 位

再存地址 低 8 位.

//addr :代表 8 位地址 addr16:代表 16 位地址 data :立即数

八、[转移指令]

程序在大多数的情况下是按顺序执行的,即依靠程序计数器 PC 不断自动加 1 的操作,指示出下一条指令所在地址,这样

计算机就可以按照程序中指令的排列顺序一条接一条的执行下去.

不过在某些条件下,需要改变程序顺序执行的次序,而转入执行另一个地址中存放的指令,这就要依靠转移指令来实现

在 6502 中有无条件转移和条件转移,无条件转移是无条件的将程序转向另外一个地址,而条件转移是当满足某些条件时

程序才发生转移,比如 C=0,C=1,Z=1,Z=0 等条件.

无条件转移的跳转步长为整个 64K 内存,即可以跳转到任意地址

条件跳转指令的跳转步长是有限制的 *正跳转 127 个字节 负跳转 128 个字节*

1. JMP--无条件转移指令

符号码格式	指令操作码	寻址方式
JMP $data16	4C	绝对寻址
JMP ($data16)	6C	间接寻址

JMP($data16)原文指令操作码为 5C，但 FC 是 6C，所以编者将之更正。

2. 条件转移指令

符号码格式	指令操作码	寻址方式	指令功能
BEQ $data16	F0	相对寻址	如果标志位 Z=1 则转移，否则继续
BNE	D0	相对	如果标志位 Z=0 则转移，否则继续

$data16		寻址	
BCS $data16	B0	相 对 寻址	如果标志位 C=1 则转移，否则继续
BCC $data16	90	相 对 寻址	如果标志位 C=0 则转移，否则继续
BMI $data16	30	相 对 寻址	如果标志位 N=1 则转移，否则继续
BPL $data16	10	相 对 寻址	如果标志位 N=0 则转移，否则继续
BVS $data16	70	相 对 寻址	如果标志位 V=1 则转移，否则继续
BVC $data16	50	相 对 寻址	如果标志位 V=0 则转移，否则继续

这里我重点讲讲用的最多的 BEQ，BNE，BCC，BCS

①BNE 如果标志位 Z = 0 则转移，否则继续

在 6502 中，要判断两个数是不是相同，就可以使用该指令

例 1： 判断 地址 3000 与地址 3001 的内容是不是相同，若相同，则送 01 到地址 3002，否则送 00

```
A 2000

2000:LDA $3000          ;读取地址 3000 的内容到寄存器 A

2003:CMP $3001          ;和地址 3001 的内容比较,其实就是把地址 3000 的内容减去地址 3001 的内容

2006:BNE $200E          ;若 Z = 0,说明结果不等于 0 ,那么说明两个地址的内容不同,程序转到地址 200E

2008:LDA #$01           ;这里说明 Z = 1,那么说明两个数相同

200A:STA $3002

200D:RTS

200E:LDA #$00

2020:STA $3002

2023:RTS
```

从上面的程序,我们知道,要比较两个数是不是相同,需要先使用比较指令,然后通过标志寄存器的状态位来判断是不是相同

②BEQ 如果标志位 Z = 1,那么就转移,否则继续

例: 判断 地址 3000 与地址 3001 的内容是不是相同，若相同，则送 01 到地址 3002，否则送 00

```
A 2000

2000:LDA $3000

2003:CMP $3001

2006:BEQ $200E

2008:LDA #$00

200A:STA $3002

200D:RTS

200E:LDA #$01

2020:STA $3002

2023:RTS
```

该程序和上面的几乎是一样的,大家分析一下吧.

③BCC 如果标志为 C = 0,转移,否则继续

该指令可以用来判断两个数谁大谁小

例:判断地址 3000 和地址 3001 的内容,谁大谁小,若地址 3000 的内容大,送 01 到地址 3002,若地址 3001 的内容大,送 02 到地址 3002

如果都是一样大,送 00 到地址 3002

```
A 2000

2000:LDA $3000          ;读取地址 3000 的内容到寄存器 A

2003:CMP $3001          ;和地址 3001 的内容比较,其实就是把地址 3000 的内容减去地址 3001 的内容

2006:BEQ $2010          ;如果 Z = 1,那么说明相同,转地址 2010,送 00 到地址 3002
```

2008:BCC $2016 ;如果 C = 0,那么说明地址 3000 的内容 < 地址 3001 的内容,转地址 2016,送 02 到地址 3002

200A:LDA #$01 ;这里说明 C = 1,那么说明地址 3000 的内容 > 地址 3001 的内容

200C:STA $3002

200F:RTS

2010:LDA #$00

2012:STA $3002

2015:RTS

2016:LDA #$02

2018:STA $3002

201B:RTS

④BCS 若 C = 1,则转移,否则继续

和上面的用法是一样的,大家用该指令完成上面的功能吧!

3. 转移到子程序指令 JSR 和从主程序返回指令 RTS

JSR 指令仅仅是 绝对寻址，它的操作码是 20

RTS 指令是 隐含寻址，它的操作码是 60

在程序设计中，如果程序比较大，那么一般是采取模块化设计方法，把一个大的程序分割成若干小程序，然后在主程序调用这些小程序

在 6502 中就是用 JSR 这条指令调用子程序的.

转子指令和转移指令的区别在于转移指令控制程序转出后就不再返回了，而转子指令使程序转向子程序后，当子程序被执行完后还要返回

主程序被打断处，实现这个返回是依靠在子程序末尾使用一条子程序返回指令 RTS，就可以控制程序自动返回到主程序被打断处

例如： 指令

.

.

.

2000:JSR 2100 //程序执行到这里时，就跳转到地址 2100 开始执行那里的程序

2003：.

.

.

2100:指令

.

.

.

RTS //当程序执行到这里后，会自动返回主程序被打断处，即跳转到地址 2003 那里继续执行

九、中断指令

在文曲星内部大量使用了这种指令,该指令占三个字节.

操作符为 INT,机器码为 00

例如 INT $8A01

INT $C001

那么 INT $8A01 是执行那里的程序呢?首先 8A 是页码,01 是字偏移量

我们先转到 8A 页码,看到下面的数据

图(略)

这里我们先要切换到 8A 页码,然后根据 01,知道执行地址是 4085,因为 60EA 不算,从 8540 开始算偏移 1

5549 算偏移 2,所以 INT $8A02 就是执行地址 4955,大家可以自己算.

这里的 INT $C001 就不是转到 C0 页码了,而是当地址(0A) = 00 时,C000-DFFF 那里算

图(略)

这里 INT $C001 就是执行 C059

////////////////////////////////

END