STRAATEGIES FOR FUNCTIONAL TESTING OF MICROPROCESSORS

A. Noore
West Virginia University

B.E. Weinrich
California University

ABSTRACT

Effective strategies for generating tests for microprocessors are presented. Modular block, comprehensive instruction set and microinstruction set approaches are proposed. These practical approaches are viable alternatives to the exhaustive testing which aims at considering all possible instructions, addressing modes and data patterns. The proposed approaches are versatile and are effective especially in the user environment.

INTRODUCTION

As the complexity of microprocessor increases, the problem of testing to ensure proper operation becomes very challenging. Additional features such as memory management, floating-point processing, and cache memory are being integrated into the chip to enhance performance and speed. Chip densities of more than a million transistors are being introduced, which further makes testing more difficult. Manufacturers are using testability techniques at chip level in order to enhance fault coverage. Generation of test vectors is usually derived from the knowledge of the internal structure of the chip. Even with all the information available, the test complexity becomes impractical to verify exhaustively all possible combinations of several parameters. Test vectors generated from the physical layout and translated to a set of microprograms are also very complex especially in the users environment [1]. Functional tests for microprocessors have been widely used both at the manufacturers and users environment as a solution to the limitations of exhaustive testing. Further, some microprocessors are tested so that the test complexity is a function of a specific application. Application oriented tests do not verify all possible operations of the microprocessor but instead exercises tests covering all possible paths for a dedicated application. In this paper, several strategies are propounded to facilitate testing microprocessors using the information available to the user.

MICROPROCESSOR MODEL

In order to develop a functional test strategy for a microprocessor, the system level behavior that describes the correct operation is defined. Using the information provided by the manufacturer, regarding the functional modular blocks, timing, instruction set, and microinstructions, the microprocessor model is developed. A microprocessor is defined by a basic set of modular blocks such as the Arithmetic Logic Unit, general purpose registers, multiplexer, stack pointer, program counter, instruction register, instruction decoder, read write memory, input/output ports and buffers. These modular blocks are interconnected so that when the microprocessor instructions are executed, all appropriate blocks perform the correct functions. When a physical failure occurs in the microprocessor, it causes the modular blocks to function incorrectly during the testing process.

FAULT MODEL

Arithmetic and Logic Unit (ALU): All microprocessors have a section in the hardware that performs arithmetic and logical operations. Under a faulty condition,

a) An incorrect ALU operation is performed

b) Incorrect status flag(s) are generated.
General purpose registers, stack and read write memory: The general purpose registers are modeled as a subset of read write memory. Under a faulty condition,

a) a cell in the read write memory array will not be capable of undergoing a change from its present state to its complementary state and/or from its complementary state to its original state.

b) if \( i \) is the address of cell \( C_i \), then any cell(s) other than \( C_i \) will respond to a read or a write operation when addressed.

c) an operation on any cell \( C_i \) will influence the state of other cells in the memory array due to restrictive coupling. Let \( MC_n \) denote the read write memory array of cells \( C_0, C_1, C_2, \ldots, C_{n-1} \). A memory zone is defined as a set of contiguous cells in \( MC_n \). Although this can represent any arbitrary set of cells, the memory zone is defined as a set of cells that are contained within valid lower and upper boundaries, both inclusive. For the memory \( MC_n \), the notation \((0, n-1)\) represents the boundary conditions. An active memory zone (AMZ) refers to that portion of memory zone that is subjected to functional testing at any instant of time. An inactive memory zone (IMZ) refers to that portion of the memory zone that is not being subjected to functional testing at any instant of time.

Multiplexers: Under a faulty condition in the microprocessor, if the multiplexer is affected, one of the following conditions would occur

a) either an incorrect source or no input source is selected

b) more than one input source is selected so that the output is unpredictable.

Instruction decoder: Depending on the number of bits of the microprocessor and the instruction length, the microprocessor decodes the contents of the instruction register to selectively activate only one out of a number of output lines. Under faulty conditions,

a) no output is activated or more than one output is activated.

In either case the incorrectly activated line combines with other system timings causing the wrong control or function to be performed when a specific instruction is executed.

Control block: Various modular blocks that describe the microprocessor model are activated and deactivated by the control block logic. The control block operations are synchronized by the clock. Under faulty conditions,

a) improper sequence of events are triggered that does not correspond to the instruction or microinstruction processed.

b) improper response to external signals such as interrupt, wait or DMA operations

c) improper internal status information is provided externally.

TEST STRATEGIES

The microprocessor model described above is activated by a set of instructions and microinstructions provided by the manufacturer.

Modular block approach

Each module that describes the microprocessor is tested individually. The accumulator is the most frequently used register. Using the instruction set of the microprocessor different test diagnostics are developed. Independence of various tests are maintained by grouping all instructions that relate to a particular module. Each instruction cycle is subdivided into machine cycles \( M_i \) and each machine cycle is further subdivided into \( T_j \) states. \( T_j \) represents the time required for the smallest activity performed by the microprocessor. Sets of instructions that correspond to the accumulator, (\( a_1, a_2, a_3, \ldots, a \)), temporary register, (\( t_1, t_2, t_3, \ldots, t \)), and the Arithmetic Logic Unit, (\( A_1, A_2, A_3, \ldots, A_k \)), are grouped. These sets are minimized so that they are mutually exclusive. The reduced set \( R \) covers all the instructions involving the above modules. To this core \( R \), a few basic instructions, \( B \), are added for communicating the status of execution after the diagnostics is run. The set \( (R+B)\) should be able to test for stuck-at-0 (s-a-0) faults, stuck-at-1 (s-a-1) faults and other functions of the modules systematically.
In a similar manner for each modular block i, a set of minimal instructions (R+Bli) is derived and exercised. For testing the read write memory on microprocessor chips with cache memories, at any instant if i cells are subjected to test, then the AMZ is defined by the boundary condition (0, i-1) while the IMZ is defined by the boundary conditions (i, n-1). As the test proceeds, i varies and causes the upper and lower boundaries of the AMZ and the IMZ to change dynamically. The test algorithm is described by the following steps.

Step 1: A background pattern of all 0s and 1s are written into the entire memory.

Step 2: A cell C_i is written to store a value that is the complement of the background pattern.

Step 3: All cells in the AMZ except cell C_i are read sequentially downwards to verify that they contain the background pattern and that the contents are not changed due to the interaction between cells.

Step 4: Cell C_i (current AMZ, IMZ boundary) is read to verify the write operation of Step 2.

Step 5: Cell C_i is restored to its original value.

Step 6: Repeat Steps 2, 3, 4, and 5 for all values of i, 0 ≤ i ≤ (n-1).

Step 7: Repeat Steps 1 through 5 complementing the background pattern in Step 1 and reading sequentially upwards in Step 3. Let the AMZ increase dynamically starting from cell (n-1) and progressing towards cell 0.

The proposed test algorithm is a variation of the Walking Is and Os. The functionality of the memory MC_n is verified by testing only the AMZs. At any instant, i cells of MC_n are subjected to test. For these i cells, Property 1 is verified by Steps 1, 2, 4, and 5 using the original and complemented background patterns. Property 2 is verified by Steps 3 and 4. Property 3 is verified by assuming that a read operation of any cell in the AMZ does not affect the state of the other cells. The algorithm however tests for the effect of the cell interactions in the AMZ due to the write operations being performed in each AMZ-IMZ boundary cell. The proposed algorithm does not test the IMZ cells defined by [i, n-1]. As the test progresses, the upper boundary of the AMZ dynamically increases because a cell from the IMZ gets dissociated and becomes part of the AMZ. This process continues until the IMZ no longer exists, and all cells in the MC_n are contained in the AMZ. The entire memory is now verified.

If t represents the cycle time for a read or write operation of MC_n, then for a given background pattern to be loaded in MC_n, (Step 1), n operations are required. In addition each cell C_i undergoes two write operations (Steps 2 and 5) and one read operation (Step 4). The remaining (i-1) cells in the AMZ at the instant are read sequentially (Step 3). The test time T is given by

\[ T = \left[ n + 2n + \frac{n(n+1)}{2} \right] t = n(n + 7)t \]

The test time required by WAKPAT is 2n(n + 3)t while that of GALPAT is 2n(2n + 1)t. [7]

In order to test the stack pointer for s-a-0 and s-a-1, some microprocessor architectures do not have provisions for reading from the stack pointer. So in order to set the stack pointer to any desired value or all zeros or all ones, verification becomes difficult. An example of such an architecture is the Intel 8085 8-bit microprocessor. Using additional instructions from set B, the following sequence of instructions

LXI SP, FFFF; loads stack pointer with ones
LXI H, 0000; clears H and L register pair
DAD SP; contents of SP are added to contents

will detect if any bit in the stack pointer is s-a-0

Similarly, LXI SP, 0000
LXI H, 0000
DAD SP

will detect any s-a-1 fault in the stack pointer by examining the contents of the H and L register pair. Other tests related to the stack pointer includes checking if the stack pointer can automatically increment or decrement during push and pop operations.

Even though the microprocessor is modeled as an interconnection of several modular blocks, these blocks are not independent. Localization of faults is contingent on the following factors.
1. Function of each modular block characterized by the fault and the extent of its influence on other modules.

2. Number of such faulty modular blocks in the microprocessor model.

3. The disposition and nature of the faults encountered such as stuck-at faults, intermittent faults or transient faults.

**Comprehensive instruction set approach**

In the modular block approach, each module \( i \) was tested by a set of \( (R+B)_i \) instructions. These instructions within any set \( (R+B)_i \) were mutually exclusive and minimized. But for the microprocessor model made up of \( k \) modular blocks, the testing complexity is given by

\[
\sum_{i=1}^{k} (R + B)_i
\]

In the comprehensive instruction set approach, a global minimization of all instructions required to test various modular blocks is performed. The reduced instruction set is augmented with another set of instructions, \( P \), that are defined in the microprocessor model but are not contained in the set \( \min \sum_{i=1}^{k} R_i \). The fault coverage is enhanced due to exercising the extra instructions from set \( P \). The complexity is reduced due to the global minimization. The overall test complexity is given by

\[
\sum_{i=1}^{k} (R)_i + P + B
\]

The advantage of this approach is that the diagnostics can be developed to exercise the functional behavior of all the modules within the microprocessor and also verify the execution under different address modes. If the diagnostics terminates at the proper address with the correct terminal response, the microprocessor is assumed to be functional. Experimental runs have shown that a higher coverage is obtained when the results of diagnostics are verified at intermediate levels rather than at the very end of execution. This is because the interdependency on the result of the previous instructions can cause failures to be masked. Faults can be localized by using additional error detecting subroutines once a failure is detected.

**Microinstruction set approach**

An effective approach that uses the data provided by the manufacturer is proposed. All instructions are executed as a sequence of machine cycles which are further subdivided into clock periods or states. Certain instructions are subjected to the same microsequences during each state of every machine cycle. A check on the microprocessors activities at different clock periods during each machine cycle is effective in guaranteeing successful functional operation. For example, in the Intel 8085 microprocessor, during state T1 of the first machine cycle a memory or I/O address is placed on the address bus and the status information of the microprocessor is made available. During state T2, the program counter is incremented by 1 and the READY and HOLD inputs are checked. During state T3, an instruction byte is fetched from the data bus. These three microinstructions are common to all instructions for the microprocessor model. Likewise there are several microinstructions that are executed at identical clock periods. In this approach, all microinstructions that cover the flow of data through all interconnecting paths and modular blocks of the microprocessor are selected. Minimization is performed at microinstruction level. Compared to the comprehensive instruction set approach, the microinstruction set approach drastically reduces the number of instructions that form the diagnostics. Consider the following arithmetic and logic instructions \( \text{ADD} \), \( \text{ANA} \), \( \text{XRA} \) and \( \text{ORA} \). A sample diagnostics formulated under the comprehensive instruction set approach will include all the mutually exclusive instructions as shown.

\[
\begin{align*}
\text{LXI} & \ B, \ FF00 ; \quad \text{loads} \ B \text{ and } C \text{ with patterns FF and 00} \\
\text{XRA} & \ A ; \quad \text{clears accumulator} \\
\text{JNZ} & \ \text{FSR} ; \quad \text{if the above is not executed correctly}, \\
\text{.............} & ; \quad \text{jump to faulty subroutine} \\
\text{.............} & ; \\
\text{ADD} & \ B ; \quad \text{performs} \ (A) + (B) > (A) \\
\text{CMP} & \ B ; \\
\text{JNZ} & \ \text{FSR} ; \\
\text{.............} & ; \\
\text{.............} & ; \\
\text{ANA} & \ C ; \quad \text{performs} \ (A) \ \text{AND} \ (C) > (A) \\
\text{JNZ} & \ \text{FSR} ; \\
\text{.............} & ;
\end{align*}
\]
The complexity of this diagnostics is 21 bytes. For the same example using the microinstruction set approach, a significant minimization is performed.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Machine Cycle</th>
<th>State</th>
<th>Microinstructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD r</td>
<td>M1</td>
<td>T1</td>
<td>PC OUT, STATUS</td>
</tr>
<tr>
<td>ANArC</td>
<td>T2</td>
<td></td>
<td>PC = PC + 1</td>
</tr>
<tr>
<td>XRA r</td>
<td>T3</td>
<td></td>
<td>INST &gt; Inst Reg</td>
</tr>
<tr>
<td>ORA r</td>
<td>T4</td>
<td></td>
<td>Source &gt; Temp Reg</td>
</tr>
<tr>
<td></td>
<td>M2</td>
<td>T1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T2</td>
<td>(ACT) + (TMP) &gt; A</td>
</tr>
</tbody>
</table>

*For this architecture, the results of these arithmetic and logical instructions are not moved into the accumulator A until state T2 of the next instruction cycle. A is loaded while the next instruction is being fetched. This overlapping of operations allows for faster processing. The following diagnostics uses the minimized microinstruction sequences and has a complexity of 7 bytes.

Additional set of instructions are required to include other failure modes not covered by the minimized set of microinstruction sequences.

CONCLUSION

Methodologies to develop diagnostics to test microprocessors functionally have been proposed. These approaches can be practically applied in the user environment. They are developed based on the information provided by the manufacturer. These approaches are versatile and are applicable to a number of other microprocessor architectures. The fault models do not include pattern sensitive faults or transient faults. Testing under these conditions can be quite exhaustive and many require additional external hardware.

ACKNOWLEDGMENTS

This research is funded in part by Electric Power Research Institute and the Energy and Water Research Center, WVU.

REFERENCES


