Trans. KIEE, Vol 38. No. 1 JAN 1989

論 文
38~1~9

同時自己診斷이 可能한 마이크로프로세서의 하드웨어 構成에 관한 硏究

A Design Approach to a Concurrent Self-Diagnosable Microprocessor

河貴在*·申明澈**
(Gyung-Jae Ha · Myong-Chul Shin)

요 약

本 論文은 同時自己診斷이 可能한 마이크로프로세서의 하드웨어 構成方法을 提示하였다. 프로세서 自身의 問題프로그램의 遂行時 實行命令에 參與하지 않는 內部 機能유니트를 動的으로 分類하여 檢査를 해나가는 同時故障診斷의 基本原理를 旣存의 汎用 마이크로프로세서에 適用하되, 診斷專用 하드웨어 및 I/O핀 數를 可能한 限 最少化하고 원래의 프로세서 設計思想을 크게 變化시키지 않도록 하였다.

8080A 마이크로프로세서의 適用 例를 통하여 그 性能을 評價한 結果, 本 論文에서 提示한 故障診斷方法으로써 프로세서自身의 問題프로그램 遂行과 並行하여 檢査對象 유니트 全部가 빠른 時間內에 完全診斷 可能함을 알 수 있었으며, 따라서 提案된 同時自己診斷型 마이크로프로세서의 하드웨어 構成方法이 有用하다는 것을 立證할 수 있었다.

Abstract-In this paper, a design approch is presented for a concurrent self-diagnosable microprocessor. An efficient diagnostic procedure to dynamically test the processor functional units decomposed by each instruction, concurrently with a program under execution is suggested. The basic goals of the proposed design approach are the low hardware overhead, no increase in the pin count and the minimal change of the initial design concepts on conventional microprocessors.

The results of the performance evaluation of the suggested self-diagnosing hardware for an 8080 type microprocessor show that the suggested diagnosis scheme would be efficiently applicable, since diagnosing the processor functional units can be completed in a reasonably short time with the execution of an arbitrarily chosen sample program.

1. 序 論

마이크로프로세서를 비롯한 VLSI素子들로 構成되는 디지탈 시스템이 온라인 工程制御,電子交換시스템 및 宇宙關聯 컴퓨터制御등과 같은 實時間處理를 위한 應用分野에 많이 使用되고 있는 바, 이들 VLSI素子들은 그 信賴度, 利用度面에서의 要求가 必然的이다.[1],[2]

요즈음 마이크로프로세서의 效率的인 故障檢査方法이 많이 硏究되고 있으며 이것들은 주로, 별도의 檢査裝置를 使用한 檢査 (off-line testing)[3]를 前提로 하거나 혹은 마이크로프로세서가 시스템上의 構成要素이면서 動作遂行을 계속하다가 故障檢査時에는 遂行을 중단하는 경우(on-line non-concurrent

*正 會 員 : 慶南大 工大 電子計算學科 助敎授
**正 會 員 : 成均館大 工大 電氣工學科 敎授, 工博
接受日字 : 1988年 8月 26日
一次修正 : 1988年 11月 22日

testing)[4] 가 大部分이다.

그런데, 이와같은 檢査方式은 檢査패턴의生成을 위한 外部 tester가 必要하고, 故障診斷時間이 많이 걸리기 때문에 實時間應用에는 適合하지 못하다. 따라서, 요즈음에는 內藏檢査裝置(BIT; Built-In-Test)의 附加에 의한 檢査容易化設計(easily testable design)[5]-[7]라든가, 에러檢出, 訂正코드의 原理[8]를 適用한 自己체크回路를 통한 이른바 自己체크型 (self-checking) 프로세서를 實現하고자 하는 硏究[9]-[12]가 많이 행하여 지고 있다.

이러한 硏究의 窮極的인 目的은 프로세서가 自身의 作業을 계속 遂行하면서 故障診斷도 並行하는 이른바 同時自己診斷(on-line concurrent diagnosis)이 可能한 컴퓨터를 實現하는 것이다.

그러나, 그동안 硏究되어 온 故障診斷을 위한 故障檢出技法들은, 現在 널리 使用되고 있는 汎用 마이크로프로세서에 適用하여 同時自己診斷이可能하도록 하는 데는 많은 問題點이 있다. 즉, 檢査를 위한 外部 I/O 핀의 數가 增加하거나 故障을 체크하기 위한 또 다른 專用 VLSI프로세서를 必要로 할 뿐만 아니라[13], 旣存의 마이크로프로세서 設計思想의 전적인 變化를 要하여 初期設計費用이 많이 든다는 點 等이다. 특히, 이 汎用 마이크로프로세서는 마이크로프로그램이 可能하지 않기 때문에, 自己체크型 프로세서의 實現을 위해 사용한 存旣의 故障檢出技法을 適用하기 어렵고 더구나,이 경우는 制限된 故障모델을 使用하기 때문에 實際의 故障環境과 一致하지 않을 수도 있다.

따라서, 본 論文에서는 이러한 問題點들을 改善하면서 프로세서가 一般的인 問題프로그램의 遂行時, 內藏檢査裝置로써 故障檢査까지 並行하는 同時自己診斷의 한 方法을 8비트系 汎用 마이크로프로세서를 對象으로 提示하고자 한다.

2. 故障診斷의 基本原理

基本的인 故障診斷原理는 다음과 같다.

우선, 마이크로프로세서의 內部回路를 機能에 따라 分割하여, 그 個個의 機能유니트들이 現在 遂行中인 하나의 명령(ie)에 대하여 直接實行에 參與하는 것(CFU$_{ie}$: Computational Functional Units)과 參與하지 않는 것(NCFU$_{ie}$: Non-Computational Functional Units)으로 分類해 낸다. 이들 중 CFU$_{ie}$에 속하는 機能유니트에 대하여, 檢査패턴生成器에서 生成되는 檢査패턴을 利用하여 該當命令이 實行되는 동안 機能檢査를 並行해 나간다. 그러나,이 機能유니트가 다음에 實行될 命令에 대하여 實行해 參與한다고 할 경우에는, 다음 명령이 페취될 때까지만 檢査되고 通常의 프로그램 遂行環境으로 되돌아 간다. 이때, 檢査對象유니트(UUT)의 機能檢査가 該當命令의 實行中에 完了되지 않는다면 그때까지의 狀況을 記憶시켜둔 後, 다음에 그 유니트의 檢査機會가 돌아오면 그 狀況 以後부터 檢査를 再開한다. 이렇게 하여 어느 程度 긴 遂行時間을 갖는 프로그램이 프로세서내의 모든 檢査對象 機能유니트들을 檢査完了하면 다시 처음부터 檢査를 再開한다.

2.1 許容檢査時間 및 檢査時間, 檢査命令 集合

命令 ie가 페취된 後, 集合 NCFU$_{ie}$中 하나의 機能유니트를 UUT로 選擇하는 데, 이 選擇된 UUT는 다음 命令이 페취될 때까지만 檢査가 계속될 수 있다. 이 사이의 時間을 許容檢査時間(TEST$_{ie}$)이라 定義한다.

하나의 檢査벡터를 利用하여 UUT의 故障有無를 判定하는 데 걸리는 時間을 檢査時間(TVT)으로定義하며, 이 時間동안은 하나의 檢査벡터를 UUT에 印加하여 그 應答出力을 觀測하고 또한 自體的으로 生成되는 期待出力벡터와 比較하는 過程 全部를 包含한다.

따라서, 檢査命令의 集合 I는 다음 條件을 滿足하는 檢査命令 ie의 集合으로 定義한다.

$$I = \{ie \mid TEST_{ie} \geqq TVT\}$$

2.2 部分檢査, 檢査完了 및 未檢査유니트

故障診斷 도중의 어떤 時點에서, UUT의 한 機能에 대하여 그 機能檢査에 필요한 檢査벡터 全部에 의해 檢査完了된 狀態는 아니면서 적어도 하나 以上의 檢査벡터로써 部分的으로 檢査를 거쳤던 機能유니트를 部分檢査유니트라 하며 이들의集合을 PTFU라 하기로 한다. 같은 方法으로, 檢査 完了된 경우 및 檢査되지 않은 경우 各各에 대하여 TFU,NTFU로 定義하기로 한다.

2.3 機能유니트 및 그 機能[14]

汎用 마이크로프로세서의 代表的인 機能유니트들로서는 ALU, Register File, Flag Register, SP등을 들 수가 있으며, 一般的으로 fu(i)로 表記하고, 이들 유니트의 全體集合을 H, 또한 總數를 m으로 表記하기로 한다. 한 例로서, 機能유니트 ALU를

들어보자.

이 유니트는 fu(1)으로 나타낼 수 있으며,一般的으로 ALU가 遂行하는 機能들로는 AND(j=1), OR(j=2), EXOR(j=3), COMP lement (j=4), SHIFT(j=5), ADD(j=6), INC/DEC(=7), DAA(j=8)등이 있다.

以後, 기능유니트 fu(i)의 遂行機能들의 集合을 FFU(i)로 表記하기로 한다면 ALU의 경우는 다음과 같이 나타낼 수가 있다.

$$FFU(1)=\{fu(1,j)\mid j=1,\cdots\cdots,8\}$$

2.4 機能유니트의 檢査를 위한 檢査벡터

本 論文에서는 機能유니트의 故障을 機能檢査(functional testing) 方式[15] 을 통하여 檢出하도록 하였으며 특히, 이들 檢査벡터들은 미리 구하여 프로세서內部의 專用메모리에 記憶시킨다. 따라서,可能한 限 專用메모리의 容量을 最小化하기 위해서는, 各 機能유니트의 機能 各各에 대한 檢査벡터를 最小化해야 할 뿐만 아니라 可能하면 最初의 檢査 벡터가 쉬프트레지스터로써 循環자리이동 하면서 이들 一連의 檢査벡터를 動的으로 生成하도록 해야 한다.

그런데, 프로세서內의 各 機能유니트가 갖는 機能들의 檢査를 위한 檢査벡터를 구하는 方法은 紙面制約上 省略하기로 한다.

단, 이들 各 機能유니트의 機能을 檢査하기 위한 最初檢査벡터의 生成時 에는 最初檢査벡터 및 그것이 자리이동해야 할 回數에 대한 情報를 專用 ROM으로부터 읽어 쉬프트레지스터를 통하여 모든 檢査벡터를 生成시킨다.

특히, 後述하는 바와 같이 部分檢査된 유니트가 다음 機會에 다시 檢査가 再開되기 위해서는 그때까지의 狀況을 貯藏해 두어야 하므로, 이를 위하여 檢査에 다시 繼續 利用될 最初檢査벡터 및 앞으로 더 자리이동해야할 回數를 專用 RAM에 保管시킨다.

그런데, 各 機能유니트의 機能檢査를 위해서는該當 入力檢査벡터(이때는 그 機能動作의 오퍼란드가 된다)에 대하여 診斷專用 制御裝置가 獨立的으로機能을 遂行시키도록 한다.

3. 故障診斷 專用 하드웨어

故障診斷 專用 하드웨어는 旣存의 汎用 프로세서칩의 基本 設計思想을 크게 變化시키지 않고 또한 하드웨어 overhead도 可能한 限 最少化하도록 構成한다. 특히, 프로세서 自體가 갖는 원래의 遂行機能이 診斷하드웨어의 附加에 의해 전혀 影響을 미치지 않도록 하였다.

3.1 檢査專用 하드웨어

故障診斷의 制御를 위한 制御裝置(TCU)를 除外한 檢査專用 하드웨어部는 다음과 같은 機能을 갖는다. 단, 各 하드웨어部는 機能유니트의 檢査벡터와 관련하여 構成되어 져야 하나, 여기서는 그 構成結果만 記述한다.

(1) 一時貯藏 레지스터(TTSR)

UUT가 데이터貯藏 機能을 갖는 유니트일 경우, 檢査開始 以前에 그 UUT의 內容을 一時 待避시키고 檢査가 끝난 後, 원래의 장소에 되돌려 주어 다음의 命令實行에 그대로 使用할 수 있도록 하는 레지스터이다.

(2) 檢査專用 버스

UUT의 內容을 TTSR로의 待避 및 再貯藏을 위한 데이터 經路가 되며 또한 檢査벡터들의 移動經路가 된다.

(3) 쉬프트레지스터

各 機能유니트의 機能檢査에 必要한 最初의 入力檢査벡터 및 期待出力벡터를 받아서 그 機能의 檢査에 必要한 나머지 檢査벡터들을 生成하는 레지스터(ITVSR, ETVSR)이다.

(4) 쉬프트카운터 레지스터

最初의 檢査벡터로써 該當 UUT가 檢査完了時까지 IVTSR 및 ETVSR의 內容이 자리이동해야 할 回數를 나타내는 레지스터(SCTR)이다.

(5) 應答出力 및 期待出力用 버퍼

入力檢査벡터를 UUT에 印加하여 얻은 應答出力벡터와 프로세서내부에서 生成된 期待出力벡터를 比較하기 前에 잠시 이들 벡터를 貯藏하는 곳(OTVBF, ETVBF)이다.

(6) ROM

프로세서의 機能유니트가 갖는 機能들을 檢査하기 위해 필요한 最初檢査벡터와, 이 벡터가 ITVSR 및 ETVSR에 의해 자리이동해야 할 回數情報를 貯藏하는 메모리(ITVROM, ETV & SCROM)이다.

(7) RAM

部分檢査된 機能에 대하여 檢査에 使用되지 않은 入力檢査벡터와 期待出力벡터에 대한 最初벡터 및 앞으로의 자리이동 回數가 잠시 貯藏되는 메모리(ITVRAM, ETV & SCRAM)이다.

(8) 비교기 및 故障指示部

비교기는 OTVBF와 ETVBF 내의 두 데이터에 대하여 各 비트별로 EXOR의 論理를 취할 수 있도록 構成하면 되는데 이 비교기의 出入이 0이면 正常이고 1이면 故障이 發生한 경우가 되도록한다.

특히, 故障이 發生한 경우에는 故障指示部를 통해 故障이 났음을 시스템모니터에 알려야 한다. 本論文에서는 外部入出力 핀數가 增加되지 않도록 하기 위해, 旣存의 入出力핀 中 하나를 故障表示情報를 出力하는 핀으로 代替하여 使用하는 것으로 한다.

3.2 故障診斷專用 制御裝置

(1) UUT의 選定

現在 遂行中인 命令 ie에 대하여 프로세서내의 機能유니트가 CFU_{ie} 혹은 $NCFU_{ie}$에 속하는 지를 나타내고, 또한 그것들이 故障診斷中 TFU_{ie}혹은$NTFU_{ie}$에 속하는지 아니면 $PTFU_{ie}$에 속하는지를 나타내기 위해 레지스터 P, Q, R을 使用한다.

이들 세 레지스터를 使用하여 각 機能유니트와 그 機能들에 대한 檢査狀態를 나타내기 위해 그림1과 같이 構成하되 P와 Q, P와 R의 各 셀에 대하여 論理 AND를 취한다. 따라서, 이와 같이 構成하면, 한 機能유니트의 機能에 대한 檢査狀態를 判斷하는 原理가 表1과 같이 된다.

표 1 機能의 檢査狀態 情報

Table 1 Testing status information for the function.

Sab	Tab	檢 査 狀 態
0	0	機能 ffu(a, b)는 檢査對象이 아님
0	1	機能 ffu(a, b)는 檢査對象이 되며 이는 未檢査된 경우이다
1	0	이 狀態는 發生하지 않는다.
1	1	機能 ffu(a, b)는 檢査對象이 되며 이는 部分檢査된 경우이다.

(2) 故障診斷알고리즘의 遂行

故障診斷이 始作되는 t=0인 時點부터 考察하기로 한다. 이때는 어떠한 機能도 檢査되지 않았기 때문에 Q레지스터의 內容은 各 비트 모두 0이며, P레지스터의 各 비트는 모두 1이다.

그러면, 檢査命令集合 I에 속하는 모든 命令에 대하여 TCU는 다음과 같은 種類의 制御動作으로써 診斷制御알고리즘을 遂行하게 한다.

1) 檢査狀態의 체크

a. 그림1의 出力을 체크하여, 表1에서와 같이 Sab=Tab=1이면 機能 ffu(a, b)는 部分檢査된 것이다. 따라서, 이 경우에는 이 機能이 檢査對象이 된다. 즉,

IF $PTFU_{ie} \neq 0$ ($PTFU_{ie} \subseteq NCFU_{ie}$) Then Choose a UUT from the set $PTFU_{ie}$

b. 만일 $S_{ab}=0$, $T_{ab}=1$이고, $NCFU_{ie}$에 속하는 모든 機能유니트의 機能에 대하여 $S_{ij}=T_{ij}=1$인 것이 하나以上 있다면, 未檢査機能이 하나以上 있다는 것을 意味하며 이때는 TCU가 未檢査된 한 機能을 任意로 選擇한다. 즉,

IF $NTFU_{ie} \neq 0$ ($NTFU_{ie} \subseteq NCFU_{ie}$ and $PTFU_{ie}=0$) Then Choose a UUT from the set $NTFU_{ie}$

c. 모든 機能 및 機能유니트에 대하여 $S_{ij}=T_{ij}=0$이라면, 다음 中 하나의 경우에 該當된다.

① 모든 機能유니트가 한번은 檢査完了되었거나, 혹은

② 該當 實行命令 ie에 대하여, $NCFU_{ie}$에 속하는 機能유니트들 만이 모두 한번은 檢査完了되었다.

이를 判斷하기 위하여 TCU는 R레지스터의 內容을 調査해야 한다. 만일, R레지스터의 內容이 모두 0이 아니라면 프로세서 內의 모든 機能유니트들이 檢査完了된 것은 아니다. 따라서, 이 경우는 TCU가 任意로 하나의 機能유니트를 UUT로 選擇

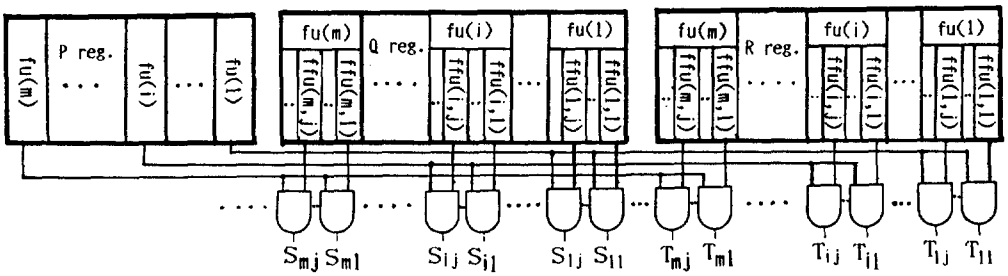

그림 1 檢査狀態의 체크를 위한 하드웨어

Fig. 1 Hardware required to check the testing status for functional units.

한다. (이때는 該當命令이 終了되어도 檢査狀態에 대한 情報는 變化하지 않는다.)그러나, R레지스터의 內容이 모두 0이라면 프로세서內部의 모든 機能유니트가 한번은 檢査完了된 것을 意味하며,따라서 이 경우는 TCU가 檢査狀態情報를 모두 初期狀態로 한 後 檢査를 再開한다. 즉,

IF TFU≠H (And $PTFU_{ie}$=0, $NTFU_{ie}$=0)Then
Choose a UUT from the set TFU_{ie}
Else if TFU=H Then
Stop test : Reset NTFU=H : Choose a UUT from the set $NTFU_{ie}$, and restart the diagnostic procedure

2) 리세트

프로세서內의 모든 機能들이 檢査完了되면, Q와 R레지스터의 檢査狀態情報를 리세트시킨다. 이러한 狀態를 체크하기 위해서는 R레지스터內의 모든 셀에 대하여 OR論理를 取하면 되는데, 이때 그 結果가 0이면 모든 機能이 檢査完了된 것을 意味하게 된다. 따라서, 다음의 動作을 통해 檢査狀態情報를 리세트시킨다.

Q[ffu(i, j)]←0;R[ffu(i, j)]←1 for all i, j

3) 檢査벡터의 生成

上記 選擇過程을 통해 檢査對象機能으로서, ffu(a, b)를 選擇하였다고 하자. 이때, ffu(a, b)가 檢査되지 않은 것이라면 最初의 入力檢査및 該當 期待出力벡터 그리고 자리이동回數를 ROM 內의 指定場所로부터 얻는다. 그러나, 만일 ffu(a, b)가 部分檢査된 機能이라면 上記情報를 RAM 內의 指定場所로부터 얻도록 한다. 따라서, 다음과 같은 데이터伝送을 同時에 遂行하도록 한다.

ITVSR←ITV & SC Y[ffu(a, b)];ETVSR, SCTR←ETV & SC Y[ffu(a, b)]

단, 여기서 Y는 檢査狀態에 따라 ROM이 된다.

4) 檢査前 데이터 待避

UUT가 데이터貯藏機能을 갖는 유니트라면 檢査以前에 TTSR에 待避시켜야 한다. 즉,

TTSR←[fu(a)]

여기서, TCU는 3)과 4)의 制御를 同時에 遂行한다.

5)檢査벡터의 印加

UUT의 한 機能을 檢査하기 위해 ITVSR의 內容을 UUT에 印加해야 하며, 이때 出力된 應答出力벡터는 OTVBF에 貯藏한다.

6) 데이터伝送

ETVSR의 內容을 ETVBF에 伝送한다.

여기서, TCU는 5)와 6)의 制御를 同時에 遂行한다.

7) 比較

OTVBF의 內容과 ETVBF의 內容을 比較하여 該當 機能유니트의 故障有無를 判斷한다.

8) 再貯藏

4)의 制御와 關聯하여 그 UUT의 機能檢査가 끝난 後, 혹은 實行中인 命令이 끝난 後는 다시 TTSR의 內容을 가져와 다음 命令의 遂行에 支障이 없도록 한다. 즉,

[fu(a)]←TTSR

9) SCTR의 체크

SCTR의 內容을 체크하여 0이 아니면 繼續 자리이동 동작을 遂行하나 0이면 下記의 14)의 制御를 遂行한다.

여기서, TCU는 7), 8), 9)의 制御를 同時에 遂行한다.

10) 자리이동

새로운 檢査벡터의 生成을 위해 ITVSR, ETVSR의 內容을 循環자리 이동 시킨다.

11) DEC 動作

SCTR의 內容을 1씩 減少시키며 만일 t<$TEST_{ie}$이면, 檢査를 繼續遂行하고 그렇지 않으면 下記의 12), 13)의 제어를 통해 그때까지의 狀況을 保管하고 檢査狀態情報를 更新시킨다.

여기서, TCU는 10), 11)을 同時에 制御할 수 있다.

12) 檢査狀況의 貯藏

ITVSR, ETVSR, SCTR의 內容을 各各 ITVRAM, ETV & SCRAM의 指定場所에 貯藏한다. 즉 TCU는 다음의 데이터伝送을 同時에 行한다.

ITVRAM[ffu(a, b)]←ITVSR;ETV & SC RAM[ffu (a, b)]←ETVSR+SCTR

13) 檢査狀態情報의 更新-①

만일, 유니트 fu(a)가 機能 ffu(a, b)에 대하여 部分檢査되는 것이라면 이 狀態를 更新하기 위해 다음의 동작을 遂行한다.

Q[ffu(a, b)]←1;R[ffu(a, b)]←1

14) 檢査狀態情報의 更新-②

上記와 같이 檢査完了되는 것이라면 다음과 같이 狀態를 更新한다.

Q[ffu(a, b)]←0 ;R[ffu(a, b)]←0

여기서, TCU의 制御動作 12), 13) 혹은 12),14)는 同時에 遂行된다

15) 故障指示

故障指示를 위한 하드웨어部에 故障信號를 보낸다.

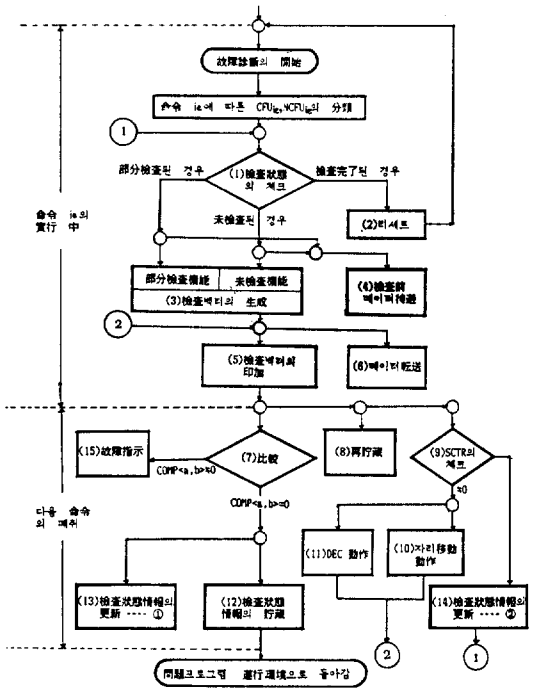

그림 2 故障診斷앨고리즘의 遂行 흐름도

Fig. 2 Flowchart for the diagnosis algorithm processing.

그림 2는 이제까지 記述한 故障診斷 앨고리즘의 遂行過程을 흐름도로 나타낸 것인데, 그림에서 같은 레벨에 있는 制御動作은 同時에 遂行한다는 것을 意味한다.

3.3 故障診斷專用 하드웨어量

그림 3은 이제까지 提示한 專用 하드웨어를 8080A 마이크로프로세서에 附加했을 경우의 하드웨어 블럭圖이다.

一般的으로 VLSI칩의 檢査容易化設計에 있어서 考慮해야 할 事項들은 여러가지가 있지만, 그 中에서도 특히 세가지 點에서의 考慮가 重要視되고 있다.[16]

즉, 旣存의 VLSI칩에 比해 增加되는 하드웨어量(hardware overhead) 및 附加되는 I/O핀의 數는 어느 程度인가 하는 것이고, 또 한가지는 診斷機能의 附加로 因한 性能低下는 어느 程度인가 하는 것이다.

따라서, 本 論文에서 提案한 同時自己診斷의 實現을 위한 하드웨어 構成方法 역시 이러한 點들을 考慮해야 할 것이다. 그런데, I/O핀 數는 3-1節에서 記述한 바와 같이 더 增加하지 않는다고 할 수 있으며, 또한 性能은 4章에서 考察하는 바, 여기서는 하드웨어의 增加 즉, 하드웨어 overhead에 대해서만 考察하기로 한다.

VLSI칩의 하드웨어量을 논할 때는 보통 트랜지스터나 게이트 等의 레벨에서 다루고 있으나, 여기서는 레지스터레벨에서 대략적으로만 구하기로 한다. 우선, 그림 3에서 보는 바와 같이 8비트 레지스터를 하드웨어量의 基本 1單位로 간주한다면, 故障診斷專用 하드웨어 및 8080A마이크로프로세서의 하드웨어量은 다음과 같다. 단, 프로세서의 內部버스와 診斷專用 버스는 하드웨어量이 同一하다고 본다.

1) 診斷專用 하드웨어量

① 檢査專用 레지스터 및 8비트 비교기…9.5 레지스터單位

② 檢査專用 메모리…機能檢査를 위한 最初檢査벡터의 數에 의존하며, 본 論文에서 채택한 機能檢査方法에 의하면 147레지스터單位가 된다.

③ TCU…TCU는 3-2節에서 詳述한 바와 같이, 서로 다른 制御動作機能이 15가지가 있으므로 이들 動作의 遂行을 위한 制御워드가 各 8비트로써 마이크로프로그램 制御메모리에 貯藏되어 있다고 하면, 15레지스터單位로 간주할 수 있다.

2) 프로세서의 하드웨어量

① 레지스터, 버퍼 및 래치…8레지스터單位

② ALU…DAA 機能을 包含한 9가지의 서로 다른 機能을 가지므로 上記한 바와 같이 9레지스터單位로 간주할 수 있다.

③ 制御部…命令코드가 8비트의 길이를 가지므로 256가지의 서로 다른 動作機能을 갖는다고 보아 TCU의 경우와 같이 256레지스터單位로 간주할 수 있다.

따라서, 故障診斷專用 하드웨어의 附加에 의한 하드웨어 overhead는 대략 (171.5/287)*100(%) =60%가 된다. 물론, 이 값은 正確한 overhead를 나타낸 것은 아니지만, 論文[9]에서 提示한 바와 같이 自己체크型 마이크로프로세서를 設計할 경우에, coding과 checking을 위한 檢査回路의 附加로 因한 하드웨어 overhead가 73%-105%에 이른다는 事實을 감안한다면 上記의 overhead값은 크지 않다고 볼 수 있다. 물론, 論文[9]에서는 Timing 및 制御裝置의 檢査도 包含하는 경우이므로, 이들 裝置의 檢査에 대한 overhead分을 考慮하드라도 그 나머지에 대한 overhead가 47%-68%가 되어, 결국 本 論文에

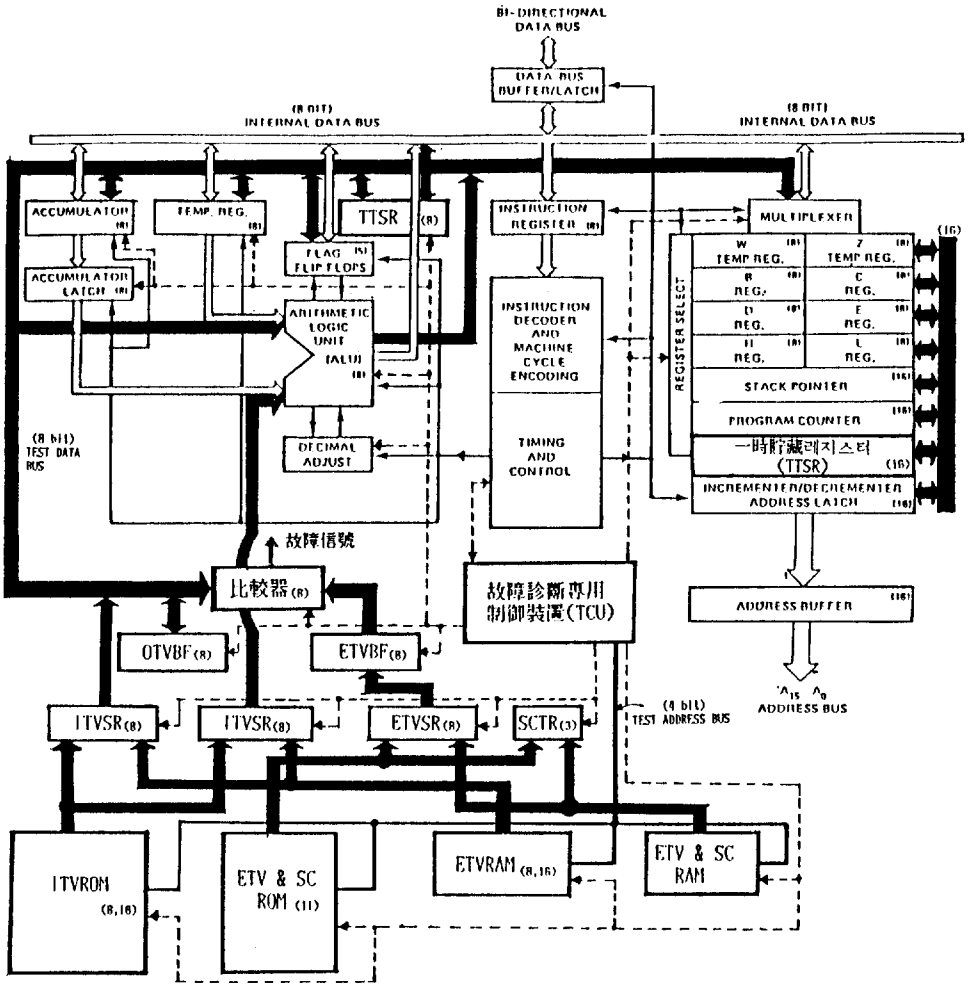

그림 3 自己診斷型 마이크로프로세서의 하드웨어 블럭도

Fig. 3 Hardware block diagram of a self-diagnosable microprocessor.

서 提案한 故障診斷 하드웨어의 構成方法의 妥當性을 間接的이나마 찾을 수 있다고 사료된다.

4. 同時 故障診斷의 性能評價

本 節에서는 이제까지 提示한 故障診斷方式을 汎用 8비트 마이크로프로세서에 適用한다고 할 경우에 있어서, 同時診斷의 觀點에서 본 性能을 評價한다.

프로세서가 任意의 주어진 問題프로그램을 遂行한다고 할 때, 프로세서內部의 各 機能유니트들 이 命令實行과 同時에 어느 程度의 故障 檢査가 이루어 지는가 하는 尺度로서 完全檢査比(CTR : Complete Testing Ratio)를 定義하고, 나아가 프로세서 내의 모든 機能유니트에 대하여 적어도 한번의 完全診斷에 必要한 命令實行의 回數인 同時診斷性能指標(Concurrent Diagnosis Performance Index)를 利用, 그 性能을 評價한다.

性能評價는 다음의 過程을 통하여 行한다.

1) 프로세서內의 各 機能유니트의 機能檢査에 必要한 檢査벡터를 구한다.

2) 프로세서가 遂行할 任意의 問題프로그램을選定한다.

3) 問題프로그램의 遂行에 參與하는 檢査命令의 種類및 實行頻度등의 情報를 얻을 수 있는 이른바

有效命令믹스를 구한다.

4) 有效命令믹스를 利用하여 各 機能유니트에대한 CTR을 구할 수 있으며, 따라서 이 CTR 로써 CDPI를 計算하여 프로세서내의 全體 機能유니트의 完全診斷에 걸리는 診斷時間(DT;Diagnosis Time) 및 完全診斷의 回數(NCD;Number of Complete Diagnoses)를 구한다.

4.1 同時診斷의 性能指標

(1) 機能유니트의 檢査性

機能유니트 fu(i)의 檢査性(Testability)이란, fu(i)의 檢査에 必要한 總 檢査벡터의 數에 대하여 TEST$_{ie}$ 中 fu(i)에 平均的으로 印加될 수 있는 檢査벡터 數의 比로 定義한다.

$$\text{즉, } t_i = \frac{\text{TEST}_{ie} \text{ 中에 印加되는 平均 檢査벡터의 數 } [(\text{TEST}_{ie}/\text{TVT})]}{\text{Card}[\text{IV}[\text{fu}(i)]]}$$

단, 여기서 Card[IV[fu(i)]]는 fu(i)의 總 檢査벡터 數이다.

(2) 檢査可用度

機能유니트의 檢査可用度(Availability)란 하나의 命令이 實行될 때 그 機能유니트가 UUT로 될 確率이라고 定義한다.

이제, 機能유니트 fu(i)의 檢査에 參與할 수 있는 檢査命令의 集合을 I[fu(i)]라 하고, 任意의 한 命令 ie의 實行頻度를 FREQ$_{ie}$라 하면 檢査可用度 a_i는 다음과 같이 구할 수 있다.

$$a_i = \sum_{\text{all } ie \in I[fu(i)]} \text{FREQ}_{ie}$$

단, $\text{FREQ}_{ie} = \dfrac{\text{命令 ie의 實行回數}}{\text{實行되는 命令의 總 實行回數}}$

이며, I[fu(i)]는 對象 프로세서의 各 命令들의 實行過程을 잘 分析함으로써 쉽게 구할 수 있다.

(3) CTR 및 CDPI

問題프로그램의 遂行中 任意의 한 命令이 實行될 때, 各 機能유니트 fu(i)에 대한 CTR$_i$은 다음과 같다.

$$CTR_i = t_i * a_i$$

따라서, fu(i)가 最小限 한번 完全檢査되는데 必要한 平均 命令實行의 回數는 CTR$_i$의 逆數와 같다.

結局, 프로세서內의 全體 機能유니트에 대한 CDPI는 다음 式과 같이 된다.

$$CDPI = \sum_{\text{all}} \left[\frac{1}{CTR_i} \right] = \sum_{\text{all}} \left[\frac{1}{t_i * a_i} \right]$$

4.2 適用 例

(1) 검사벡터의 印加時間

하나의 檢査벡터의 印加時間 TVT는 3-2.節 에 依據하여 다음과 같이 算定할 수가 있다.

우선, 診斷中 UUT의 選擇은 命令페취 後의 解讀中에 TCU에 의해 同時에 이루어 진다고 보고 이를 위한 所要時間은 考慮하지 않는다 또한, TCU에의해 同時에 遂行될 수 있는 制御動作時間은 檢査專用 메모리로부터의 R/W(2 T states)를 除外하고 1 T state가 所要된다고 간주한다면, 3-2.節에서 보는 바와 같이 TVT는 最大 7 T states라 할 수 있다. 특히, 故障診斷은 TCU에 의해 命令의 實行과는 전혀 獨立的으로 進行될 수가 있으므로 시스팀클럭 以上의 周波數下에서도 制御動作의 遂行이 可能하다. 따라서, TVT를 더욱 短縮시킬 수도 있어 TVT를 7 T states로 算定하는 것이 전혀 무리가 아니다.

(2) 問題프로그램

對象으로 하는 마이크로프로세서에 대하여, 그것이 遂行하는 問題프로그램으로서 데이터管理에 자주 利用되는 整列프로그램 中 Bubble sorting[16]을 例로 들었다. 이 問題프로그램의 遂行앨고리즘은 다음과 같다. 단, 任意의 n개의 要素가 메모리의 隣接한 記憶場所 A[i](1 i≤n)에 記憶되어 있다고 하며, 解析을 간단히 하기 위해 배열 A가 처음에 오름차순으로 整列되어 있는 最惡의 條件이라고 한다.

```
Procedure Bubble Sort[A]
  For j=n-1 Step-1 Until 1 Do
    For i=1 Step 1 Until j Do
      If A[i+1]<A[i]
        Then Interchange A[i] and A[i+1]
```

(3) 有效命令믹스

檢査命令集合 I에 속하는 命令들은 TEST$_{ie}$≥TVT의 條件을 滿足하여야 하므로 7 T states以上의 實行時間을 갖는 命令들이 I에 속한다. 따라서, 檢査命令集合에 속하는 명령들과 그들의 實行頻度를 구하면 表2와 같다. 단, 여기서는 整列할 要素의 數 n을 200으로 하였을 때이며, 이 때의 總 命令의 實行回數는 $8n^2+3=320003$이 된다.

(4) 故障診斷의 性能指標

上記의 有效命令믹스를 利用하고 機能유니트의檢査벡터를 考慮하면 各 機能유니트에 대한 檢査性및 檢査可用度, 그리고 完全檢査比를 表3과 같이 구할 수 있다. 특히 여기서는 解釋을 간단히 하기 위해 檢査命令集合에 속하는 모든 命令에 대하여 TEST$_{ie}$中에 印加되는 平均 檢査벡터의 數를 1로 하였다.

표 2 檢査命令과 實行頻度

Table 2 Test instructions and their frequency of occurrences.

Mnemonic	實行時間	實行回數	實行頻度
MVI B, #n	7T	2	0.0000
MVI C, #1	7T	200	0.0006
LXI D, #n	10T	400	0.0013
LXI H, #n	10T	400	0.0013
MOV A, M	7T	19900	0.0622
MOV M, A	7T	19900	0.0622
J<C>addr	7/10T	40000	0.1250
JMP addr	10T	39800	0.1244
CMP M	7T	19900	0.0622
LDAX D	7T	19900	0.0622
LDAX B	7T	19900	0.0622
STAX B	7T	19900	0.0622
PUSH PSW	12T	19900	0.0622
POP PSW	10T	19900	0.0622
RET	10T	1	0.0000

표 3 機能유니트의 檢査性, 檢査可用度 및 完全檢査比

Table 3 The testability, availability and CTR of functional units.

機能유니트 fu(i)	檢査性 t_i	檢査可用度 a_i	完全檢査比 CTRi	檢査벡터의 數
ALU : fu(1)	0.031	0.6880	0.019	32
Flag R : fu(2)	0.019	0.5630	0.011	20
Temp R : fu(3)	0.029	0.6880	0.020	34
Acc : fu(3)	0.028	0.2526	0.007	36
B Reg : fu(5)	0.028	0.6252	0.018	36
C Reg : fu(6)	0.028	0.6867	0.018	36
D Reg : fu(7)	0.028	0.6867	0.019	36
E Reg : fu(8)	0.028	0.5623	0.019	36
H Reg : fu(9)	0.028	0.5623	0.016	36
L Reg : fu(10)	0.028	0.5623	0.016	36
SP : fu(11)	0.014	0.6258	0.009	70

따라서, 表에서 보는 바와 같이 全體 機能유니트에 대한 CDPI는 다음과 같이 된다.

$$CDPI = \sum_{all} \left\lceil \frac{1}{CTR_i} \right\rceil = 793$$

4.3 考察

結局, 프로세서內의 모든 機能유니트에 대한 CTR의 逆數를 모두 더한 結果, 주어진 問題프로그램에 대한 CDPI 즉, 機能유니트 全體가 적어도 한번 完全診斷되기 위한 命令의 實行數를 구할 수 있었다. 따라서, 任意의 하나의 命令이 實行되는 時間을 平均的으로 대략 10 T states로 하고 또한 프로세서의 시스팀클럭의 周波數를 2MHz로 假定한다면, 프로세서內의 全 機能유니트의 完全診斷時間 DT는 다음과 같다.

$$DT = \frac{CDPI * 10\ T\ States}{2\ MHz} = 793 * 10 * 500\,(nsec)$$
$$= 3.97\,(msec)$$

또한, 問題프로그램이 遂行되는 동안에 全體 機能유니트가 몇 回정도 反復하여 檢査되는 가 하는, 이른바 完全診斷의 回數(NCD)는 320003/793=403回이다. 따라서, 自身의 問題프로그램을 遂行함과 同時에 全體 機能유니트를 約 400回 정도 反復하여 診斷을 하게된다.

5. 結 論

本 論文에서는 汎用 마이크로프로세서가 同時故障診斷의 機能을 갖도록 하기위해 必要한 基本的인 하드웨어構成方法과 그것들을 制御하기 위한 하드웨어動作 앨고리즘을 提示하였다.

本 論文에서 提示한 診斷方法을 하나의 汎用프로세서를 對象으로 適用할 때, 그 하드웨어 overhead는 旣存의 方法에 比해 그다지 크지 않고, 더구나 任意로 選定한 問題프로그램에 대하여 同時診斷面에서의 性能을 評價한 結果 역시 全體 機能유니트가 1回의 完全診斷에 所要되는 診斷時間은 아주 짧으며, 反復되는 診斷回數도 상당히 많음을 알 수 있었으므로, 本 方法의 効用性을 立證할 수 있었다.

특히, 本 論文에서 提示한 同時自己診斷方式의長點으로는 다음과 같은 것들을 들 수가 있다.

1) 遂行中인 問題프로그램만으로도 自己診斷이 充分히 可能하므로 프로세서의 通常 遂行速度에는 전혀 影響을 주지 않는다.

2) 提示한 同時診斷方式은 特定한 프로세서모델에만 局限하지 않고 一般的인 다른 汎用프로세서의 境遇에도 그대로 擴張하여 適用할 수가 있다.

3) 프로세서의 칩內部에 附加되는 故障診斷 專用하드웨어의 量은 많지 않을 뿐만 아니라, 이로 因한 外部 入出力핀의 增加도 없다.

4) 自己診斷機能을 附加시키기 위해 提示한 하드웨어 設計方法은, 旣存의 汎用 마이크로프로세서에 바로 適用할 수 있어 基本 設計思想을 크게 變

化시키지 않으므로 初期設計費用을 줄일 수 있다.

5) 프로세서 自身의 故障診斷을 빠른 時間에 遂行해 낼 수 있을뿐만 아니라 그 診斷回數도 상당히 많으므로, 이 프로세서를 包含한 全體 시스템레벨에서의 高信賴化에 크게 寄與할 수 있다.

그러나, 本 論文에서는 프로세서內의 하드웨어중 制限된 機能유니트들 만이 UUT가 되고 그밖에 프로세서 制御裝置등과 診斷專用 하드웨어는 故障이 없다는 假定下에서 同時故障診斷方法을 提示하였기 때문에 이 部分들의 故障診斷역시 앞으로 考慮해야 할 것이다.

참 고 문 헌

1) V.P.Nelson and B.D.Carroll, Tutorial : Fault-Tlerant Computing, pp.1-4, IEEE Computer Society Press (1987).

2) S.A.Elkind, "Reliability and Availability Techniques," Chp.3 of Reliable System Design : Theory and Practice, Digital Press(1982).

3) S.M.Thatte and J.A.Abraham, "Test Generation for Microprocessors," IEEE Trans. on Computers C-29 (6), pp.429-441 (Jun.1980)

4) W.Barraclought, A.C.L.Chang and W.Sohl, "Techniques for Testing the Microcomputer Family," Proceedings of IEEE 64(6), pp.943-950 (1976).

5) 小鳩 撤 外, "マイクロプロセツサのテスト手法 とテスト容易化設計" 情報處理 25卷 10號, pp. 1125-1130 (Oct. 1984).

6) B.Koenamann. J.Mucha and G.Zwieoff, "Built-in-Test for Complex Integrated Circuits," IEEE J. of Solid-States Circuits 15(3), pp.315-321 (Jun.1980).

7) D.R.Renick, "Real World Built-in Test for VLSI " COMPCON 86, 31th. IEEE Comp. Society Int. Conf., pp 436-440 (1986).

8) J.Wakery, Error Detection Codes, Self-Checking Circuits and Applications, Elsevier North Holland (1978).

9) Y.Crouzet and C.Landrault, "Design of Self-Checking MOS-LSI Circuits : Application to a Four-Bit Microprocessor," IEEE Trans. on Computers C-29(6), pp. 532-540 (Jun.1980).

10) 南谷 崇, 河村 後明, "セルフチエキングプロセッサの一構成法" 日本電子通信學會論文誌 68卷 12號, pp.2-015-2026 (Nov.1985).

11) M.P.Halbert and S.M.Bose, "Design Approach for a VLSI Self-Checking MIL-STD-1750A Microprocessors," FTCS-14, pp.254-259(Jun. 1984).

12) M.M.Tsao, D.P.Siewiorek and etc., "A Single Chip Fault-Tolerant Microprocessor," FTCS-12, pp.63-69 (Jun.1980).

13) R.M.Sedmak and H.L.Liebergot, "Fault-Tolerance of a General Purpoes Computer Implemented by Very Large Scale Integration," IEEE Trans. on Computers C-29(6), pp.492-500 (Jun.1980).

14) A.C.L.Chiang and R.McCaskill, "Two Approaches simplify Testing of Microprocessors," "Electronics, pp.100-105 (Jan. 1976).

15) M.A.Breuer and A.D.Friedman, Chp.3 of Diagnosis and Reliable Design of Digital Systems, Computer Science Press Inc. (1976).

16) Aho, Hopcroft, Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesely Pub. Co (June 1976).