マイクロコンピュータとワークステーション 64－1
（１９９０．　１１．　３０）

マイコン向き誤動作自己検知法

坂巻　佳壽美
東京都立工業技術センター

安価なマイコン応用システムに向いた「安価な誤動作自己検知法」を２例紹介している。１つは、インストラクションフェッチ時のメモリアドレスが適正であるかをチェックする方法で、多くのマイコンが採用している可変語長命令体系を疑似固定語長化することにより可能としている。もう１つは、マイコンの実行に伴って発生するコントロール信号をチェックする方法で、全メモリアドレス空間に於ける適正なコントロール情報データを予め用意しておく必要がある。いずれの方法に於ても、Ｚ８０ＣＰＵでの具体的な実施例を示したが、特定のＣＰＵに依存する部分が少ないために、多くのＣＰＵに採用可能である。

SELF DETECTION MEASURES TO COUNT ERRORS IN MICROCOMPUTER SYSTEMS

Kazumi Sakamaki
Tokyo Metropolitan Industrial Technology Center

There are two inexepensive self detection methods for identifing errors in microcomputer systems. The first checks the appropriateness of the memory address during the instruction fetch cycle. It functions by converting the popularly used variable word length instruction system into a one based on the pseudo fixed word length instruction system. The second, which requires the prior preparation of appropriate status data available from all address spaces, concurrently checks the control signal generated while the microcomputer is running. In either case specific test examples using the Z80CPU are indicated. And these methods are practical for use in other CPU cases because their characteristics are less affected by a specific CPU.

１．はじめに（安価な誤動作自己検知法の必要性）

マイコンの利用拡大が広範囲に浸透するに伴い、単に機能の向上を目指すだけのマイコン応用ではなく、マイコン応用システムの信頼性向上が社会的な関心事となっている。

マイクロプロセッサ（以下ＣＰＵという）自身に信頼性向上対策を機能として持たせたものとしては、４３２（インテル）がよく知られている。また、国産ＣＰＵに於いても、Ｖ６０（ＮＥＣ）以降のＶシリーズにＦＲＭ機能が搭載されている。

しかし、これらによる信頼性向上策は、従来から良く知られているハードウェアの冗長性を用いた構成を前提にしている。そして、ＯＬＴＰ向けのフォルトトレラント・コンピュータなど、マイコン応用システムとしてはかなり上位部門で多く採用されている。

一方、『安い、小さい、高機能』というマイコンの特徴を活かした小規模な応用システムに於いては、コストに直接影響するハードウェアの冗長は受け入れ難い。そこで、安価なマイコン応用システムに向いた「安価な誤動作自己検知法」が求められることになる。

これまでのマイコン応用システムに多く採用されている安価な暴走対策としては、ウォッチドッグ・タイマがある。しかし、その動作原理上、誤動作の発生を即時に検知することはできない。また、タイムアップ時の対処に、システム・リセットによる再スタートを採用しているケースの多いことにも問題があると思われる。

今後も、更に付加価値の高いマイコン応用システムが、身近に増えることが予測される。そして、便利さが増すに伴い、信頼性への要望も高くなって行くことは必至である。

以下に紹介する二例の誤動作自己検知法は、８ビット・クラスのＣＰＵを用いたマイコン応用システムを前提にしているが、特定のＣＰＵに依存する部分は少ないため、多くのＣＰＵに於いても採用可能と思われる。

２．インストラクションフェッチ・モニタ法

各インストラクション・フェッチサイクルに於て、メモリアドレスが適正であるかをチェックする方法である。この方法を実施する前提条件として、インストラクション部の存在するメモリアドレスに一定のルール付けが必要であり、ここでは多くのマイコンに採用されている可変語長命令体系を疑似固定語長化することにした。

2-1　命令体系の疑似固定語長化

マイコンの暴走は、原因が何であれ、結果として命令を構成しているインストラクション部とオペランド部とを取り違えてしまうことから発生することが予想される。そこで、取り違えがおきたかどうかをチェックすることができれば、暴走の発生を検知することが可能となる。

ところが、今日のＣＰＵのほとんどが可変語長命令体系を採用しているために、それらの区別を簡単に行うことは出来ない。

そこで、疑似固定語長命令体系を導入した。疑似固定語長命令体系とは、全ての命令が同じバイト数で構成されているという意味である。つまり、現有の可変語長命令体系で、最長な命令語長のバイト数に合わせて疑似的に固定語長化するのである。

その際、従来の個々の命令の語長だけでは不足する短いバイト長の命令には、パッド命令を追加し、最大バイト長の命令に語長を合わせた疑似固定語長とする必要がある（図１参照）。そのため

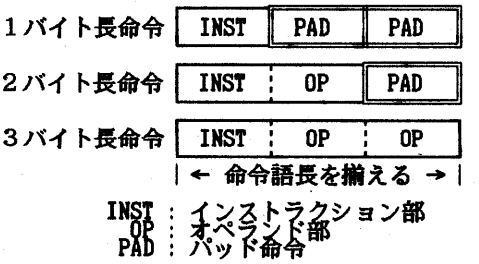

図１．命令体系の疑似固定語長化
（３バイト長の場合）

```
if（ インストラクションフェッチ・サイクル  ==  TRUE  ）{
   if（!（ アドレス%固定語長==0  |  データ==パッド命令  ））  誤動作発生;
}
正常動作;
```

図２．インストラクションフェッチ・モニタ法による誤動作検知のＣ言語的表記

のパッド命令としては、実質的に無効な命令（ＮＯＰ命令相当）で、かつ実行時間の少ないものを充てるのが望ましい。

以上の結果、必ず一定バイト数毎に、命令の先頭（インストラクション部）が存在するようになる。したがって、このルールに着目したチェックを行えば、プログラムの進行状況が正常かどうかをチェックすることが可能となる。この方式による誤動作検知法を、Ｃ言語的に表記すると図２のようになる。

しかし、この疑似固定語長命令体系を採用すると、そのために挿入したパッド命令の分だけ、メモリサイズと実行時間が冗長となる。

2-2　Ｚ８０ＣＰＵによる実施例

Ｚ８０ＣＰＵを例に、具体的に説明すると、命令の最長語長が４バイトなので、疑似４バイト固定語長命令体系として扱うことになる。本来が１～３バイト長の命令に対しては、不足するバイト数分のパッド命令を、単にバイト数を合わせるためだけの目的で追加する。パッド命令としては、ＮＯＰ命令が適当であろう。

その結果、命令のインストラクション部は４バイト毎に存在することになり（リスト１(b)参照）、それ以外の場所にあるインストラクション部は、パッド命令であるＮＯＰ命令だけに限定されることになる。

一方、疑似４バイト固定語長化されたプログラムの実行を、図２に従ってチェックするための誤動作検知回路を用意する必要がある。そのためには、Ｚ８０ＣＰＵのコントロール信号のうち、ＣＰＵ動作がインストラクション・フェッチ・サイクルにあることを外部に示すＭ１信号を利用する。

具体的には、以下のことをチェックする回路とすればよい。

①Ｍ１信号がアクティブ（つまりインストラクション・フェッチ・サイクル）の時、アドレスは４の倍数でなければならない。
②もし、そうでなければ、その時の命令はＮＯＰである。
③あるいは、インストラクション部が２バイト構成となっている命令（Ｚ８０ＣＰＵ固有）の、２バイト目である。

```
0100  E6 0F        p_hex:  and     0fh
0102  C6 30                add     a,30h
0104  FE 3A                cp      3ah
0106  38 02                jr      c,print
                    ;
0108  C6 07                add     a,7
                    ;
010A  4F           print:  ld      c,a
```

(a)もとのプログラム

```
0100  E6 0F        p_hex:  and     0fh
0102  00                   nop
0103  00                   nop
0104  C6 30                add     a,30h
0106  00                   nop
0107  00                   nop
0108  FE 3A                cp      3ah
010A  00                   nop
010B  00                   nop
010C  38 06                jr      c,print
010E  00                   nop
010F  00                   nop
                    ;
0110  C6 07                add     a,7
0112  00                   nop
0113  00                   nop
                    ;
0116  4F           print:  ld      c,a
```

(b)ＮＯＰを挿入して４バイト長に揃える

リスト１．疑似４バイト固定語長プログラム例

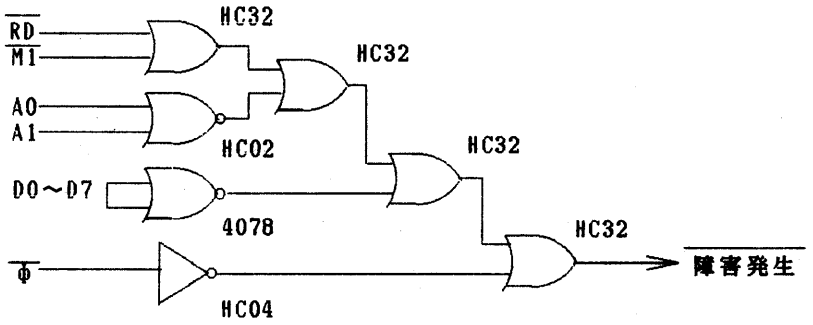

図３．Ｚ８０ＣＰＵによる誤動作検知回路例（インストラクションフェッチ・モニタ法）

①の状態をチェックするためには、アドレス線の下位２ビット（A0とA1）が、ともに“0”となっているかをチェックすれば良い。また、②については、データ・バス上のデータをチェックし、ＮＯＰ命令のインストラクションコード“00”であるかを確認すれば良い。

③については、よりシンプルな回路例として示すために本稿では省略した。

以上から、前記した２つの条件（①と②）のどちらにも該当しない場合には、条件を満たしていないことになり、何等かの誤動作が発生したものと判断し、割り込み等によりＣＰＵへ知らせるようにする。

具体的な誤動作検知回路例を、図３に示す。ただ、実際のＣＰＵがＴ３サイクルの立ち上がりエッジで、データバス上のデータのサンプルを行なっているのに対し、本回路の誤動作検知のタイミングがＴ２サイクルの前半の半サイクルに、しかもレベル信号として行なっている。そのために、本回路を使用するに当たっては、出来るだけアクセスタイムの速いＲＡＭやＲＯＭを使用することが必要である。

この方法によると、４バイトの疑似固定語長とするために挿入したＮＯＰ命令の分だけ、メモリを余分に使用するとともに、実行時間も長くなってしまうところに難がある。

経験的に言って、インテル系の８ビットＣＰＵでは、１ステップの平均は２バイトである。さらにＺ８０ＣＰＵでは、相対アドレッシング・モードの命令（２バイト）が加わるために、なおさら２バイト平均に近付くことになる。それらが、全て４バイト均一となるので、単純に計算しても、本方式を採用すると２倍のメモリと２倍の実行時間が必要となることがわかる。

３．コントロールバス・モニタ法

前述した命令の疑似固定語長化では、プログラムの冗長に伴う実行スピードの低下を避けられない。また、ライブラリ・プログラムなどのように、始めから機械語で提供されているプログラムについては、疑似固定語長化する作業が困難である。

そこで、プログラムそのものには手を触れずに、そのプログラムやデータの存在する全アドレス空間に対して、正常動作時に発生するであろうコントロール信号を予め記録しておき、実行時にそれらをチェックしようとするのが、これから紹介する方法である。

3-1　制御信号の一致検出

プログラムが決まれば、それを構成する各命令毎に、インストラクション・フェッチやオペランドの取り込み、処理の実行といった全てのサイク

動作の種類	M1	RD	WR	MREQ	IORQ
INSTRUCTION FETCH	L	L	H	L	H
READ MEMORY	H	L	H	L	H
WRITE MEMORY	H	H	L	L	H
READ/WRITE MEMORY	H	L	L	L	H
READ I/O	H	L	H	H	L
WRITE I/O	H	H	L	H	L
READ/WRITE I/O	H	L	L	H	L

表1．コントロール情報データ

ルに従ってアクティブとなるコントロール信号の種類も決定する（表1参照）。つまり、プログラムの各バイト毎にコントロール信号の状態（コントロール情報データと呼ぶ）が一意的に決まることになる（リスト2）。

また、RAMに割り当てたメモリアドレスについても、コントロール情報データが用意できる。この場合には、読み書きの両方とも可能にしておく必要がある（表1の READ/WRITE MEMORY 参照）。つまり、コントロール情報データとは、各々のアドレスに於いて、どのコントロール信号がアクティブとなることが許されているのかを登録したものである。したがって、使用していないアドレス空間には、全てのコントロール信号の使用を禁止しておくことによって、誤動作等による違法なアクセスを検知する事が出来る。

このコントロール情報データを、プログラムと同一アドレスに配置した別のROMに記録しておき、プログラムの実行に伴って実際に発生するコントロール情報との一致をチェックすることによって、マイコンの誤動作検知が可能となる。

3-2 Z80CPUによる実施例

Z80CPUを例に採ると、リスト2に示すように、本来の命令コードの各バイト毎に、コントロール情報の状態をコード化したデータをも用意することになる。ここでは、コントロール信号の内、M1（インストラクションフェッチ）、RD（リード）、WR（ライト）の3つのタイミング信号を使用し、MREQ（メモリ・リクエスト）によりメモリ・アドレスへのアクセスのみをチェックするような回路構成とした（図4）。

コントロール情報データは、MSBから順にM1、RD、WRを割り当て、信号がアクティブになる場合には、該当するビットを“1”にしてある。これらのデータを得るには、専用のアセンブラを用意することによって、比較的容易に解決することが出来る。

なを、図4に示した回路は、あくまでも考え方

```
メモリ・アドレス
     機械語プログラム
       コントロール情報データ
↓    ↓  ↓
0200  3E  C0          LD      A,80H
0201  80  40
0202  32  C0          LD      (CONV),A
0203  02  40
0204  80  40
0205  3A  C0    LOOP: LD      A,(CONV)
0206  02  40
0207  80  40
0208  E6  C0          AND     04H
0209  04  40
020A  20  C0          JR      NZ,LOOP
020B  F9  40
020C  3A  C0          LD      A,(ADC)
020D  01  40
020E  80  40
020F  3A  C0          LD      (BUF),A
0210  00  40
0211  10  40
        ..............................

;***    RAM アドレス     ***
1000  xx  60    BUFF:   DS      1
        ..............................

;***  メモリマップドI／O  ***
8000  xx  40    PA:     DS      1
8001  xx  40    ADC:    DS      1
8002  xx  60    CONV:   DS      1
8003  xx  20    CWR:    DS      1
```

リスト2．コントロール情報を含むプログラム例

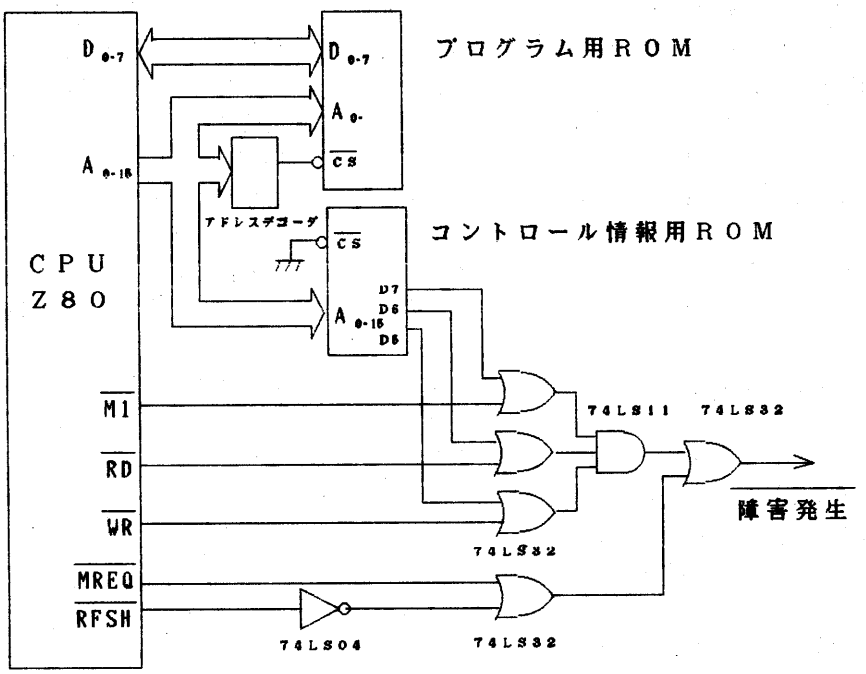

図４．Ｚ８０ＣＰＵによる誤動作検知回路例（コントロールバス・モニタ法）

を示したものである。リフレッシュ動作時に発生するＭＲＥＱ信号については対処したつもりであるが、ＤＭＡや割り込み等に関した細かな検討は充分に行なっていない。

また、インテル系ＣＰＵでは、メモリアドレスの他にＩ／Ｏアドレスがあるが、本方式を効果的に適用するためには、Ｉ／Ｏアドレスは使用せず、メモリマップドＩ／Ｏとすべきである。

４．おわりに

安価なマイコン応用システム向きの「安価な誤動作検知法」として、８ビット・クラスのＣＰＵを前提とした２例を紹介した。いずれの方法も、ハードウェアの追加を極力少なくすることに重点を置き、プログラムサイズの増大や事前の準備作業で対処できる方針を採った。

特に小規模なシステムでは、ハードウェアの増加は直接にコストアップにつながるため、可能な限りソフトウェア的手法に重点を置いた信頼性向上策が有効である。

また、これまでのマイコン応用システムの高信頼化としては、ノイズ対策が中心的なテーマとなりがちであった。しかし、処理内容と有機的な結び付きを持たせながら本稿の誤動作検知法を採用することにより、ノイズ対策部品を一部削減させることが可能であり、コストの低減効果が期待できるであろう。

今後は、安価なマイコンシステムと言えども、処理の高速化、高機能化への要望は高まる一方であり、また多ビット、高機能ＣＰＵの価格低下ともあいまって、キャッシュやパイプライン機能の搭載されたＣＰＵを採用することになるのも時間の問題である。そのため、それらＣＰＵに対応し、かつソフトウェアの冗長性に重点を於いた「安価な誤動作自己検知法」の開発が、早急に求められているところである。