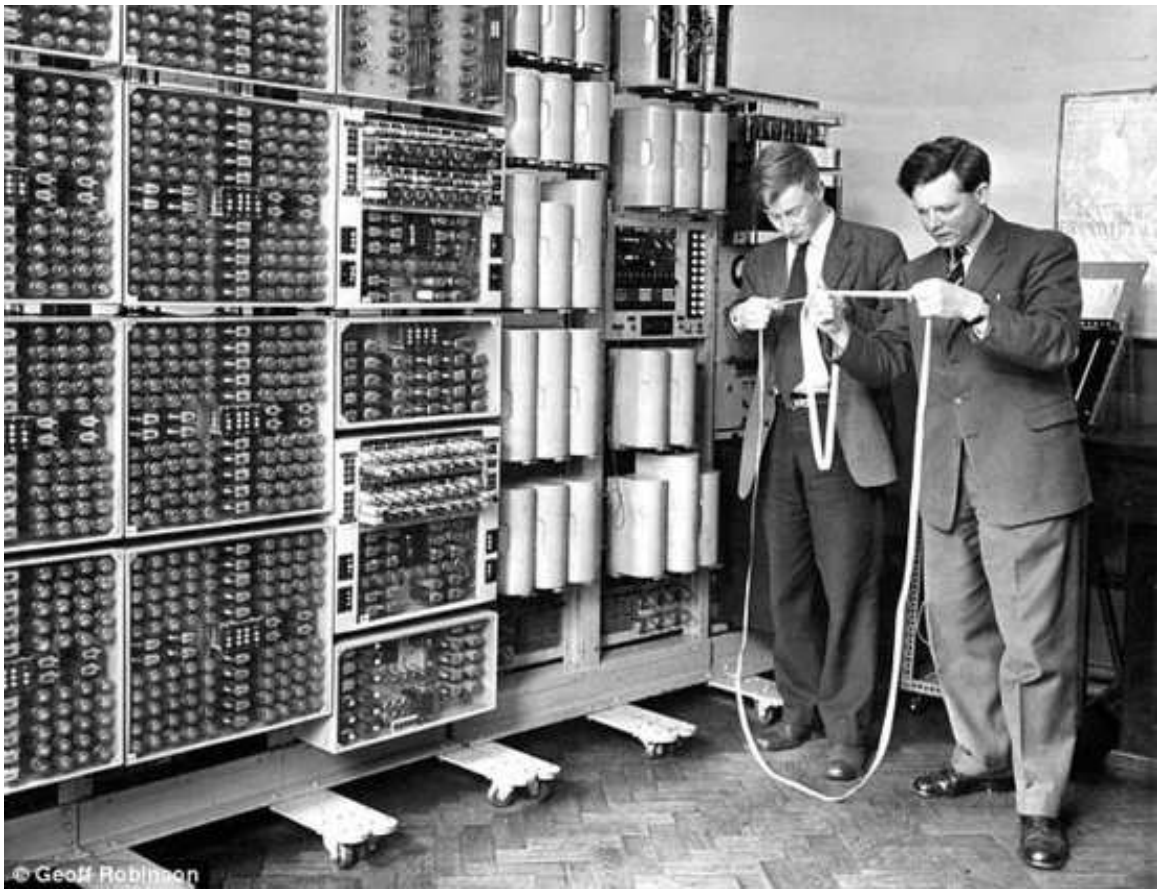
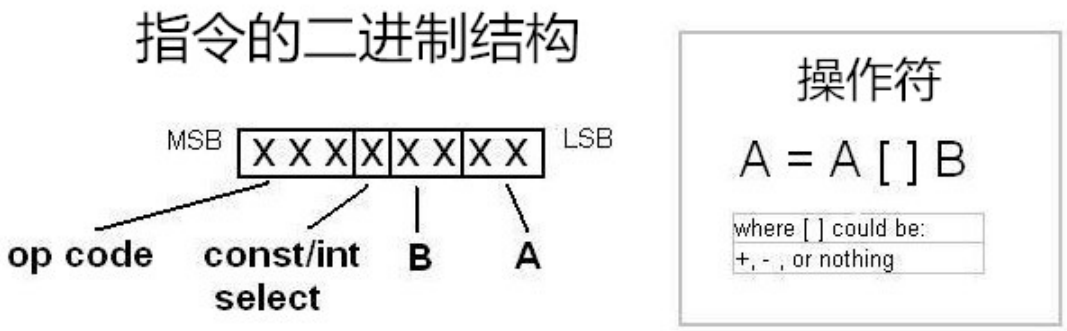
CuteCPU 指令集讲解



编写人：qql

初次编写日期：2017.04.02

最后修改日期：2018.07.24



操作码解释：

OP code	功能	源码示例	功能示意
000	加法	ADD B, A	A=A+B
001	读RAM存储器或按键	RM B, A	
010	进位加	ADDC B, A	A=A+B+C进位
011	写RAM存储器	WM B, A	
100	减法	SUB B, A	A=A-B
101	IO输出	OUT B, A	
110	< 小于比较	< B, A	A-B
111	= 等于比较	= B, A	A-B

常数位寄存器位解释：

当常数位 const/int select 是 0 时，B 的二进制值表示寄存器，如：B=00，表示 R0, B=01，表示 R1, B=10，表示 R2, B=11，表示 R3。

当常数位 const/int select 是 1 时，B 的二进制值表示常数值，如：B=00，表示常数 0, B=01，表示常数 1, B=10，表示常数 2, B=11，表示常数 3。

注意点：

可以看到常数最大只能表示到 3，如果想表示大一些的数，比如指令 ADD 7，R3 怎么办呢？实际上这个写法是错误的，只能分开几条加法指令填写，如要加 7，需要两条加 3 指令和一条加 1 指令。

指令示例讲解：

一、加法指令：ADD B,A

例子如： ADD R0,R2 功能是 R2 = R2+R0
ADD 3,R3 功能是 R3 = R3+3

二、读RAM存储器指令： RM B,A

例子如： RM R0,R2 功能是 R2 = [R0] 即将地址R0的RAM的值赋给R2
RM 3,R3 功能是 R3 = [3] 即将地址3的RAM的值赋给R3

三、进位加法指令：ADDC B,A

例子如： ADDC R0,R2 功能是 R2 = R2+R0+C (C表示进位值：1或者0)
ADDC 3,R3 功能是 R3 = R3+3+C

四、写RAM存储器指令： WM B,A

例子如： WM R0,R2 功能是 [R0] = R2 即将R2的值赋给地址R0的RAM
WM 3,R3 功能是 [3] = R3 即将R3的值赋给地址3的RAM

五、减法指令：SUB B,A

例子如： SUB R0,R2 功能是 R2 = R2-R0
SUB 3,R3 功能是 R3 = R3-3

六、IO输出指令： OUT B,A

例子如： OUT R0,R2 功能是 [R0] = R2 即将R2的值赋给地址R0的IO
OUT 3,R3 功能是 [3] = R3 即将R3的值赋给地址3的IO

七、< 比较指令：< B,A

例子如： < R0,R2
GOTO addr123
功能是 R2-R0 得到的结果不回写寄存器，如果结果小于0为假（注意是为假，即结果大于或等于0），就跳转到地址标号addr123处， 注意这里的 < 判断指令后面必须跟着GOTO指令，两条指令要一起写才行。
< 3，R3
GOTO addr456
同上面一样

八、= 比较指令：= B,A

例子如： = R0，R2
GOTO addr123
功能是 R2-R0 得到的结果不回写寄存器，如果结果等于0为假（注意是为假，即结果大于或小于0），就跳转到地址标号addr123处， 注意这里的 = 判断指令后面必须跟着GOTO指令，两条指令要一起写才行。
= 3，R3

GOTO addr456

同上面一样

注意：使用<，= 比较指令进行比较就是为了后面跳转服务的，所以这两条必须要写在一起，而且在比较指令前面需要添加 R3 的赋值，如：

```
QCLR    R3
<       R0,R0
GOTO 0
```

具体使用注意点请看后面自定义指令的讲解。

额外自定义指令:

由于原始指令只有 8 条，有时候不是很直观和方便，所以增加了自定义的指令，每一个自定义指令都是以 Q 开头，自定义指令一共有：

```
QCLR    清空
QMOV    赋值
QADD    加法
QSUB    减法
QGOTO   长跳转
QJMP    立即跳转
```

==========

```
QCLR    清空
如 QCLR  R1
实际上经过汇编器后指令为
SUB  R1,R1
```

==========

```
QMOV    赋值
如 QMOV 2,R1
实际上经过汇编器后指令为
SUB  R1,R1
ADD  2,R1
```

使用这条指令可以把大于 3 的常数赋值给寄存器（但是最大不能大于 15），

如 QMOV 15,R1 这时候汇编器就翻译成 5 条 ADD 3，R1 指令

==========

```
QADD    加法
如 QADD 15,R1
```

汇编器就翻译成 5 条 ADD 3，R1 指令

```
ADD  3,R1
ADD  3,R1
ADD  3,R1
ADD  3,R1
ADD  3,R1
```

==========

```
QSUB    减法
```

与上面的 QADD 指令类似

==========

GOTO　小跳转
QGOTO　长跳转
QJMP　立即跳转
跳转稍微有点复杂，首先大家知道，跳转的原理就是将要跳转的地址值加载到 74163 计数器，这样下一条指令就是这个地址的指令。

查看电路原理图，可以看到指令存储器的地址线有 12 根，分别由 3 个 74163 计数器提供，最低 8 位接到了指令存储器输出端，最高 4 位接到了 R3 寄存器。所以地址也需要这两部分提供。看如下指令：

```
QCLR    R3
<       R0,R0
GOTO 0
```

首先将 R3 清零(因为地址是 0)，然后判断 < R0,R0，必定为假，所以再下一条指令放置跳转地址的低 8 位，这里是 0（GOTO 0）这样 12 根地址线上都是 0，跳转地址就是 0 了。可以看到使用 goto 后面地址最大是 255

实际上除了 0 地址，想跳转到其他地方，地址数值到底是多少是很难知道的，所以建议利用类似于 C 语言的标号。程序可以修改为：

```
start:
  ...
  ...
QCLR    R3
<       R0,R0
GOTO start
```

标号在 GOTO 前方和后方都可以，需要注意的是这个标号代表的地址范围还是数值 255 内，所以建议使用 QGOTO，这样程序可以修改为：

```
start:
  ...
  ...
<       R0,R0
QGOTO start
```

可以看到减少了代码 QCLR　R3，跳转地址范围可以达到 4K，为什么这里的 R3 不需要赋值呢？实际上是汇编器计算出 R3 的值，自动插入一句给 R3 赋值的代码。如果不需要判断就立即跳转，可以使用 QJMP，代码简化为：

```
start:
  ...
  ...
 QJMP start
```