Automachef

HANDBUCH FÜR DEN PROGRAMMIERBAREN COMPUTER

AC-32

INHALTSVERZEICHNIS

SYSTEMARCHITEKTUR

Der AC-32 ist ein hochmoderner, programmierbarer Computer, der bis zu vier weitere Geräte über die Output-Anschlüsse mit den Bezeichnungen O0 bis O3 verwalten kann. Alle Outputs sind standardmäßig auf „aus" gestellt, wenn der Computer eingeschaltet oder zurückgesetzt wird. Der Computer verfügt über acht eingebaute 16-Bit-Register mit Vorzeichen, die für jeden Zweck eingesetzt werden können und mit V0 bis V7 bezeichnet sind. In diesem Handbuch werden sie auch als „Variablen" bezeichnet. Sie können Ganzzahlwerte zwischen -32.768 und 32.767 speichern.

Manche Küchenkomponenten unterstützen das Auslesen der Anzahl von Aktionen, die sie durchgeführt haben, zum Beispiel Zutatenschranken, Zusammensetzer und Roboterarme. Die Anzahl der Aktionen, die sie seit dem Einschalten durchgeführt haben, kann aufgerufen werden, indem die Variablen I0 bis I3 ausgelesen werden. I0 liest die Anzahl der ausgeführten Aktionen für das an O0 angeschlossene Gerät aus, I1 für das an O1, und so weiter. Komponenten, die das Auslesen dieses Wertes nicht unterstützen (zum Beispiel Fließbänder), geben immer 0 aus.

Der Hauptprogrammspeicher des Computers kann bis zu 32 Zeilen Code in der Assemblersprache AC enthalten. Er verfügt über vier zusätzliche Codeseiten, die jeweils bis zu 32 Zeilen Code enthalten können. Diese können mit dem Befehl „cal" aufgerufen werden. Details dazu findest du weiter unten im Programmierhandbuch. Das gesamte Programm wird exakt 30-mal pro Sekunde ausgeführt, wodurch es ein Leichtes ist, Routinen zu programmieren, die präzises Timing erfordern. Apropos Timing: Indem du auf das besondere schreibgeschützte Register TT zugreifst, kannst du auf die aktuelle Tageszeit im 24-Stunden-Format zugreifen. Wenn es also gerade 15:45 Uhr ist, enthält das Register TT die 16-Bit-Ganzzahl 1545.

Eine Gruppe von vier besonderen, schreibgeschützten Registern wird automatisch mit Informationen aus dem eingebauten Bestellungsleser gefüllt. Jedes dieser Register enthält eine Zahl,

die größer als 0 ist, wenn mindestens eine neue Bestellung des angegebenen Typs im aktuellen Zeitabschnitt von 33 Millisekunden eingegangen ist, und enthält ansonsten 0. Die Zahl legt fest, wie viele neue Bestellungen dieses Typs im aktuellen Zeitabschnitt von 33 Millisekunden eingegangen sind. In Befehlen in der Assemblersprache AC kann sie mit R0 bis R3 referenziert werden.

Die eingebauten Bestellungsleser können auch zwischen Bestellungen unterscheiden, die von verschiedenen Quellen ausgehen, zum Beispiel von einem Drive-in-Schalter oder Gästen, die Essen zum Mitnehmen bestellen. Zu diesem Zweck kannst du die folgenden Buchstaben an die Variablen R0 bis R3 anhängen:

R für Bestellungen, die aus dem Restaurant eingehen.

T für Bestellungen, die aus dem Mitnahmebereich eingehen.

D für Bestellungen, die vom Drive-in-Schalter eingehen.

Beispiele

Der eingebaute Bestellungsleser R2 ist so eingestellt, dass er nach Cheeseburgern prüft.

Die Variable R2R enthält die Anzahl der im Restaurant bestellten Cheeseburger.

Die Variable R2T enthält die Anzahl der im Mitnahmebereich bestellten Cheeseburger.

Die Variable R2D enthält die Anzahl der am Drive-in-Schalter bestellten Cheeseburger.

Die Variable R2 enthält die Gesamtzahl der im aktuellen Zeitabschnitt von 33 Millisekunden bestellten Cheeseburger.

R2 enthält daher die Summe aus R2R + R2T + R2D.

PROGRAMMIERHANDBUCH

LABELS

Labels sind benannte Stellen im Code, mit denen der Ablauf der Ausführung über die Sprungbefehle (jump (jmp/jne/jeq/jgt/jlt)) verändert werden kann. Sie enthalten nur Buchstaben, müssen zwischen 1 und 10 Stellen haben und mit einem Semikolon abschließen.

BEISPIELE

```
loopagain:
```

```
belton:
```

```
endprogram:
```

BEFEHL „ADD"

Der Befehl add („addieren") addiert zwei Werte und speichert das Ergebnis in der Variable, die im dritten Parameter angegeben wird. Vergiss nicht, es handelt sich um 16-Bit-Register, also muss das Ergebnis zwischen -32.768 und 32.767 liegen, um Fehler zu vermeiden.

SYNTAX

```
add <operand1> <operand2> <operand3>
<operand1> kann eine Variable oder eine Ganzzahl sein.
<operand2> kann eine Variable oder eine Ganzzahl sein.
<operand3> muss eine Variable sein.
```

Beispiele

```
add V1 15 V2
```

```
add V0 V1 V0
```

BEFEHL „CAL"

Der Befehl cal (kurz für „call", also „aufrufen") kann nur auf der Haupt-Codeseite verwendet werden. Dieser Befehl führt automatisch alle Befehle auf der angegebenen Codeseite aus und fährt dann mit der Ausführung der nächsten Zeile fort, wenn dies abgeschlossen ist.

Programmierer können die Codeseiten als Prozesse betrachten, die vom Hauptprogramm aus aufgerufen werden können. Der AC-32-Computer kann keine Codeseiten von anderen Codeseiten aus aufrufen, da er nicht über einen Aufrufstapel verfügt.

Syntax

```
cal <operand1>
<operand1> sollte eine Ganzzahl zwischen 1 und 4 sein. Diese gibt die
Nummer der Codeseite an, die aufgerufen werden soll.
```

Beispiele

```
cal 2
```

BEFEHL „CMP"

Der Befehl cmp (kurz für „compare", also „vergleichen") vergleicht zwei Werte und setzt das Vergleichsregister auf entweder -1, 0 oder 1. Wenn der erste Wert kleiner ist als der zweite, wird es auf -1 gesetzt. Wenn beide gleich sind, wird es auf 0 gesetzt. Ist der erste Wert größer als der zweite, wird es auf 1 gesetzt. Das Ergebnis kann dann benutzt werden, um in Abhängigkeit vom Wert über die Befehle jne/jeq/jlt/jgt zu einem anderen Teil des Codes zu springen.

Syntax

```
cmp <operand1> <operand2>
<operand1> kann eine Variable oder eine Ganzzahl sein.
<operand2> kann eine Variable oder eine Ganzzahl sein.
```

Beispiele

```
cmp V1 30
```

```
cmp V1 V3
```

BEFEHL „DEC"

Der Befehl dec (kurz für „decrement", also „verringern") verringert die angegebene Variable um 1, lässt jedoch nicht zu, dass diese je negative Zahlen erreicht, und stoppt daher automatisch bei 0. Er ist besonders nützlich, um Timer einzurichten.

Syntax

```
dec <operand1>
<operand1> muss eine Variable sein.
```

Beispiele

```
dec V0
```

```
dec V3
```

BEFEHLE „JMP"/„JNE"/„JEQ"/„JGT"/„JLT"

All diese Befehle springen zum jeweils angegebenen Label. jne steht für „jump if not equal", also „springe, wenn nicht gleich"; jeq steht für „jump if equal", also „springe, wenn gleich"; jlt steht für „jump if less than", also „springe, wenn weniger als", und jgt steht für „jump if greater than", also „springe, wenn größer als". Sie springen zum angegebenen Label, wenn das Ergebnis des letzten mit dem cmp-Befehl ausgeführten Vergleichs dem Ergebnis im Namen des Befehls entspricht. Wenn du zum Beispiel den Befehl jne nach einem Vergleich ausführst, wird der Sprung nur ausgeführt, wenn die verglichenen Parameter nicht gleich waren. Der Befehl jgt führt den Sprung nur aus, wenn der erste Parameter im Vergleich größer war als der zweite, und so weiter. Der Befehl jmp („jump", also „springe") springt immer (und bedingungslos) zum angegebenen Label.

Syntax

```
jmp <operand1>
jne<operand1>
jeq<operand1>
jlt<operand1>
jgt<operand1>
<operand1> muss ein Label sein.
```

Beispiele

```
jne endprogram
```

```
jlt loopagain
```

BEFEHL „MOV"

Der Befehl mov (kurz für „move", also „bewegen") kopiert einen Wert in die jeweils angegebene Variable.

Syntax

```
mov <operand1> <operand2>
<operand1> kann eine Variable oder eine Ganzzahl sein.
<operand2> muss eine Variable sein.
```

Beispiele

```
mov 30 V2
```

```
mov V1 V2
```

BEFEHL „MUL"

Der Befehl mul („multiplizieren") addiert zwei Werte und speichert das Ergebnis in der Variable, die im dritten Parameter angegeben wird. Dieser Befehl ist nur für den AC-32-Computer verfügbar. Vergiss nicht, es handelt sich um 16-Bit-Register, also muss das Ergebnis zwischen -32.768 und 32.767 liegen, um Fehler zu vermeiden.

Syntax

```
mul <operand1> <operand2> <operand3>
<operand1> kann eine Variable oder eine Ganzzahl sein.
<operand2> kann eine Variable oder eine Ganzzahl sein.
<operand3> muss eine Variable sein.
```

Beispiele

```
mul V1 15 V2
```

```
mul V0 V1 V0
```

BEFEHL „NOT"

Der Befehl not („nicht") schaltet den Wert einer Variable einfach um. War er 0, wird er zu 1; andernfalls wird er zu 0.

Syntax

```
not <operand1>
<operand1> muss eine Variable sein.
```

Beispiele

```
not V1
```

```
not V3
```

Befehl „out"

Der Befehl out („output", also Ausgabe) befiehlt dem am jeweils angegebenen Output angeschlossenen Gerät, sich ein- oder auszuschalten. Ist der zweite Operand 0, wird es ausgeschaltet. Bei jedem anderen Wert wird der Output auf „ein" geschaltet.

Syntax

```
out <operand1> <operand2>
<operand1> muss ein Output sein.
<operand2> kann eine Variable oder ein Ganzzahlwert sein. 0 bedeutet
„aus", alles andere bedeutet „ein".
```

Beispiele

```
out O2 1
```

```
out O2 V3
```

BEFEHL „RET"

Der Befehl ret (für „return") schließt das Ausführen des Codes für den aktuellen Zyklus von 33 Millisekunden ab. Er hat dieselbe Bedeutung wie ein Sprung zu einem Label, das ganz am Ende der letzten Codezeile steht. Er hat keine Operanden.

Syntax

```
ret
```

Beispiele

```
ret
```

BEFEHL „SUB"

Der Befehl sub („subtrahieren") berechnet die Differenz zwischen zwei Werten und speichert das Ergebnis in der Variable, die im dritten Parameter angegeben wird. Vergiss nicht, es handelt sich um 16-Bit-Register, also muss das Ergebnis zwischen -32.768 und 32.767 liegen, um Fehler zu vermeiden. Kurz gesagt: Er führt „operand1 - operand2" aus und speichert das Ergebnis in der Variable, die mit operand3 festgelegt wird.

Syntax

```
sub <operand1> <operand2> <operand3>
<operand1> kann eine Variable oder eine Ganzzahl sein.
<operand2> kann eine Variable oder eine Ganzzahl sein.
<operand3> muss eine Variable sein.
```

Beispiele

```
sub V1 15 V1
```

```
sub V2 V3 V0
```

BEISPIELPROGRAMM NR. 1

Dieses Beispielprogramm schaltet das an O1 angeschlossene Gerät eine Sekunde ein, dann eine Sekunde lang aus und so weiter.

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

endprogram:
```

BEISPIELPROGRAMM NR. 2

Dieses Beispielprogramm schaltet das an O2 angeschlossene Gerät ein, nachdem 5 Bestellungen eingegangen sind.

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

endprogram:
```

BEISPIELPROGRAMM NR. 3

Dieses Beispielprogramm hält das an O0 angeschlossene Gerät für

jede neu eingegangene Bestellung für 5 Sekunden eingeschaltet. Ein guter Einsatz für dieses Programm ist, einen Spender für jede eingegangene Bestellung eine Zutat ausgeben zu lassen. Beachte, dass dieses Programm das Gerät nach dem Start zusätzlich 4 Sekunden lang vorwärmt, sodass Zutaten fast sofort nach Eingang der Bestellung ausgegeben werden und deine Kunden kostbare Zeit sparen! Ein einfacher Bestellungsleser kann so etwas nicht, oder?

```
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
```

BEISPIELPROGRAMM NR. 4

Dieses komplexe Beispielprogramm liest zwei verschiedene Bestellungsleser-Module aus (R0 und R1) und verwaltet die Outputs O0, O1 und O2. Das Ziel des Programms ist, O0 und O1 für drei Sekunden einzuschalten, wann immer eine neue Bestellung bei R0 oder R1 eintrifft, aber O2 nur dann einzuschalten, wenn eine neue Bestellung bei R1

eingeht. Ein guter Einsatz für dieses Programm ist, R0 Einfache Burger und R1 Cheeseburger überwachen zu lassen, während O0 an einen Spender für Rohe Pattys, O1 an einen Burgerbrötchen-Spender und O2 an einen Käse-Spender angeschlossen ist. Außerdem werden die Spender beim Start zwei Sekunden lang vorgewärmt.

```
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
```