Automachef

# REFERENCIA DEL ORDENADOR PROGRAMABLE **AC-16**

# ÍNDICE DE CONTENIDOS

# ARQUITECTURA DEL SISTEMA

El AC-16 es un ordenador programable avanzado capaz de controlar hasta cuatro dispositivos diferentes mediante los pins de salida, enumerados del 00 al 03. Todas las salidas están configuradas por defecto en la posición desactivada cuando el ordenador se enciende o se reinicia. Tiene cuatro registradoras de 16 bit integradas que pueden usarse para cualquier propósito, de la V0 a la V3. En este manual de referencia también las llamamos variables y son capaces de almacenar números enteros entre -32 768 y 32 767.

Algunas piezas de la cocina pueden leer el número de acciones que han realizado, como las puertas de ingredientes, las montadoras y los brazos robóticos. Se puede obtener el número de acciones que han realizado desde que se activaron leyendo las variables I0 a I3. La I0 leerá el número de acciones realizadas por el dispositivo conectado a 00, la I1 el del 01, y así con todas. Las piezas que no pueden leer este valor (como las cintas transportadoras) siempre darán 0.

El programa de memoria del ordenador puede contener hasta 32 líneas de código escritas en el lenguaje ensamblador AC. El programa completo se ejecuta exactamente 30 veces por segundo, por lo que es fácil programar rutinas que requieran una sincronización precisa. Además, al acceder al registro TT especial de solo lectura, podrás ver la hora actual del día en formato de 24 horas. Así que si son las 3:45 p. m., el registro TT contendrá el número entero de 16 bit 1545.

Hay un total de cuatro registros especiales de solo lectura que se completarán automáticamente con información del lector de pedidos integrado. Cada uno de estos registros contendrá un número mayor que 0 si se ha recibido al menos un pedido del tipo especificado en el intervalo actual de 33 milisegundos. En caso contrario, será 0. El número indica cuántos nuevos pedidos de ese tipo se han recibido en el intervalo actual de 33 milisegundos. En las instrucciones del lenguaje ensamblador AC se usan las variables R0 a R3.

Los lectores de pedidos integrados también pueden distinguir entre pedidos procedentes de diferentes fuentes, como una ventanilla del servicio "Sobre ruedas" o clientes que piden comida para llevar. Con ese fin, puedes añadir las siguientes letras como sufijos a las variables R0 a R3:

R: pedidos procedentes del restaurante.

T: pedidos procedentes de la zona para llevar.

D: pedidos procedentes de la ventanilla del servicio "Sobre ruedas".

## Ejemplos

Lector de pedidos integrado R2 configurado para detectar hamburguesas con queso.

La variable R2R contendrá el número de hamburguesas con queso pedidas en el restaurante.

La variable R2T contendrá el número de hamburguesas con queso pedidas en la zona para llevar.

La variable R2D contendrá el número de hamburguesas con queso pedidas en la ventanilla del servicio "Sobre ruedas".

La variable R2 contendrá el número total de hamburguesas con queso pedidas en el intervalo actual de 33 milisegundos.

R2, por tanto, contendrá la suma de R2R + R2T + R2D.

# LENGUAJE

## ETIQUETAS

Las etiquetas son ubicaciones con nombre en el código que pueden usarse para intercambiar el flujo de ejecución mediante las instrucciones jump (jmp/jne/jeq/jgt/jlt). Llevan un signo de dos puntos al final y solo pueden contener entre 1 y 10 caracteres alfabéticos.

### EJEMPLOS

```
loopagain:
```

```
belton:
```

```
endprogram:
```

## INSTRUCCIÓN ADD

La instrucción add suma dos valores y almacena el resultado en la variable especificada en el tercer parámetro. Recuerda que los registros son de 16 bit, de manera que el resultado debe estar entre -32 768 y 32 767 para evitar errores.

### SINTAXIS

```
add <operand1> <operand2> <operand3>
<operand1> puede ser una variable o un número entero.
<operand2> puede ser una variable o un número entero.
<operand3> tiene que ser una variable.
```

### Ejemplos

```
add V1 15 V2
```

```
add V0 V1 V0
```

## INSTRUCCIÓN CMP

La instrucción cmp compara dos valores y establece el registro de comparación entre -1, 0 o 1. Si el primero es menor que el segundo, se establecerá en -1. Si son iguales, se establecerá en 0. Si el primero es mayor que el segundo, se establecerá en 1. El resultado puede usarse para saltar condicionalmente a una parte distinta del código con las instrucciones jne/jeq/jlt/jgt.

### Sintaxis

```
cmp <operand1> <operand2>
<operand1> puede ser una variable o un número entero.
<operand2> puede ser una variable o un número entero.
```

### Ejemplos

```
cmp V1 30
```

```
cmp V1 V3
```

## INSTRUCCIÓN DEC

La instrucción dec reducirá la variable especificada en uno, pero esta nunca alcanzará valores negativos, así que parará automáticamente en 0. Esto es especialmente útil para implementar cronómetros.

### SINTAXIS

```
dec <operand1>
<operand1> tiene que ser una variable.
```

### EJEMPLOS

```
dec V0
```

```
dec V3
```

## INSTRUCCIONES JMP/JNE/JEQ/JGT/JLT

Todas estas instrucciones permiten saltar a la etiqueta especificada. jne significa "saltar si no es igual", jeq es "saltar si es igual", jlt es "saltar si es menor que" y jgt es "saltar si es mayor que". Saltarán a la etiqueta especificada si el resultado de la última comparación con la instrucción cmp coincide con el resultado del nombre de la instrucción. Por ejemplo, al ejecutar una instrucción jne tras una comparación, esta saltará a la etiqueta especificada solo si dicha comparación determinó que los parámetros no eran iguales, una

instrucción jgt solo saltará si el primer parámetro de la comparación era mayor que el segundo. La instrucción jmp siempre (incondicionalmente) saltará a la etiqueta especificada.

### Sintaxis

```
jmp <operand1>
jne <operand1>
jeq <operand1>
jlt <operand1>
jgt <operand1>
<operand1> tiene que ser una etiqueta.
```

### Ejemplos

```
jne endprogram
```

```
jlt loopagain
```

## INSTRUCCIÓN MOV

La instrucción mov copia un valor en la variable especificada.

### Sintaxis

```
mov <operand1> <operand2>
<operand1> puede ser una variable o un número entero.
<operand2> tiene que ser una variable.
```

### Ejemplos

```
mov 30 V2
```

```
mov V1 V2
```

## INSTRUCCIÓN NOT

La instrucción not simplemente alterna el valor de una variable. Si era 0, se convertirá en 1. De lo contrario, se convertirá en 0.

### Sintaxis

```
not <operand1>
<operand1> has to be a variable.
```

### Ejemplos

```
not V1
```

```
not V3
```

## INSTRUCCIÓN OUT

La instrucción out ordena al dispositivo conectado a la salida especificada que se active o desactive. Si el segundo operando es 0, se desactivará. Cualquier otro valor hará que la salida cambie a la posición activada.

### Sintaxis

```
out <operand1> <operand2>
<operand1> tiene que ser una salida.
<operand2> puede ser una variable o un número entero. 0 significa desactivado y cualquier otra cosa significa activado.
```

### Ejemplos

```
out O2 1
```

```
out O2 V3
```

## INSTRUCCIÓN RET

La instrucción ret (del inglés "return") finaliza la ejecución del código para este ciclo de 33 milisegundos. Significa lo mismo que saltar a una etiqueta colocada en la última línea de código. No tiene operandos.

### Sintaxis

```
ret
```

### Ejemplos

```
ret
```

# INSTRUCCIÓN SUB

La instrucción sub calcula la diferencia entre dos valores y almacena el resultado en la variable especificada en el tercer parámetro. Recuerda que los registros son de 16 bit, de manera que el resultado debe estar entre -32 768 y 32 767 para evitar errores. Básicamente, calculará "operand1 - operand2" y almacenará el resultado en la variable especificada en operand3.

## SINTAXIS

```
sub <operand1> <operand2> <operand3>
<operand1> puede ser una variable o un número entero.
<operand2> puede ser una variable o un número entero.
<operand3> tiene que ser una variable.
```

## EJEMPLOS

```
sub V1 15 V1
```

```
sub V2 V3 V0
```

## *PROGRAMA EJEMPLO 1*

Este programa ejemplo activa el dispositivo conectado a O1 por un segundo, lo desactiva por un segundo, y así sucesivamente.

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

endprogram:
```

## PROGRAMA EJEMPLO 2

Este programa ejemplo activa el dispositivo conectado a O2 después de recibir 5 pedidos.

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

endprogram:
```

## PROGRAMA EJEMPLO 3

Este programa ejemplo mantiene activo el dispositivo conectado a

O0 durante 5 segundos con cada nuevo pedido. Puede resultar muy útil para hacer que una dispensadora produzca un ingrediente con cada nuevo pedido. ¡Ten en cuenta que este programa también precalentará el dispositivo durante 4 segundos al inicio, de manera que los ingredientes se dispensarán casi inmediatamente tras la llegada del pedido, lo que te permitirá ahorrarles tiempo a tus clientes! Un lector de pedidos estándar no puede hacer eso, ¿verdad?

```
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
```

## PROGRAMA EJEMPLO 4

Este complejo programa ejemplo leerá de dos módulos de lectura de pedidos distintos (R0 y R1) y gestionará las salidas O0, O1 y O2. El objetivo del programa es activar O0 y O1 durante tres segundos cada vez que llega un nuevo pedido a R0 o R1, pero solo activar O2 cuando

llega un nuevo pedido a R1. Puede resultar muy útil para hacer que R0 se ocupe de las hamburguesas simples y R1 de las hamburguesas con queso. En este caso, O0 se conectaría a una dispensadora de hamburguesas crudas, O1 a una dispensadora de panes de hamburguesas y O2 a una dispensadora de queso. También precalienta las dispensadoras durante dos segundos al inicio.

```
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
```