神琴的吞食天地修改教程 ASM指令

80%内容来自大奥秘 修正的大奥秘错误的地方

正确的进阶方式:

先不看寻址 先看指令 看对应寻址方式 在脑海里计算

寻址方式是潜移默化的学习 而不是先学习寻址 在学习指令

最后学习6502 assembly的使用 (可以看后面分析例子)

寻址方式:

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

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 3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.

LDA $3000,该指令的功能是将 地址3000中的内容送寄存器A

STA $3001,该指令的功能是将寄存器A的内容送地址3001

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.

以上三个就是属于没有寄存器参与寻址

LDA举例子

LDA #$80 立即寻址 把数值80直接装载到寄存器A

LDA $0700 直接寻址 把内存$0700的内容装载到寄存器A

LDA $00 零页寻址 同直接寻址 但只能从零页内存(本次地址$00)装载到寄存器A

LDX LDY都同理 只不过是寄存器X,Y

4.累加器寻址 1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 1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 ;......

以上两个属于1字节指令 其实我们也不关心他们的寻址方式

总结来说就是单字节指令都会这样

INX DEX INY DEY (隐含寻址)

PHA PLA PHP PLP (隐含寻址)

ASL LSR ROL ROR (不跟操作数就是累加器寻址 跟了就不是)

6.使用寄存器X的直接变址 3字节指令

为了方便起见,我们称该寻址方式为 直接X变址

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 3字节指令

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

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 2字节指令

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

以上四个指令有x,y寄存器参与

举例子

LDA $8000,X

LDA $8000,Y

LDA $80,X

LDA $80,Y

10.间接寻址 3字节指令

在 6502中,仅仅用于无条件跳转指令 JMP这条指令

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

请注意 大奥秘的5C指令对应这个是错误的 正确的机器码是6C

6502的$6C指令（间接绝对跳转）有一个BUG，当低位字节是$FF时CPU将不能正确计算有效地址。例如：

　 C100: 4F

　 C1FF: 00

　 C200: 23

　 ..

　 D000: 6C FF C1 - JMP ($C1FF)

本来它是应该跳到$2300的，但是在计算高位字节的时候，在页面边界处地址是不能再增加的，所以实际将跳转到$4F00。

11.先变址X后间接寻址 2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

12.后变址Y间接寻址 2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

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

//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 |

[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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 符号码格式 | 指令操作码 | 寻址方式 |
| 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

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

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

//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

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

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 |

[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

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 #$0F (0F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.

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

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.

吞食2基本没怎么用过这个命令

[置标志位指令]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLC | 清除进位标志 | 0→C | 机器码 | 18 |
| SEC | 置进位标志C | 1→C | 机器码 | 38 |
| CLD | 清除十进制运算标志D | 0→D | 机器码 | D8 |
| SED | 置十进制运算标志D | 1→D | 机器码 | F8 |
| CLV | 清除溢出标志V | 0→V | 机器码 | B8 |
| CLI | 清除中断禁止指令I | 0→I | 机器码 | 58 |
| 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

//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

//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

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

//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

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

//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

//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

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

//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

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

[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 |

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 | $data16 | D0 | 相对寻址 | 如果标志位Z=0则转移，否则继续 |
| 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

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

2000:LDA $3000

2003:CMP $3001

2006:BEQ $200E ;一样跳转$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

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 | 绝对寻址 | 它的操作码是 | 20 |
| RTS | 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：

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

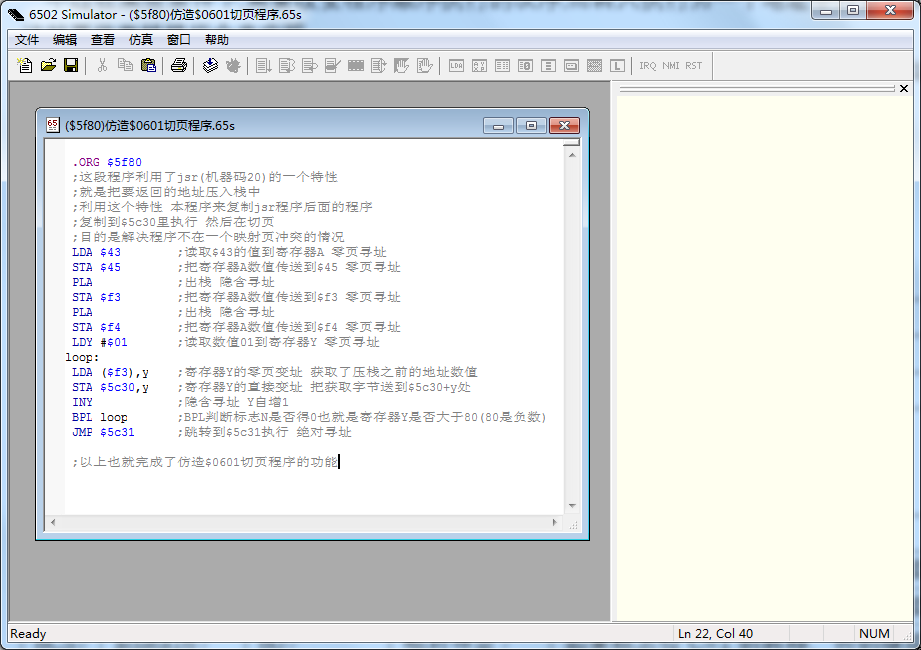
.

.

.

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

6502 assembly 使用方法

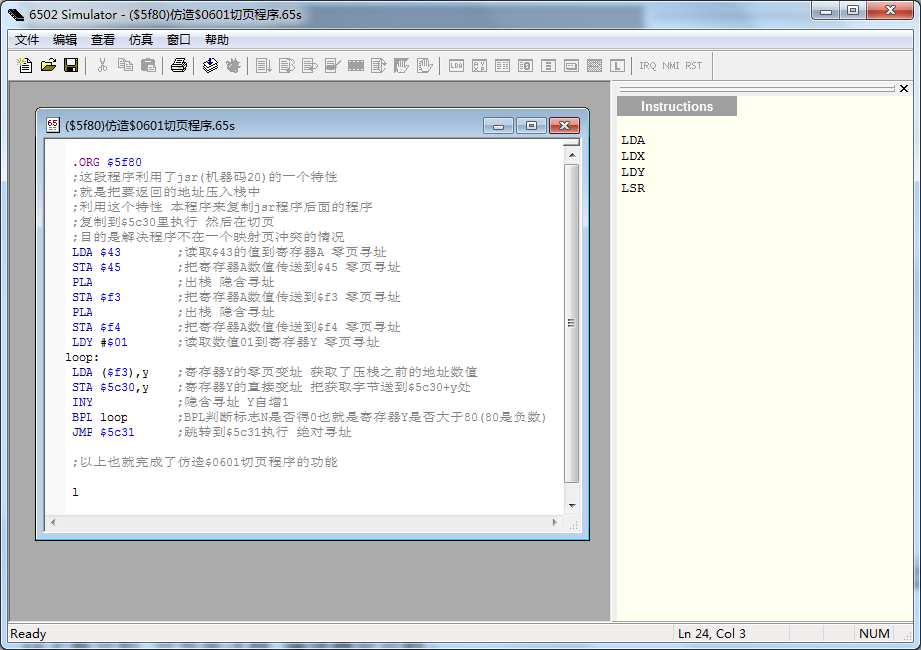


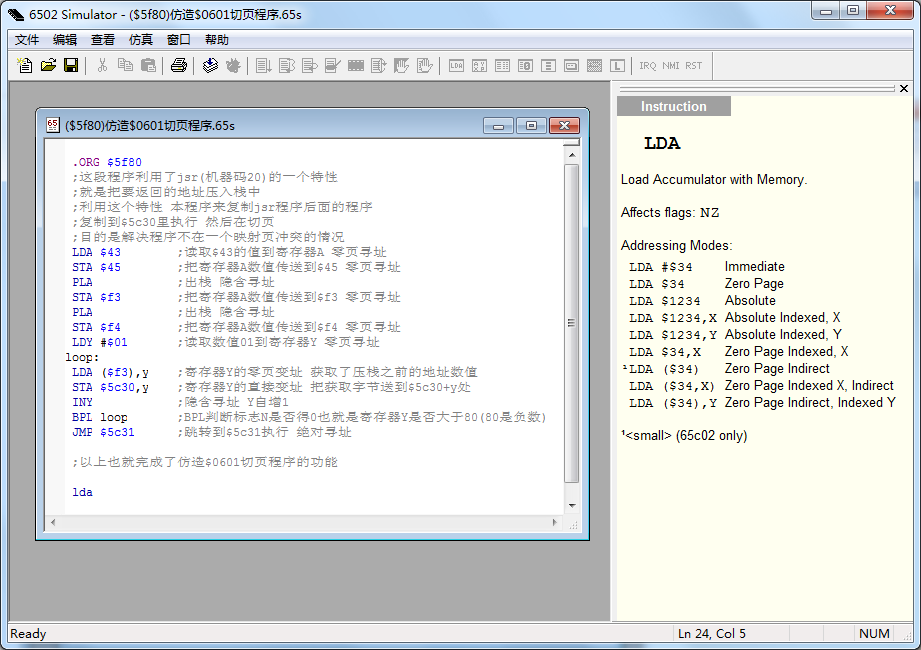
以上有分析 分号是注释 编译器会忽略

.org $5f80 证明程序起始于$5f80(需要空一格 否则会提示错误)

loop:是label标签 以免手算地址用的 条件判断一般跳转符号即可 必须顶格写

写程序的时候 右面会提示

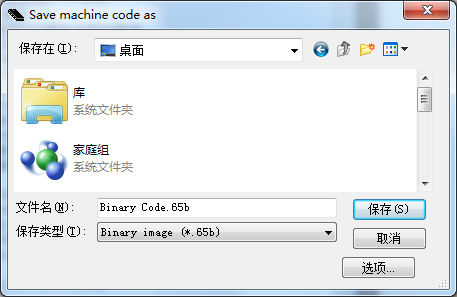




下面讲如何把程序编译并且手动放入rom

首先点仿真-编译

然后点保存代码



保存类型要选择binary image 然后就会生成机器码

用十六进制编辑器打开 .org后面的就是程序所在逻辑地址 也是.65b文件的物理地址



首先你需要知道你程序该放到哪个位置 上一篇讲过RAM映射 同理逆运算

但是我这个有点不同(NTS2R特有)

$5100-$5fff内存死映射到0x80010-0x80f0f

其中 刚刚为了仿造程序 使用了$5c30-$5caf 所以这段不能写程序(\*)

我们把程序复制到对应物理地址保存即可