

AC-16

プログラマブル コンピ ューターの解説

目次

システム構成	i	not 命令	vii
言語解説	iii	out 命令	vii
ラベル	iii	ret 命令	viii
add 命令	iii	sub 命令	viii
cmp 命令	iv	プログラム例#1.....	x
dec 命令	iv	プログラム例#2.....	x
jmp/jne/jeq/jgt/jlt 命令	v	プログラム例#3.....	x
mov 命令	vi	プログラム例#4.....	xi

システム構成

AC-16 は 00～03 までの出力端子を介して最大 4 種類の装置を制御できる最新鋭のプログラマブルコンピュータです。AC-16 の起動時またはリセット時にすべての出力端子はデフォルトでオフの状態に設定されます。V0～V3 までの符号付き 16 ビットの汎用レジスタが 4 つ内蔵されています。この解説書では変数と呼ばれるもので、-32,768～32,767 までの整数を格納できます。

食材ゲート、アセンブラー、ロボットアームなどの一部の装置は実行したアクションの回数を測定できます。これらの装置が起動してから実行したアクションの回数は、I0～I3 までの変数を読み取ることで取得できます。I0 は 00、I1、01 などに接続された装置が実行したアクションの回数を読み取ります。この値を測定できない装置(コンベアベルトなど)は必ず 0 を返します。

コンピュータのプログラムメモリーには AC アセンブリ言語で書かれた最大 32 行のコードを格納できます。全プログラムは毎秒 50 回、確実に実行されますので、正確なタイミングを要するルーチンのプログラミングを容易に行えます。タイミングに関してですが、特殊な読み取り専用 TT レジスタを利用すれば、24 時間形式で現在の時刻を取得できます。例えば、現在の時刻が午後 3 時 45 分なら、TT レジスタには 16 ビット整数の 1545 が格納されます。

内蔵の注文リーダーから送られてくる情報を自動的に入力する 4 セットの特種な読み取り専用レジスタがあります。各レジスタは現在時刻の 33 ミリ秒以内に指定された種類の注文が入った場合は 0 より大きい数値を格納し、それ以外の場合は 0 を格納します。この数値は現在時刻の 33 ミリ秒以内に入った指定された種類の注文の数によって決まります。数値は AC アセンブリ言語命令の R0～R3 で参照可能です。

内蔵の注文リーダーは、ドライブスルーの窓口やテイクアウト用の料理を注文する客といった様々な発注元を区別することができます。区別させるには、下記の文字をサフィックスとして R0～R3 の変数に追加してください：

Rはレストランからの注文を表します。

Tはテイクアウトの窓口からの注文を表します。

Dはドライブスルーの窓口からの注文を表します。

例

内蔵の注文リーダー R2 を設定してチーズバーガーを検出させます。

変数 R2R にはレストランで注文されたチーズバーガーの数が格納されます。

変数 R2T にはテイクアウトの窓口で注文されたチーズバーガーの数が格納されます。

変数 R2D にはドライブスルーの窓口で注文されたチーズバーガーの数が格納されます。

変数 R2 には現在時刻の 33 ミリ秒以内に注文されたチーズバーガーの総数が格納されます。

従って、R2 には $R2R + R2T + R2D$ の合計数が格納されます。

言語解説

ラベル

ラベルとはjump (jmp/jne/jeq/jgt/jlt) 命令で実行フローを変更するのに使用できるコード内の行の名前です。1~10 文字までの英字のみを格納可能で、文字列の最後にコロンを入れる必要があります。

例

```
loopagain:
```

```
belton:
```

```
endprogram:
```

ADD 命令

add 命令は 2 つの値を加算し、その結果を 3 番目のパラメーターで指定された変数に格納します。レジスタは 16-bit でするので、エラーを防ぐために結果は -32,768~32,767 の範囲でなければなりません。

構文

```
add <operand1> <operand2> <operand3>
```

<operand1> は変数または整数にできます。

<operand2> は変数または整数にできます。

<operand3> は変数でなければなりません。

例

```
add V1 I5 V2
```

```
add V0 V1 V0
```

CMP 命令

cmp 命令は 2 つの値を比較してコンペアレジスタを -1、0、1 のいずれかに設定します。最初の値が 2 番目の値よりも小さい場合は、-1 に設定されます。値が等しい場合は、0 に設定されます。最初の値が 2 番目の値よりも大きい場合は 1 に設定されます。この結果は、jne/jeq/jlt/jgt 命令を使ってコードの別の部分へ条件付きジャンプを行うのに使用できます。

構文

```
cmp <operand1> <operand2>
```

<operand1> は変数または整数にできます。

<operand2> は変数または整数にできます。

例

```
cmp V1 30
```

```
cmp V1 V3
```

DEC 命令

dec 命令は指定された変数の値を 1 ずつ減らしますが、負の値は

許容しませんので、自動的に 0 で停止します。タイマーを実装する時に役に立つ命令です。

構文

```
dec <operand1>
```

<operand1> は変数でなければなりません。

例

```
dec V0
```

```
dec V3
```

JMP/JNE/JEQ/JGT/JLT 命令

これらはすべて指定されたラベルにジャンプするための命令です。jne は「等しくない場合にジャンプ」、jeq は「等しい場合にジャンプ」、jlt は「小さい場合にジャンプ」、そして jgt は「大きい場合にジャンプ」を意味します。cmp 命令で行われる最後の比較結果が各命令の結果と一致した場合に指定されたラベルにジャンプします。例えば、jne 命令は比較したパラメーターが等しくないと判断された場合にのみ指定されたラベルにジャンプします。jgt 命令は比較で最初のパラメーターが 2 番目のパラメーターよりも大きいと判断された場合にのみ指定されたラベルにジャンプします。jmp 命令は必ず (無条件に) 指定されたラベルにジャンプします。

構文

```
jmp <operand1>
jne <operand1>
jeq <operand1>
jlt <operand1>
jgt <operand1>
<operand1> はラベルにする必要があります。
```

例

```
jne endprogram
```

```
jlt loopagain
```

MOV 命令

mov 命令は指定された変数に値をコピーします。

構文

```
mov <operand1> <operand2>
<operand1> は変数または整数にできます。
<operand2> は変数でなければなりません。
```

例

```
mov 30 V2
```



```
movV1V2
```

NOT 命令

not 命令は変数の値を切り替えます。0 の場合は 1 に、1 の場合は 0 に切り替えます。

構文

```
not <operand1>
```

<operand1> は変数でなければなりません。

例

```
notV1
```

```
notV3
```

OUT 命令

out 命令は指定された出力端子に接続された装置を オンまたはオフにします。第 2 オペランドが 0 の場合は装置をオフにします。その他の値では装置をオンにします。

構文

```
out <operand1> <operand2>
```

<operand1> は出力端子にする必要があります。

<operand2> は変数または整数にできます。0 はオフ、その他の値はオンになります。

例

```
out O2 1
```

```
out O2 V3
```

RET 命令

ret 命令 (return の略) は、この 33 ミリ秒サイクルのコードの実行を終了させます。これはコードの最後の行に配置されたラベルにジャンプするのと同義です。オペランドはありません。

構文

```
ret
```

例

```
ret
```

SUB 命令

sub 命令は 2 つの値の差を計算し、その結果を 3 番目のパラメータで指定された変数に格納します。レジスタは 16-bit ですので、エラーを防ぐために結果は -32,768~32,767 の範囲でなければなりません。基本的には、「operand1-operand2」の差を計算し、その結果を operand3 で指定された変数に格納します。

構文

```
sub <operand1> <operand2> <operand3>
```

<operand1> は変数または整数にできます。

<operand2> は変数または整数にできます。

<operand3> は変数でなければなりません。

例

```
sub V1 15 V1
```

```
sub V2 V3 V0
```

プログラム例 #1

このプログラム例では O1 に接続された装置を 1 秒間オンにし、1 秒間オフにできます。

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

endprogram:
```

プログラム例 #2

このプログラム例では 5 件の注文が入った後に O2 に接続された装置をオンにできます。

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

endprogram:
```

プログラム例 #3

このプログラム例では新しい注文が入るごとに O0 に接続された装置を

5 秒間オンにできます。新しい注文が入るごとにディスペンサーに1つの食材を供給させたいときに最適です。このプログラムは起動時に装置を4秒間ウォームアップしますので、食材は新しい注文が入った直後に供給されます。ですので、貴重な時間を節約できます。通常の注文リーダーではできない制御です。

```
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
```

プログラム例 #4

この複雑なプログラム例では、内蔵の注文リーダー2つ(R0とR1)の注文を読み取り、出力端子O0、O1、O2を制御できます。このプログラムの目的は、R0かR1のいずれかに新しい注文が入った時にO0とO1を3秒間オンにし、R1に新しい注文が入った時にO2だけをオンにすることです。O0が生のパティのディスペンサーに、O1がバーガーのバンズのディスペン

サーに、そして O2 がチーズのディスペンサーに接続されている状態で R0 にプレーンバーガーを処理させ、R1 にチーズバーガーを処理させるのに最適です。また、起動時にディスペンサーを 2 秒間ウォームアップします。

```
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
```