Automachef

RIFERIMENTO COMPUTER PROGRAMMABILE **AC-16**

INDICE

ARCHITETTURA DI SISTEMA

L'AC-16 è un computer programmabile all'avanguardia, in grado di gestire fino a quattro altri dispositivi attraverso i connettori di output, da O0 a O3. Di default, tutti gli output sono impostati su off quando il computer viene acceso o resettato. In esso sono cablati quattro registri a 16 bit con segno, chiamati da V0 a V3, che possono essere usati per qualunque scopo. In questo manuale di riferimento sono anche chiamati variabili e sono in grado di memorizzare numeri interi da -32.768 a 32.767.

Alcune parti della cucina forniscono supporto per leggere il numero di azioni che hanno compiuto, come porte ingredienti, assemblatori e bracci robot. Il numero di azioni che hanno compiuto dal loro avvio può essere recuperato leggendo le variabili da I0 a I3. I0 legge il numero di azioni compiute dal dispositivo connesso a O0, I1 da quello connesso a O1 e così via. Parti che non supportano la lettura di questo valore (come ad esempio i nastri trasportatori) restituiranno sempre 0.

La memoria di programma del computer può contenere fino a 32 righe di codice scritte in linguaggio AC Assembly. L'intero programma è eseguito esattamente 30 volte al secondo, facilitando la programmazione di procedure che richiedano una temporizzazione precisa. E a proposito di temporizzazione, dallo speciale registro TT di sola lettura puoi accedere all'ora corrente del giorno, in formato 24 ore. Perciò, se sono le 3:45 del pomeriggio, il registro TT conterrà l'intero a 16 bit 1545.

Ci sono quattro speciali registri di sola lettura che vengono popolati in maniera automatica con le informazioni provenienti dal lettore di ordinazioni integrato. Ognuno di questi registri conterrà un numero maggiore di 0 se è arrivata almeno una nuova ordinazione del tipo specificato nel corrente intervallo di 33 millisecondi; 0 in caso contrario. Il numero specifica quante nuove ordinazioni di quel

tipo sono state ricevute nel corrente intervallo di 33 millisecondi. Nelle istruzioni del linguaggio AC Assembly si può far riferimento ad esso con i nomi da R0 a R3.

Il lettore di ordinazioni integrato può anche fare distinzione tra gli ordini che provengono da fonti diverse, come il finestrino di un'automobile o i clienti che ordinano cibo da asporto. A questo scopo puoi aggiungere le seguenti lettere come suffissi alle variabili da R0 a R3:

R per ordinazioni provenienti dal Ristorante.

T per ordinazioni provenienti dall'area da Asporto.

D per ordinazioni provenienti dal finestrino dell'Automobile.

Esempi

Il Lettore di ordinazioni integrato R2 è impostato per riconoscere i Cheeseburger.

La variabile R2R conterrà il numero di Cheeseburger ordinati nel Ristorante.

La variabile R2T conterrà il numero di Cheeseburger ordinati nell'area da Asporto.

La variabile R2D conterrà il numero di Cheeseburger ordinati dall'Automobile.

La variabile R2 conterrà il numero totale di Cheeseburger ordinati nel corrente intervallo di 33 millisecondi.

R2 conterrà, perciò, la somma di R2R + R2T + R2D.

RIFERIMENTO LINGUAGGIO

ETICHETTE

Le etichette sono punti del codice dotati di nome che possono essere usati per modificare il flusso di esecuzione mediante le istruzioni jump (jmp/jne/jeq/jgt/jlt). Possono contenere solo caratteri alfabetici e devono essere lunghe da 1 a 10 caratteri seguiti dai due punti.

ESEMPI

```
loopagain:
```

```
belton:
```

```
endprogram:
```

ISTRUZIONE ADD

L'istruzione add somma due valori e ne salva il risultato nella variabile specificata nel terzo parametro. Ricorda, i registri sono a 16 bit, quindi il risultato deve essere contenuto tra -32.768 e 32.767 per evitare errori.

SINTASSI

```
add <operand1> <operand2> <operand3>
<operand1> può essere una variabile o un numero intero.
<operand2> può essere una variabile o un numero intero.
<operand3> deve essere una variabile.
```

Esempi

```
add V1 15 V2
```

```
add V0 V1 V0
```

ISTRUZIONE CMP

L'istruzione cmp confronta due valori e imposta il registro di confronto su -1, 0 o 1. Se il primo valore è minore del secondo, lo imposterà su -1. Se sono uguali, lo imposterà su 0. Se il primo valore è maggiore del secondo, lo imposterà su 1. Il risultato può essere usato per eseguire un salto condizionato a una diversa parte del codice usando le istruzioni jne/jeq/jlt/jgt.

Sintassi

```
cmp <operand1> <operand2>
<operand1> può essere una variabile o un numero intero.
<operand2> può essere una variabile o un numero intero.
```

Esempi

```
cmp V1 30
```

```
cmp V1 V3
```

ISTRUZIONE DEC

L'istruzione dec decrementa la variabile specificata di uno, ma non permette di raggiungere numeri negativi, quindi si fermerà automaticamente a 0. Risulta particolarmente utile per implementare i timer.

SINTASSI

```
dec <operand1>
<operand1> deve essere una variabile.
```

ESEMPI

```
dec V0
```

```
dec V3
```

ISTRUZIONI JMP/JNE/JEQ/JGT/JLT

Tutte queste istruzioni saltano all'etichetta specificata. jne sta per "jump if not equal" ("salta se non è uguale"), jeq sta per "jump if equal" ("salta se è uguale"), jlt sta per "jump if less than" ("salta se è minore di") e jgt sta per "jump if greater than" ("salta se è maggiore di"). Se il risultato dell'ultimo confronto fatto usando l'istruzione cmp coincide con il risultato presente nel nome dell'istruzione, si avrà un salto all'etichetta specificata. Per esempio, l'esecuzione di un'istruzione jne dopo un confronto farà saltare all'etichetta specificata solo se detto confronto ha determinato che i

parametri non erano uguali, un'istruzione jgt farà saltare solo se il primo parametro era più grande del secondo e così via. L'istruzione jmp fa sempre saltare (in maniera incondizionata) all'etichetta specificata.

Sintassi

```
jmp <operand1>
jne <operand1>
jeq <operand1>
jlt <operand1>
jgt <operand1>
<operand1> deve essere un'etichetta.
```

Esempi

```
jne endprogram
```

```
jlt loopagain
```

ISTRUZIONE MOV

L'istruzione mov copia un valore nella variabile specificata.

Sintassi

```
mov <operand1> <operand2>
<operand1> può essere una variabile o un numero intero.
<operand2> deve essere una variabile.
```

Esempi

```
mov 30 V2
```

```
mov V1 V2
```

ISTRUZIONE NOT

L'istruzione not restituisce semplicemente il valore complementare di una variabile. Se era 0, diventa 1. In caso contrario, diventa 0.

Sintassi

```
not <operand1>
<operand1> deve essere una variabile.
```

Esempi

```
not V1
```

```
not V3
```

ISTRUZIONE OUT

L'istruzione out istruisce il dispositivo connesso all'output specificato affinché venga acceso o spento. Se il secondo operando è 0, verrà spento. Ogni altro valore imposterà l'output sulla posizione acceso.

Sintassi

```
out <operand1> <operand2>
<operand1> deve essere un output.
<operand2> può essere una variabile o un numero intero. 0 significa
off, qualsiasi altro valore significa on.
```

Esempi

```
out O2 1
```

```
out O2 V3
```

ISTRUZIONE RET

L'istruzione ret (abbreviazione di "return") termina l'esecuzione del codice per questo ciclo di 33 millisecondi. Ha lo stesso significato di un salto a un'etichetta posta proprio sull'ultima riga di codice. Non ha operandi.

Sintassi

```
ret
```

Esempi

```
ret
```

ISTRUZIONE SUB

L'istruzione sub calcola la differenza tra due valori e ne salva il risultato nella variabile specificata nel terzo parametro. Ricorda, i registri sono a 16 bit, quindi il risultato deve essere contenuto tra -32.768 e 32.767 per evitare errori. Fondamentalmente fa "operand1 - operand2" e ne salva il risultato nella variabile specificata in operand3.

SINTASSI

```
sub <operand1> <operand2> <operand3>
<operand1> può essere una variabile o un numero intero.
<operand2> può essere una variabile o un numero intero.
<operand3> deve essere una variabile.
```

ESEMPI

```
sub V1 15 V1
```

```
sub V2 V3 V0
```

ESEMPIO DI PROGRAMMA 1

Questo esempio di programma accende il dispositivo connesso a O1 per un secondo, lo spegne per un secondo e così via.

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

endprogram:
```

ESEMPIO DI PROGRAMMA 2

Questo esempio di programma accende il dispositivo connesso a O2 dopo che sono arrivate 5 ordinazioni.

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

endprogram:
```

ESEMPIO DI PROGRAMMA 3

Questo esempio di programma mantiene acceso il dispositivo

connesso a O0 per 5 secondi all'arrivo di ogni nuova ordinazione. Un buon caso d'utilizzo per questo programma consiste nel far produrre un ingrediente a un distributore all'arrivo di ogni ordinazione. Nota che all'avvio questo programma preriscalderà anche il dispositivo per 4 secondi, in modo che gli ingredienti siano consegnati quasi immediatamente dopo l'arrivo dell'ordinazione, risparmiando tempo prezioso per i tuoi clienti! Un lettore di ordinazioni standard non può farlo, vero?

```
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
```

ESEMPIO DI PROGRAMMA 4

Questo complesso esempio di programma legge da due differenti moduli di lettura ordinazioni (R0 e R1) e gestisce gli output O0, O1 e O2. L'obiettivo del programma è accendere O0 e O1 per tre secondi ogniqualvolta arriva una nuova ordinazione a R0 o R1, ma accendere O2

solo quando arriva una nuova ordinazione a R1. Un buon caso d'utilizzo per questo programma consiste nel far gestire a R0 hamburger semplici e a R1 cheeseburger, mentre O0 è connesso a un distributore di polpette crude, O1 a un distributore di panini per hamburger e O2 a un distributore di formaggio. Inoltre all'avvio preriscalda i distributori per due secondi.

```
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
```