Fig 4-13: Loading a new instruction
- Star architecture

- Ring 1

- Ring 2

- Internal scan test

- Instruction = 101

  how to design the boundary scan ??
\[ a_0 \rightarrow 0 \quad 0 \quad 1 \]
\[ a_1 \rightarrow 1 \quad 0 \quad 0 \]
\[ a_2 \rightarrow 0 \quad 1 \quad 0 \]
\[ a_3 \rightarrow 1 \quad 0 \quad 1 \]
\[ a_4 \rightarrow 1 \quad 1 \quad 0 \]
\[ a_5 \rightarrow 1 \quad 1 \quad 1 \]
\[ a_6 \rightarrow 0 \quad 1 \quad 1 \]
\[ a_7 \rightarrow 0 \quad 0 \quad 1 \]
\[ a_8 \rightarrow 1 \quad 0 \quad 0 \]
\[ \frac{1 + x^2 + x^3}{1 + x + x^2} \]

\[ x + x^3 + x^4 + x^5 + x^6 + \ldots \]

\[ x \\
x + x^2 + x^4 \\
\]

\[ x^3 + x^5 \]

\[ x^3 + x^5 + x^6 \\
\]

\[ x^4 + x^5 + x^6 \]

\[ x^4 + x^5 + x^6 + x^7 \]

\[ x^5 + x^7 \]

\[ x^5 + x^7 + x^8 \]

\[ x^8 \]

\[ a_0 x^2 + a_1 x^3 + \ldots \]

\[ = (0, 1, 0, 1, 1, 0, 0, 1, \ldots) \]
\[ p(x) = 1 + c_1x + c_2x^2 + c_3x^3 \]

\[ = 1 + x^2 + x^3 \]

\[ a_4 = 0 \quad a_3 = 1 \quad a_2 = 1 \quad a_1 = 1 \]

\[ \text{Characteristic polynomial} \rightarrow \]

\[ \text{Initialize polynomial} \rightarrow \]

\[ I(x) = c_1x^1(a_1x^1) + c_2x^2(a_2x^2 + a_1x^1) + c_3x^3(a_3x^3 + a_2x^2 + a_1x^1) \]

\[ x = x + x^2 + x^3 = 0 + x^2 + x^3 \]
\[
\begin{align*}
\frac{2m - 1}{2^n - 1} &= 2^n - 1 \\
\end{align*}
\]

# of possible error responses

\[p(\text{error response}) = \frac{# \text{ of error responses after detection}}{\text{compilation}}\]
BIST - CSBL

- Centralized and Separate Board-Level BIST [Benowitz 75]

\[ k = \left\lceil \log_2 m \right\rceil \]

- Use only one Signature Register
- Tests repeat m times to reduce hardware cost

BIST - Example
BIST - BEST

- Built - Evaluation and Self Test [Resnick 83]

- Pseudo random testing
- Hardware overhead is low
- Test length can be long for CUT with random-pattern resistant faults.

BIST - Example
BIST - STUMPS

- Self-Test using MISR and Parallel SRRS [Bardell 82, 84]
- Multiple scan chain to reduce test time
BIST - CSTP

- Circular Self Test Path [Krasniewski 89]
- Similar to SST except that boundaries are scanned
- Not all registers are self-test path registers.

BIST - Example
BIST - CBIST

- Concurrent Built-in Self Test [Saluja 88]
- Detect test patterns from normal inputs sequence
- Once a pattern is detected, compress the response and tick the test clock.
- If waited too long, insert a test pattern from PRPG.

BIST - Example
BIST - Weighted Pseudo Random Test

LFSR Based

BIST - TestGen - LFSR
Space Compression

- Use space compression to handle large output circuits.
- Use XOR gates to compress space.
- Use Error Control Coding to achieve better fault coverage.
- Example: A 16 SEC-DED code compresses 16 outputs into 5.

BIST - ResComp
BIST - Space and Time Compression

Space-Time Compression

Time-Space Compression

BIST - ResComp
# Functional Faults

<table>
<thead>
<tr>
<th>* Cell stuck</th>
<th>* Driver stuck</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Read/write line stuck</td>
<td>* Chip-select line stuck</td>
</tr>
<tr>
<td>* Data line stuck</td>
<td>* Open in data line</td>
</tr>
<tr>
<td>* Short between data lines</td>
<td>* Crosstalk between data lines</td>
</tr>
<tr>
<td>* Address line stuck</td>
<td>* Open in address line</td>
</tr>
<tr>
<td>* Open decoder</td>
<td>* Shorts between address lines</td>
</tr>
<tr>
<td>* Wrong access</td>
<td>* Multiple access</td>
</tr>
<tr>
<td>* Cell can be set to 0 but not to 1 (or vice-versa)</td>
<td></td>
</tr>
<tr>
<td>* Pattern sensitive interaction between cells</td>
<td></td>
</tr>
</tbody>
</table>
Neighborhood Pattern Sensitive Fault

* A Pattern Sensitive Fault (PSF) is defined as follows: The contents of a cell, or the ability to change the contents, is influenced by the contents of all other cells in the memory.

* The PSF can be considered the most general case of the k-coupling fault.

* The PSF model allows the neighborhood to take on any position in the memory array.

* When the neighborhood is allowed to take on only a single position, one speaks about a Neighborhood Pattern Sensitive Fault (NPSF).
Address Decoder Fault (AF)

* Address decoder Faults (AFs) concern faults in the address decoder.

* Functional faults within the address decoder will result:
  1. With a certain address, no cell will be accessed.
  2. There is no address with which this cell can be accessed. A certain cell is never accessed.
  3. With a certain address, multiple cells are accessed simultaneously.
  4. A certain cell can be accessed with multiple addresses.

![Diagram of address decoder fault](image)
## Relationship between Faults

<table>
<thead>
<tr>
<th>Fault</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAF</td>
<td>* Cell stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>* Driver stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>* Read/write line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>* Chip-select line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>* Data line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>* Open in data line</td>
</tr>
<tr>
<td>CF</td>
<td>* Short between data lines</td>
</tr>
<tr>
<td>CF</td>
<td>* Crosstalk between data lines</td>
</tr>
<tr>
<td>AF</td>
<td>* Address line stuck</td>
</tr>
<tr>
<td>AF</td>
<td>* Open in address line</td>
</tr>
<tr>
<td>AF</td>
<td>* Open decoder</td>
</tr>
<tr>
<td>AF</td>
<td>* Shorts between address lines</td>
</tr>
<tr>
<td>AF</td>
<td>* Wrong access</td>
</tr>
<tr>
<td>AF</td>
<td>* Multiple access</td>
</tr>
<tr>
<td>TF</td>
<td>* Cell can be set to 0 but not to 1 (or vice-versa)</td>
</tr>
<tr>
<td>NPSF</td>
<td>* Pattern sensitive interaction between cells</td>
</tr>
</tbody>
</table>
* Case C is wrongly accessed if \( \text{accen}_A \)

\[ \text{accen}_A \]

will be detected in

\[ \text{accen}_A \]

when \( \text{bank}_2 \) to \( C \), its value has been changed to 1.

\( \text{bank}_2 \)

But, 0 is expected.

\( \text{bank}_2 \)

* If a cell is never accessed,

then C is never accessed.

\( \text{accen}_A \)

If the value of \( \text{accen}_A \) will be given for the read \( \Rightarrow \) The error can be immediately detected.

\( \text{accen}_A \)

A \((R_0, w_0)\) will detect a 1.

A \((R_0, w_0)\) will detect a 1.

\( \text{accen}_A \)

\( \text{accen}_A \)

\( \text{accen}_A \)

\( \text{accen}_A \)
# Tests for Stuck-At, Transition and Coupling Faults

<table>
<thead>
<tr>
<th>March alg.</th>
<th>Test len.</th>
<th>Fault coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATS</td>
<td>4n</td>
<td>Some AFs, SAFs</td>
</tr>
<tr>
<td>MATS+</td>
<td>5n</td>
<td>AFs, SAFs</td>
</tr>
<tr>
<td>Marching 1/0</td>
<td>14n</td>
<td>AFs, SAFs, TFs</td>
</tr>
<tr>
<td>MATS++</td>
<td>6n</td>
<td>AFs, SAFs, TFs</td>
</tr>
<tr>
<td>March X</td>
<td>6n</td>
<td>AFs, SAFs, TFs, CFins</td>
</tr>
<tr>
<td>March C-</td>
<td>10n</td>
<td>AFs, SAFs, TFs, CFins, CFids</td>
</tr>
<tr>
<td>March A</td>
<td>15n</td>
<td>AFs, SAFs, TFs, CFins, L-CFids</td>
</tr>
<tr>
<td>March Y</td>
<td>8n</td>
<td>AFs, SAFs, CFins, TFs_I/w_CFinds</td>
</tr>
<tr>
<td>March B</td>
<td>17n</td>
<td>AFs, SAFs, CFins, L_CFids</td>
</tr>
<tr>
<td>TFs_I/w_CFids</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CF_{ids}:
  - $<\uparrow;0> <\uparrow;1>$
  - $<\downarrow;0> <\downarrow;1>$
  - $\otimes \rightarrow \otimes \uparrow; CF_{ids}$
  - Fixed Value: ids

*Note: Some of the tests are marked as 'idempotent'*
## NPSF Test Patterns

<table>
<thead>
<tr>
<th>Fault type</th>
<th># of test patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANPSF</td>
<td>(k-1)*2^k</td>
</tr>
<tr>
<td>PNPSF</td>
<td>2^k</td>
</tr>
<tr>
<td>APNPSF</td>
<td>k*2^k</td>
</tr>
<tr>
<td>SNPSF</td>
<td>2^k</td>
</tr>
</tbody>
</table>

* k: The size of the neighborhood.
* ANPSF: Active Neighborhood Pattern Sensitive Fault
* PNPSF: Passive Neighborhood Pattern Sensitive Fault
* APNPSF: Active+Passive Neighborhood Pattern Sensitive Fault
* SNPSF: Static Neighborhood Pattern Sensitive Fault
Can we just replace 0 by oo...oo and T by II - T?

Assume single bit in each memory operand.

Background Data
Note: All complex parts are different.

Yes, the factored can also be deduced if the complex cases can be deduced. How about friends? Can be different.

2) Reversal of background data
3) Reversal of background data
4) Reversal of background data
5) Reversal of background data
6) Reversal of background data
7) Reversal of background data
8) Reversal of background data
9) Reversal of background data
10) Reversal of background data

ASSUMES 8 = 4 (4 bits/word)
If the complex numbers are in the same parallelogram.

**Case 1:** CP dominates the union operation.

If CP will have no effect.

The value specified in the union operation will be.

**Case 2:** Where dominates the complex effect.

Lucky Case 3:

This will be determined by the union and merge operators.
Lena can still be calculated.

3. All states of two arbitrary cases \( i \) and \( j \) in the same

\[
\begin{array}{c}
00001111 \\
11110000 \\
00110011 \\
11001100 \\
01010101 \\
10101010 \\
11111111 \\
00000000
\end{array}
\]

\( i \)

Normal

Required background data

\( \{ <0, T> \} \)

\( <0, 0> \)

\( \circ \) to \( T \)

\( 0 \) to \( T \)

\( 0 \) to \( L \)

\( T \) to \( 0 \)

\( 0 \) to \( T \)

\( <0, T> \), \( <T, T> \), \( <0, 0> \), \( <0, 0> \), \( <0, 0> \)

State Count / Facts (SEF)