Automachef

AC-32
可编程电脑指南

目录

系统架构

AC-32 是一款高精尖可编程电脑，可以通过输出插针同时对最多四台其他设备进行控制（名称从 O0 到 O3）。电脑启动或重置时，所有输出均默认设为关闭。此电脑内置八个 16 位寄存器，可以用于任何用途（名称从 V0 到 V7）。它们在本指南手册中会被称为变量，可以储存 -32,768 到 32,767 之间的整数值。

有一些厨房部件可以支持读取其进行行动的次数，比如食材门、组装机与机械臂等。这些部件从启动起进行的行动次数，可以通过读取 I0 至 I3 变量进行获取。I0 会读取与 O0 连接的设备进行的行动次数，I1 对应 O1，以此类推。不支持读取这一数值的部件（比如传送带等）会自动返还结果 0。

这台电脑的主程序内存最多可存储 32 行使用 AC 汇编语言编写的代码。同时，本电脑还配备四个额外程序页面，每个页面最多可存储 32 行代码，可以通过 cal 指令调用。具体相关细节请查看下方的指示说明。整个程序每秒运行 30 次，方便程序中需要精准计时的例行程序计时使用。说到计时，只需读取特殊的只读寄存器 TT，即可读取当前时间（24 小时制格式）。举例来说，如果当前时间是下午 3:45，TT 寄存器就会含有 16 位整数 1545。

本电脑还包含一套四个特别的只读寄存器，并自动由内置的订单读取器获得的信息进行填充。若在当前 33 毫秒的时钟周期内收到了至少一个制定种类的新订单，这些寄存器中就会填充大于 0 的数字；如果没有订单，寄存器中的数字则是 0。该数字代表着在当前的 33 毫秒时钟周期内收到该种类新订单的数量。这些寄存器可在 AC 汇编语言中使用 R0 至 R3 进行调用。

电脑内置的订单读取器还可以区分来自不同来源的订单，如汽车订餐窗口或顾客订购外卖食物等。为此，你可以对 R0 至 R3 变量增添以下字母作为后缀：

R 代表订单来自餐馆。

T 代表订单来自外卖。

D 代表订单来自汽车订餐窗口。

示例

内置订单读取器 R2 已设定为检测奶酪汉堡。

变量 R2R 储存的数字代表着餐馆中订购奶酪汉堡的数量。

变量 R2T 储存的数字代表着外卖订购奶酪汉堡的数量。

变量 R2D 储存的数字代表着汽车订餐窗口订购奶酪汉堡的数量。

变量 R2 储存的数字代表着当前 33 毫秒时钟周期中新订购奶酪汉堡的总量。

因此，R2 储存的数字也就是 R2R + R2T + R2D 之和。

语言偏好

标签

标签是代码中的命名位置，可以与 jump (jmp/jne/jeq/jgt/jlt) 指令配合，更改程序执行的流程。标签仅可包含英文字母，长度必须在 1-10 字节之间，且必须用分号结尾。

示例

```
loopagain:
```

```
belton:
```

```
endprogram:
```

ADD 指令

add 指令可以将两个数值进行相加，并将结果存入第三个参数指定的变量中。记住，寄存器都是 16 位大小，所以结果必须在 -32768 到 32767 之间，否则便会发生错误。

语法

```
add <operand1> <operand2> <operand3>
<operand1> 可以是变量，也可以是整数值。
<operand2> 可以是变量，也可以是整数值。
<operand3> 必须是变量。
```

示例

```
add V1 15 V2
```

```
add V0 V1 V0
```

CAL 指令

cal 指令（call，调用的缩写）仅可以在主代码页中进行使用。这一指令会自动执行指定代码页中的全部指令，并在结束后自动继续执行下一行代码。

编程时，可以将代码页当成可以调入主程序进行使用的步骤。由于 AC-32 电脑不具备调用栈，因此也无法在代码页中调用其他代码页。

语法

```
cal <operand1>
<operand1> 应该是 1 到 4 之间的整数。这一数值代表着要调用的代码页页码。
```

示例

```
cal 2
```

CMP 指令

cmp 指令可以将两个数值进行比较，并将比较寄存器设定为 -1，0 或 1。如果第一个数值小于第二个数值，寄存器就会被设定为 -1。如果

两个数值相等，寄存器就会设定为 0。如果第一个数值大于第二个数值，寄存器就会设定为 1。这一结果与 jne/jeq/jlt/jgt 指令结合，可以令程序跳转到另一部分代码，进行执行。

语法

```
cmp <operand1> <operand2>
<operand1> 可以是变量，也可以是整数值。
<operand2> 可以是变量，也可以是整数值。
```

示例

```
cmp V1 30
```

```
cmp V1 V3
```

DEC 指令

dec 指令会令指定的变量数值减少 1，但不会让变量降低为负数，因此会自动在数值到达 0 时停止。对设定计时功能有一定作用。

语法

```
dec <operand1>
<operand1> 必须是变量。
```

示例

```
dec V0
```

```
dec V3
```

JMP/JNE/JEQ/JGT/JLT 指令

这些指令都会跳至指定的标签。jne 代表 " 若不相等，则进行跳转 "，jeq 代表 " 若相等，则进行跳转 "，jlt 代表 " 若小于，则进行跳转 "，而 jgt 代表 " 若大于，则进行跳转 "。如果上一次使用 cmp 指令进行的比较结果符合指令的名称，则会跳转到指定的标签。举例来说，在比较后运行 jne 指令，则只会在相应的比较结果认定两个数值不相等时进行跳转；jgt 指令只会在比较中的第一个参数大于第二个参数时进行跳转，以此类推。jmp 指令永远会（无条件地）跳转至指定的标签。

语法

```
jmp <operand1>
jne <operand1>
jeq <operand1>
jlt <operand1>
jgt <operand1>
 <operand1> 必须是标签。
```

示例

```
jne endprogram
```

```
jlt loopagain
```

MOV 指令

mov 指令可将一个数值复制到特定的变量之中。

语法

```
mov <operand1><operand2>
<operand1> 可以是变量，也可以是整数值。
<operand2> 必须是变量。
```

示例

```
mov 30 V2
```

```
mov V1 V2
```

MUL 指令

mul 指令可以将两个数值相乘，并将结果存入第三个参数指定的变量中。该指令仅适用于 AC-32 电脑。记住，寄存器都是 16 位大小，所以结果必须在 -32768 到 32767 之间，否则便会发生错误。

语法

```
mul <operand1> <operand2> <operand3>
<operand1> 可以是变量，也可以是整数值。
<operand2> 可以是变量，也可以是整数值。
<operand3> 必须是变量。
```

示例

```
mul V1 15 V2
```

```
mul V0 V1 V0
```

NOT 指令

not 指令会切换变量的数值。若数值为 0，则变为 1；若数值为 1，则变为 0。

语法

```
not <operand1>
<operand1> 必须是变量。
```

示例

```
not V1
```

```
not V3
```

OUT 指令

out 指令会控制连接至相应输出插针的设备，将其启动或关闭。如果第二个 operand 操作变量为 0，设备则会关闭。若第二个 operand 操作变量为 0 以外的任意其他数值，都会将输出设定为启动。

语法

```
out <operand1><operand2>
<operand1> 必须是输出。
<operand2> 可以是变量，也可以是整数值。0 代表关闭，其他数字代表启动。
```

示例

```
out O2 1
```

```
out O2 V3
```

RET 指令

ret （代表返回）指令会完成当前 33 毫秒时钟周期中代码的执行。该指令相当于跳转至代码最后一行的标签。此指令没有 operand 操作变量

语法

```
ret
```

示例

```
ret
```

SUB 指令

sub 指令会计算两个数值之间的差别，并将结果存入第三个参数指定的变量中。记住，寄存器都是 16 位大小，所以结果必须在 -32768 到 32767 之间，否则便会发生错误。基本来说，它会进行 "operand1 - operand2" 并将结果存入 operand3 指定的变量中。

语法

sub <operand1> <operand2> <operand3>

<operand1> 可以是变量，也可以是整数值。

<operand2> 可以是变量，也可以是整数值。

<operand3> 必须是变量。

示例

```
sub V1 15 V1
```

```
sub V2 V3 V0
```

示例程序 1

这个示例程序会将连接至 O1 的设备启动一秒，然后关闭一秒，如此往复。

```
add 1 V0 V0
cmp 30 V0
jne endprogram
mov 0 V0
not V1
out O1 V1

endprogram:
```

示例程序 2

这个示例程序会在收到 5 个订单后启动连接至 O2 的设备。

```
add V0 R0 V0
cmp V0 5
jlt endprogram
out O2 1

endprogram:
```

示例程序 3

这个示例程序每收到一个新订单，就会令连接至 O0 的设备持续启动 5

秒。此程序的一大用处就是，在每次收到一个新订单时，让分配器分配一份食材。请注意，此程序也会让设备在启动时热机 4 秒，以便收到订单时立即分配出食材，为你的顾客节省宝贵的时间！普通的订单读取器可做不到这一点，对吧？

```
prewarm:
cmp V1 1
jeq alrdywarm
add 120 V0 V0
mov 1 V1

alrdywarm:
cmp R0 1
jne nonew
add 150 V0 V0

nonew:
cmp V0 0
jlt timerended
sub V0 1 V0
out O0 1
ret

timerended:
out O0 0
```

示例程序 4

这个复杂的示例程序会从两个不同的订单读取模块（R0 和 R1）进行读取，并处理输出 O0，O1 和 O2。这一程序的目的，就是在 R0 或 R1 收到订单时启动 O0 和 O1 三秒钟，但在 R1 收到订单时仅启动 O2。此程序的一大用处就是，将 O0 连接至生肉饼分配器，将 O1 连接至汉堡面包分配器，并将 O2 连接至奶酪分配器，然后让 R0 处理普通汉堡，让 R1 处理奶酪汉堡。这一指

令还会在启动分配器时令其热机两秒。

```
prewarm:
cmp V2 1
jeq checkorder
add V0 60 V0
add V1 60 V1
mov 1 V2

checkorder:
cmp R0 1
jeq addtime
cmp R1 1
jeq addtimes

main:
out O0 V0
out O1 V0
out O2 V1
dec V0
dec V1
ret

addtimes:
add V1 90 V1
addtime:
add V0 90 V0
jmp main
```