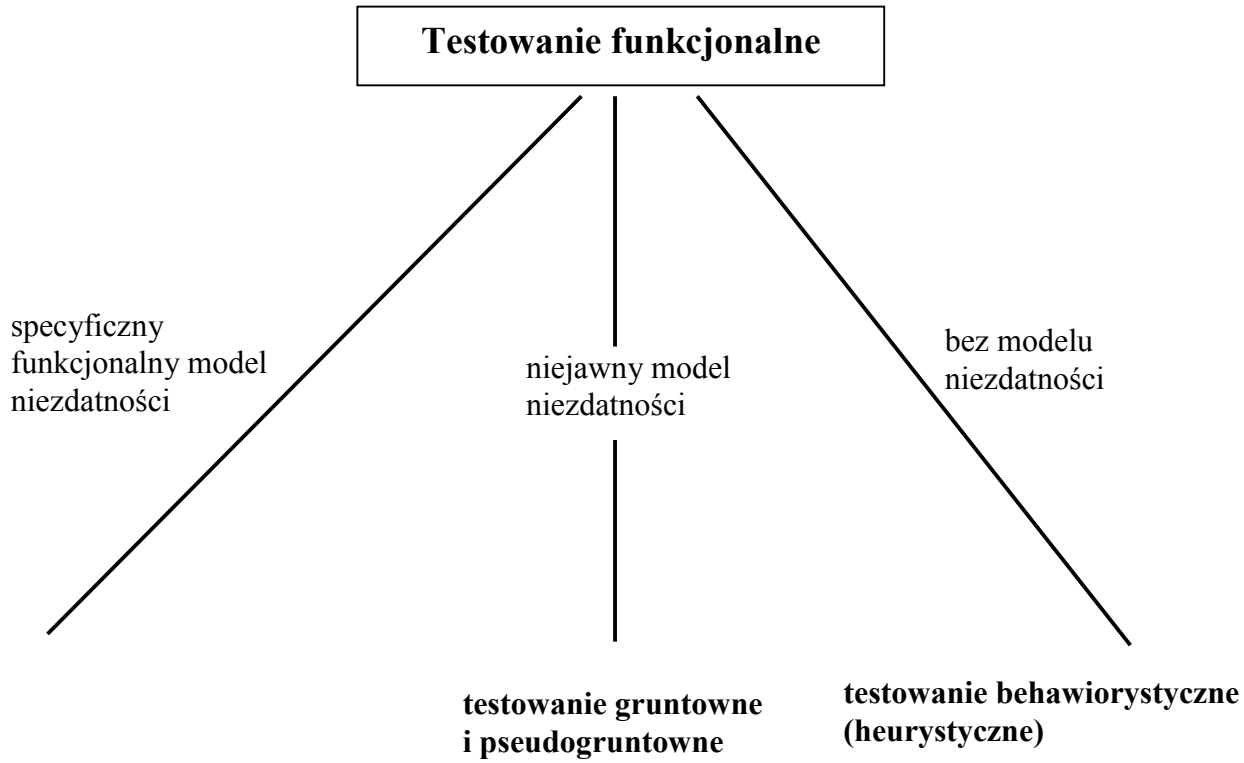
Testowanie funkcjonalne

Dotychczas omówione metody generowania testów opierały się na strukturalnym modelu systemu i ich celem było wyznaczenie testów dla wykrywania niezdatności na poziomie struktury systemu, takich jak uszkodzenia stałosygnałowe lub zwarcia zmostkowania.
Wady takiego podejścia:

- trudności w zastosowaniu do współczesnych systemów,
- brak modeli układów VLSI,
- zbyt wielka złożoność współczesnych układów powoduje określone problemy związane ze złożonością obliczeniową.

Metody testowania funkcjonalnego bazują na funkcjonalnym modelu systemu.
Celem testowania funkcjonalnego jest ocena poprawności operacji systemu ze względu na zgodność z jego specyfikacjami funkcjonalnymi.



Istotą **testowania funkcjonalnego** jest założenie ograniczonego zbioru niezdatności systemu, które wyrażane są na poziomie jego funkcji.
Podstawę testowania gruntownego stanowi założenie o możliwości wystąpienia dowolnego uszkodzenia z przyjętego zbioru niezdatności.

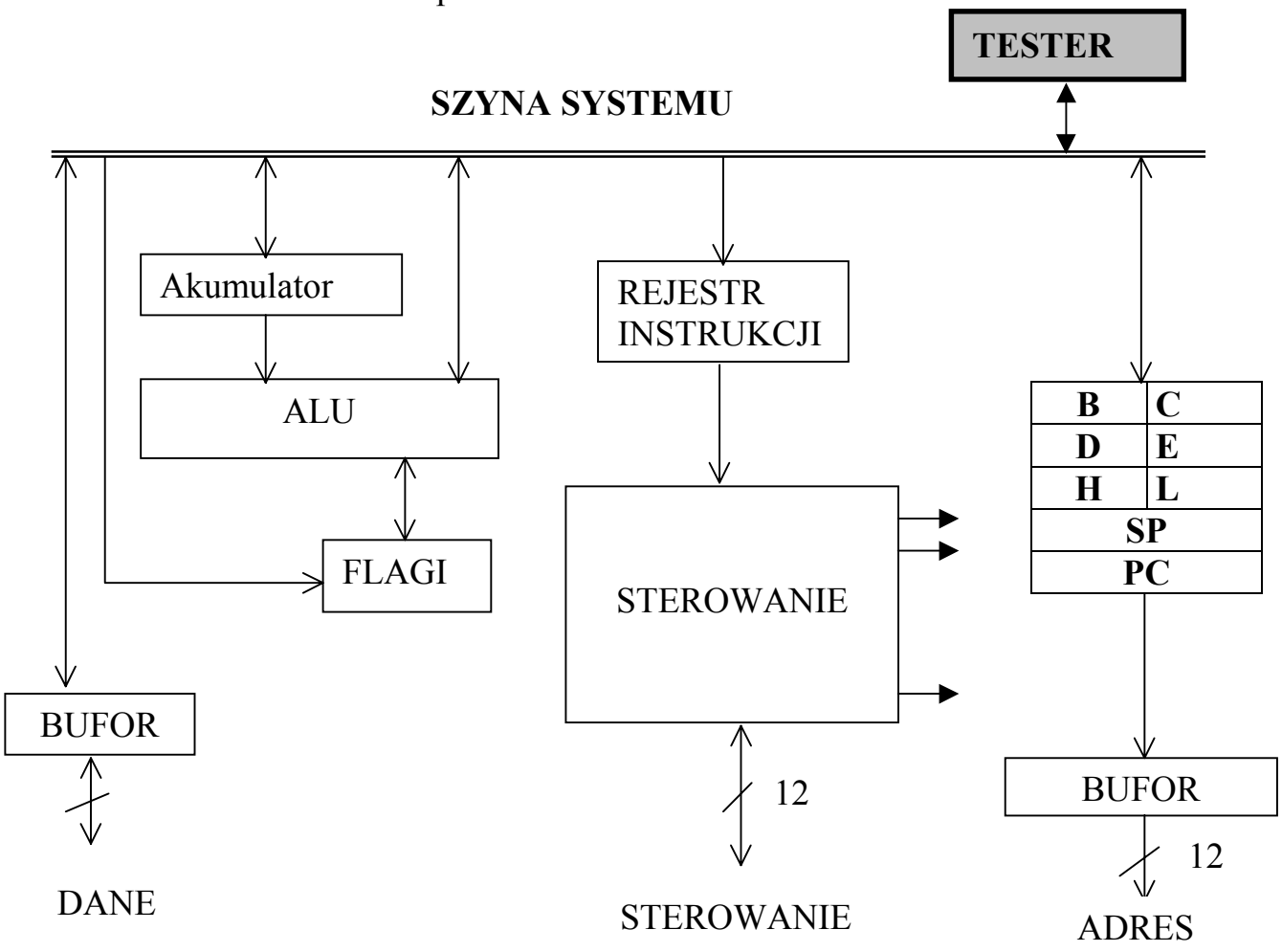
1. Testowanie funkcjonalne bez modelu niezdatności

1.1. Metody heurystyczne

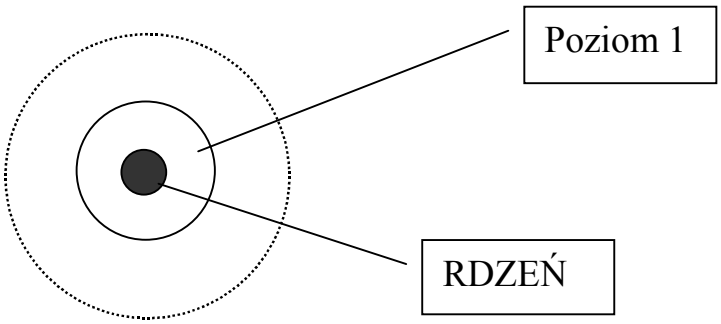
Metody heurystyczne wyrażają próby sprawdzenia funkcji systemu sformułowane *ad hoc*.

Przykład 1.
Test funkcjonalny przerzutnika obejmuje:
- test ustawienia wyjścia na wartość „0” i zmiany „0→1”,
- test ustawienia wyjścia na wartość „1” i zmiany „1→0”,
- ocena, czy przerzutnik może utrzymywać ustalony stan.

Przykład 2.
Testowanie mikroprocesora INTEL 8080.
Zakłada się, że testowanie realizowane jest przez tester zewnętrzny. Tester zewnętrzny jest podłączony do szyny systemowej (steruje szyną systemową). Tester zawiera program (instrukcje), które są wykonywane dla sprawdzenia rezultatów działania mikroprocesora.



Testy funkcjonalne mikroprocesora:



1. Test licznika rozkazów (PC):
   a) reset procesora 8080 i ustawienie inicjalne PC,
   b) umieszczenie operacji NOP na szynie danych i zmuszenie 8080 do cyklicznego wykonywania tej operacji aż PC przejdzie przez 2^{16} stanów (zawartość PC jest dostępna na szynie adresowej).
2. Test rejestrów H i L:
   a) zapis 8-bitowych wzorców do H i L za pomocą instrukcji MVI (operand bezpośredni),
   b) przesłanie zawartości H i L do PC (instrukcja PCHL),
   c) punkty a) i b) powtarzane są dla wszystkich wzorców 8-bitowych.
3. Test rejestrów B, C, D i E:
   W podobny sposób tester zapisuje 8-bitowe wzorce danych do rejestru $R \in \{B, C, D, E\}$. Następnie R jest przesyłane do PC poprzez H lub L (R nie może być bezpośrednio przesłane do PC). Wykorzystuje się tu fakt, że PC i H, L zostały przetestowane w pkt. 1 i 2.
4. Test wskaźnika stosu (SP):
   SP jest zwiększany i zmniejszany poprzez wszystkie jego stany oraz sprawdzany poprzez PC.
5. Test akumulatora A:
   Do A są zapisywane i odczytywane wszystkie możliwe wzorce danych. Może to być wykonane za pośrednictwem uprzednio przetestowanych rejestrów.
6. Test ALU i rejestru FLAG:
   Wykonywanie arytmetycznych i logicznych instrukcji. Operandy określane są bezpośrednio lub poprzez już sprawdzone rejestry. FLAGI są sprawdzane poprzez skoki warunkowe, których efekty są obserwowane za pośrednictwem PC.
7. Testowanie pozostałych rozkazów.

Ważnym założeniem w funkcjonalnym testowaniu mikroprocesora jest ustalenie czy **zbiór instrukcji** jest **ortogonalny**.
Ortogonalność pozwala na użycie każdej operacji w różnych trybach adresowania. Jeżeli zbiór instrukcji nie jest ortogonalny, to każda instrukcja musi być testowana dla jej wszystkich trybów adresacji. Ortogonalność pozwala na zmniejszenie liczby testów.
W funkcjonalnym testowania mikroprocesora stosuje się podejście znane pod nazwą *bootstrapingu* - w którym w kolejnych etapach testowania używa się komponentów sprawdzonych w etapach poprzednich.
Główny problem podejścia heurystycznego – to nieokreślona jakość uzyskanych testów funkcjonalnych. Doświadczenia pokazują, że testy opracowane metodami heurystycznymi zapewniają pokrycie 50-70% niezdatności.
Niekiedy stosuje się pewne miary heurystyczne do estymowania „kompletności" testu w odniesieniu do przepływu sterowania systemu. Miary te mogą być oparte na monitorowaniu aktywacji operacji w modelu RTL, np.:

$$\frac{\textit{liczba aktywowanych ścieżek (k)}}{\textit{liczba możliwych ścieżek (n)}}$$

1.2. Zastosowanie Binarnych Diagramów Decyzyjnych (BDD)

Można wykazać, że zbiór eksperymentów (testów) wyprowadzony za pomocą przebiegu wszystkich ścieżek, odpowiadających funkcjom wyjściowym, tworzą kompletne testy systemu (specyfikacje systemu).

2. Testowanie gruntowne i pseudogruntowne

Dla testowania gruntownego, zakłada się, że testy wykrywają wszystkie niezdatności z założonego, uniwersalnego zbioru niezdatności.

Uniwersalny zbiór niezdatności obejmuje dowolną niezdatność, która nie zmienia liczby stanów systemu cyfrowego (układu). Dla układu kombinacyjnego ***N*** realizującego funkcję ***Z(X)***, uniwersalny model niezdatności obejmuje dowolną niezdatność ***f***, która przekształca funkcję układu do postaci $Z_f(X)$.

Układy kombinacyjne

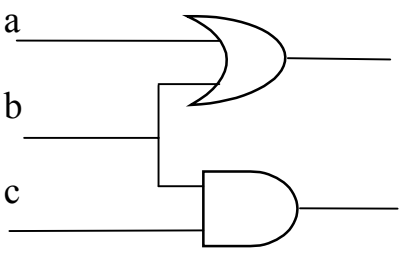
Do przetestowania wszystkich niezdatności definiowanych przez uniwersalny model niezdatności w układach kombinacyjnych o ***n*** - wejściach niezbędne jest zastosowanie wymuszenia pełnego (2^n wektorów). Ekspotencjany wzrost liczby wektorów ogranicza praktycznie testowanie gruntowne do układów posiadających nie więcej niż 20 linii wejściowych.

Metody testowania pseudogruntownego pozwalają na testowanie prawie wszystkich niezdatności ze zbioru uniwersalnego za pomocą liczby testów znacznie mniejszej niż 2^n:
- wykorzystanie układów o częściowej zależności,
- techniki segmentacji układów.

2.1. Wykorzystanie układów o częściowej zależności

Niech $O_1, O_2, \ldots, O_m$ oznaczają linie wyjściowe układu zawierającego n wejść, a n_i oznacza liczbę wejść wpływających na O_i. Układ, wktórym nie występują linie wyjściowe zależne od wszystkich wejść (tj. $n_i < n$ dla wszystkich O_i), jest układem o częściowej zależności. Dla takich układów pseudogruntowne testowanie wymaga 2^{n_i} testów dla każdej linii O_i.

Przykład



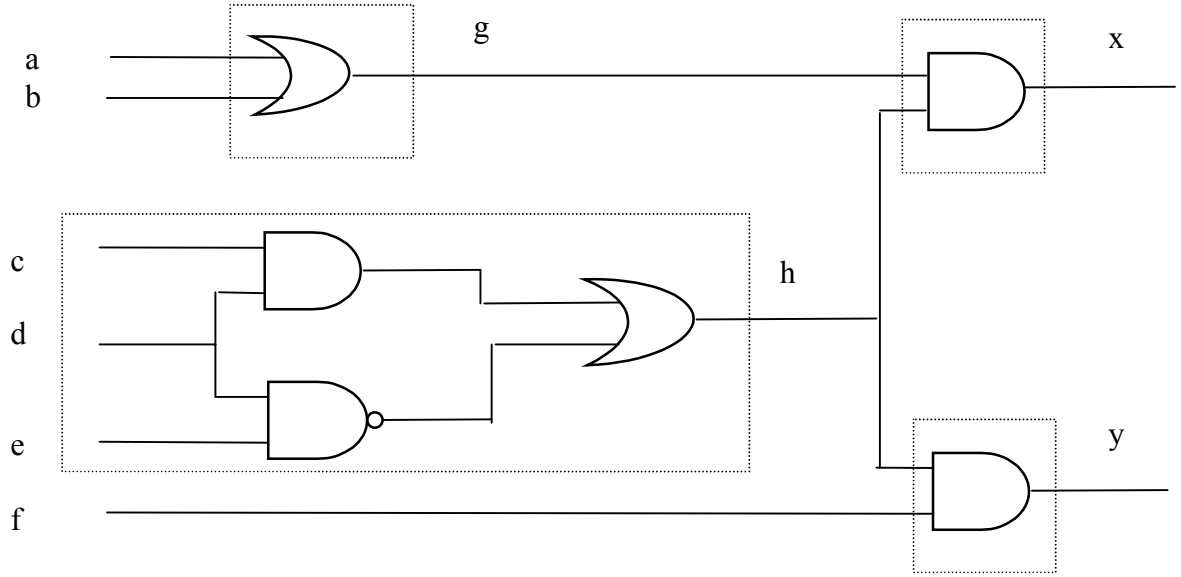
Zbiór testów :

a	b	c
0	0	0
0	1	0
1	0	1
1	1	1

Techniki segmentacji

Zasada podziału układu na segmenty:
liczba wejść każdego segmentu jest znacznie mniejsza niż liczba linii wejściowych układu.

Przykład.



Lp.	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**x**	**y**
1			0	0	0	1		1		
2			0	0	1	1		1		
3			0	1	0	1		1		
4			0	1	1	1	0	0		
5	0	0	1	0	0	1	0	1		
6	0	1	1	0	1	1	1	1		
7	1	0	1	1	0	1	1	1		
8	1	1	1	1	1	1	1	1		
9	0	1	0	1	1	0	1	0		
10		0	0	0	0	0		1		

Układy sekwencyjne

Dla układów sekwencyjnych, uniwersalny zbiór niezdatności obejmuje każdą niezdatność, która zniekształca tablicę stanów układu bez zmiany liczby stanów.

Wejściowa sekwencja wykrywająca każdą niezdatność należącą do tego modelu musi rozróżniać dany ***n*** - stanowy automat spośród wszystkich innych automatów skończonych o takiej samej liczbie wejść i wyjść oraz posiadający co najwyżej ***n*** – stanów.

Jakiego rzędu jest to problem ?

Twierdzenie [Moore 1956]

Dla każdego zredukowanego, spójnego automatu skończonego ***M*** istnieje para sekwencji wejściowej i wyjściowej, generowanej przez ***M***, która nie może być generowana przez żaden z automatów ***M'***, posiadający co najwyżej ***n*** – stanów.