ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1991
Seria: AUTOMATYKA z. 103 Nr kol. 1090

Mirosław CZAK
Andrzej MITAS
Dariusz POLOK
Zbigniew RYMARSKI

METODA TESTOWANIA MIKROPROCESORÓW Z WYKORZYSTANIEM PROCESÓW SYMULACYJNYCH. OPROGRAMOWANIE I SPRZĘT

Streszczenie. W artykule przedstawiono oryginalny sposób testowania mikroprocesorów oraz jego implementację sprzętową. Zastosowana metoda generowania testów pozwala na przejście z mnemonicznej postaci testu zapisanego w kodach programu asemblerowego do postaci sekwencji wektorów testujących podawanych na piny testowanego mirkoprocesora. Istotą metody jest naprzemienne powtarzanie dwóch procesów: symulacji pracy mikroprocesora za pomocą odpowiedniego programu oraz pobudzania wzorcowego mikroprocesora sekwencją otrzymaną w trakcie symulacji. Wyniki testowania są podstawą do kolejnego przebiegu symulacyjnego. Test może być użyty do szybkiego komparacyjnego testowania procesorów. Wszystkie opisywane procesy przebiegają w specjalnie zaprojektowanym jednorodnym środowisku programowym zorientowanym na testowanie.

Rozwój technologii elektronowej pozwala na tworzenie nowych jakościowo rozwiązań w zakresie elektroniki cyfrowej. Wraz z postępem technologicznym pojawiają się nowe zadania związane z problematyką uruchamiania i testowania opracowanych urządzeń.

Najistotniejszym elementem procesu testowania urządzeń cyfrowych jest generowanie sekwencji wektorów pobudzeń testujących. W przypadku układów wielkiej i bardzo wielkiej skali integracji jest to proces bardzo złożony i trudny w realizacji. Mikroprocesory szesnastobitowe, ze względu na rozbudowaną architekturę wewnętrzną, stanowią bardzo trudny obiekt dla konstrukcji testów. Dla automatu synchronicznego o bardzo dużej liczbie stanów wewnętrznych, jakim jest mikroprocesor, praktycznie nie można zastosować żadnej z popularnych metod systematycznego tworzenia pobudzeń testujących dla układów sekwencyjnych. Większość efektywnych metod testowania mikroprocesorów opiera się o analizę jego działania na poziomie

mikroprogramowania, a pobudzenia testujące stanowią program w kodzie maszynowym. Testy przedstawione w takiej postaci nie mogą być jednak stosowane w prostych testerach, w których pobudzenia podaje się w sposób sekwencyjny na wyprowadzenia testowanego układu, a uzyskane w wyniku testu odpowiedzi porównuje się z pamiętanym wzorcem. Istnieje jednakże sposób przekształcenia testów zapisanych w kodach maszynowych mikroprocesorów do postaci mikroprogramu testującego, polegający na zapamiętaniu, synchronicznie z zegarem taktującym, stanu wyprowadzeń mikroprocesora wzorcowego wykonującego program testujący. Podając na wejścia testowanego mikroprocesora otrzymane w wyniku powyższego procesu pobudzenia testujące, możemy porównać jego odpowiedzi z odpowiedziami zapamiętanymi w zbiorze wzorcowym. Podstawową trudnością tej metody jest stworzenie takiego zbioru.

Zastosowany w omawianej pracy tester układów scalonych MASTER-86 [10] posiada konstrukcję wewnętrzną umożliwiającą zapamiętywanie w autonomicznej pamięci testów przesyłanych do niego z zewnętrznego systemu mikroprosorowego. Istnieje także możliwość zapamiętania sekwencji odpowiedzi testowanego elementu na pobudzenia testujące. Cechy te umożliwiły stworzenie na drodze programowej symulatora zastępującego system mikroprocesorowy, w którym uruchamiane są programy testujące w celu określenia sekwencji wektorów testujących. Program ten, uruchomiony na komputerze nadzorującym pracę testera, wytwarza całą sekwencję sygnałów sterujących pracą mikroprocesora, łącznie z sygnałem zegarowym, w postaci mikroprogramu testującego podawanego w testerze na wyprowadzenia mikroprocesora wzorcowego. Odpowiedzi uzyskane w kolejnych krokach testu stanowią wraz z kodami maszynowymi programu testującego, podstawę dla symulatora do stworzenia kolejnego wektora sekwencji testującej podawanej na mikroprocesor wzorcowy. Za każdym razem po dopisaniu do sekwencji nowego wektora testującego, test jest wykonywany od początku, a po jego wykonaniu odpowiedzi są transmitowane do systemu nadzorującego pracę testera stanowiąc fragment sekwencji testującej. Proces ten jest powtarzany do momentu wykonania przez mikroprocesor wzorcowy ostatniej instrukcji programu testującego. W efekcie otrzymujemy gotową sekwencję wektorów testujących.

OPIS OPROGRAMOWANIA

W celu efektywnego wykorzystania opisanej metody generowania testu mikroprocesora wykonane zostało odpowiednie oprogramowanie zarządzające. Głównym jego zadaniem jest naprzemienne wykonywanie dwóch czynności:

inicjowanie programu symulacyjnego w celu wygenerowania kolejnej (kolejnych) linii pobudzeń oraz nadzorowanie samego procesu pobudzania mikroprocesora otrzymaną w ten sposób sekwencją wektorów testujących. W rezultacie powstaje nowy, powiększony, zbiór odpowiedzi, który jest znów przedmiotem analizy programu symulacyjnego. Ponadto oprogramowanie zarządzające musi spełniać dodatkowe funkcje, a mianowicie:

1) zapewnić poruszanie się po całym obszarze pamięci masowej komputera, tworzenie kartotek użytkownika, lokowanie w nich wygenerowanych cząstkowych i całościowych testów, wyszukiwanie ich i przemieszczanie pomiędzy kartotekami;

2) zapewnić wygodną edycję programów testujących w kodach mnemonicznych;

3) umożliwić wizualizację - w dowolnym momencie - wygenerowanego zbioru testowego, zarówno w postaci tabel logiki trójwartościowej - L,H,Z, jak i przebiegów czasowych;

4) pozwolić na "ręczną" modyfikację zbioru testującego, w celu stworzenia bardziej krytycznych warunków testowania, badania odporności testowanego układu na pojawianie się na jego pinach nietypowych poziomów logicznych, wymuszanie cykli jałowych itp.;

5) organizować szybkie testowanie komparacyjne w wykorzystaniem zbioru pobudzeń wygenerowanych opisaną metodą przy użyciu mikroprocesora a priori uznanego za sprawny, a więc pobudzanie badanego mikroprocesora wygenerowaną sekwencją i porównanie poziomów logicznych na pinach procesora wzorcowego i badanego;

6) porównywać zbiory testujące generowane za pomocą metody symulacyjnej dla rozmaitych procesorów;

Mnogość i różnorodność tych zadań była powodem stworzenia oprogramowania w formie jednorodnego środowiska zorientowanego na testowanie, spełniającego niektóre funkcje systemu operacyjnego oraz wymienione powyżej czynności specjalizowanego oprogramowania testującego [11].

Oprogramowanie zaprojektowano w formie zewnętrznie przypominającej zintegrowane środowisko programowe Turbo - Pascala nowszych wersji. Wybieranie odpowiednich opcji programu umożliwia system rozwijanych menu zgrupowanych w okienka łączące zbliżone funkcje. Wydzielono pięć zasadniczych grup procedur:

I. **ZBIORY** - są to procedury spełniające niektóre funkcje systemu operacyjnego, takie jak listowanie zawartości kartotek, tworzenie i usuwanie kartotek, a także czytanie z dysku zbiorów testujących, zapis ich na dysk z możliwością zmiany nazwy, co pozwala na niejawne kopiowanie zbiorów, a także na uchronienie przed zniszczeniem dawniejszych wersji testów. W grupie tej umieszczono też dwie procedury pozwalające zakończyć pracę programu. Pierwsza

przekazuje sterowanie do systemu operacyjnego pozostawiając program w pamięc komputera (OS Shell) i umożliwia szybki powrót do programu po naciśnięci klawiszy [CTRL] + [ALT], druga procedura kończy działanie programu i zwalni całą pamięć do dyspozycji systemu operacyjnego:

II. **EDYTOR GRAFICZNY** - ten moduł programowy pozwala na wizualizacj dowolnego ciągu binarnego w postaci przebiegów czasowych, grupowani testowanych wyprowadzeń w magistrale i wyświetlanie ich stanu w postac heksadecymalnej, "ręczne" wprowadzanie zmian do zbiorów pobudzeń, a nawe stworzenie całkowicie nowego testu, np. na podstawie katalogu. Optymaln wykorzystanie możliwości edytora graficznego polega na odczytaniu test wygenerowanego w trybie automatycznym i zmodyfikowniu go tak, aby warunk testowania badanego procesora uczynić możliwie najbardziej krytycznymi, t znaczy przesunąć zbocza przebiegów czasowych do wartości granicznyc podawanych przez producenta, wprowadzenie takich poziomów logicznych n wyprowadzeniach testowanego procesora, które nie wystąpią w trybie prac automatycznej, a które nie powinny zakłócić pracy normalnie działająceg mikroprocesora. Za pomocą edytora graficznego można też dopisać fragment testu, które w trybie automatycznym nie mogą być wygenerowane.

III. **EDYTOR/ASEMBLER** - moduł programowy, rozpoczynający właściwy proce generowania testu mikroprocesora. Za pomocą specjalizowanego wbudowaneg edytora można napisać test procesora w rozkazach mnemonicznych, a następni poddać go asemblacji używając napisanego do tego celu programu asemblera. Szczegółowy opis tego procesu opisany jest w rozdziale następnym. Progra pozwala korzystać także z pełnoekranowego edytora tekstowego Norton Edito (NE), który został wbudowany w system. Zbiory utworzone za pomocą NE, jak dowolnego edytora tekstowego mogą być poddawane opracowaniu przez progra hard-asemblera tak samo, jak programy napisane pod kontrolą specjalizowaneg edytora.

IV. **TEST PROCESORA** - opcja automatycznego generowania test mikroprocesora. Danymi wejściowymi do generowania testu jest tablica kodów tablica adresów generowane przez asembler. Działanie programu zarządzająceg polega w tym przypadku na naprzemiennym inicjowaniu dwóch procedur: symulacyjnej i testującej, jak to opisano powyżej. Proces ten powtarzany jest tak długo, aż wystąpi jeden z warunków kończących program:

- wyczerpane zostaną tablice adresów i kodów - wygenerowany zostanie cały test i program zakończy się prawidłowo;
- długość testu przekroczy 8 K wektorów pobudzeń i proces tworzenia testu nie może być kontynuowany ze względu na techniczne ograniczenia konstrukcj testera;

- program symulatora nie będzie w stanie zinterpretować wyników testu - procesor używany do generowania testu może być niesprawny;
- nastąpi błąd w układach testera, jego fizyczne odłączenie od komputera, niestabilność wyników testu lub inny błąd sprzętowy.

Przebieg pracy programu odzwierciedlany jest w słowie statusu testu, kontrolowanym zarówno przez program nadzorujący, jak i symulator. Aby zapewnić maksymalną autonomiczność programu symulatora napisano go jako rezydentą procedurę obsługującą przerwanie generowane przez program nadzorujący. W ten sposób symulator może korzystać z własnego segmentu danych, co niemożliwe jest przy wykorzystaniu typowych mechanizmów Turbo Pascala.

Po utworzeniu fragmentu testu można zachować wygenerowany zbiór i traktować go jako dane wejściowe przy inicjacji nowego procesu symulacyjnego, co ma znaczenie przy interaktywnym tworzeniu testów i bieżącej ocenie ich właściwości detekcyjnych. Proces symulacyjny może też zostać zainicjowany przy zerowych warunkach początkowych.

V. KOMPRESJA - wyniki testów w niektórych przypadkach korzystniej jest przechowywać i analizować w postaci skróconej, przy zachowaniu prawie stuprocentowej jednoznaczności odwzorowania zbioru odpowiedzi na zbiór poddany kompresji. W tym celu w środowisko programowe wbudowany został mechanizm opracowania wyników i przekształcania ich do zbioru ciągów 40-bitowych - niezależnie od długości testu. Wykorzystano zarówno liniowe, jak i nieliniowe techniki kompresji, a wybór metody dokonywny jest w sposób optymalny z uwzględnieniem cech opracowanego ciągu.

HARD-ASSEMBLER: SPECJALIZOWANY ASSEMBLER ZORIENTOWANY NA TESTOWANIE MIKROPROCESORÓW INTEL 8086/88

Pierwszym, "wejściowym" ogniwem systemu testowania procesów INTEL 8086/8088 jest program specjalizowanego asemblera zorientowanego na testowanie tych procesorów za pomocą testera MASTER-ITE, nazwany z racji swoich specyficznych własności hard-asemblerem.

"Wejściem" tego programu jest zbiór tekstowy zawierający mnemoniki instrukcji procesora i dyrektywy hard-asemblera, a wyjściem dwa zbiory robocze: pierwszy złożony z bajtów programu poddanego działaniu hard-asemblera, a drugi złożony z czterobajtowych podwójnych słów zawierających 20-bajtowe adresy bezwzględne odpowiadających im kolejnych bajtów kodu zawartego w pierwszym zbiorze oraz informację o dyrektywach

wykonawczych (ang. hard-directives) w pozostałych bitach na starszych pozycjach. W trakcie testowania procesora, bajty kodu z pierwszego zbioru roboczego są podawane na magistralę danych procesora, a odpowiadające im adresy na magistralę adresową w odpowiednich taktach zegarowych, co przedstawiono w części publikacji dotyczącej generatora sekwencji testującej. Taka organizacja programu - tzn. sztywne przypisanie bezwzględnych adresów do poszczególnych bajtów kodu umożliwia pisanie programów wykorzystujących pełną przestrzeń adresową 1 MB procesora 8086/8088. Drugą szczególną cechą hard-asemblera, odróżniającą go od innych znanych asemblerów, jest możliwość używania dyrektyw wykonawczych powodujących sprzętową generację sygnałów podawanych na piny procesora (17,18,21,22,23,31,33). Dyrektywy wykonawcze posiadają własne parametry zapisywane w trzecim zbiorze roboczym. Przykładowo po rozkazie WAIT umieszczonym w programie cykl rozkazowy procesora zostaje zawieszony aż do pojawienia się sygnału $\overline{TEST}$=0 (pin 23). Wyróżnia się dwa parametrem dyrektywy wykonawczej NTEST: liczbę taktów zegara procesora po wystawieniu na magistralę danych procesora ostatneigo bajtu kodu, po których zostaje wygenerowany sygnał $\overline{TEST}$=0, oraz liczbę taktów zegara procesora, w czasie których sygnał TEST musi być utrzymany na niskim poziomie [1,2], parametrem dyrektywy wykonawczej INT jest dodatkowo numer przerwania itp. Informację o dyrektywach wykonawczych zakodowano w najstarszych bitach podwójnych słów drugiego zbioru roboczego.

Wiadomo, że pełną 1 MB przestrzeń adresową można adresować dwudziestobitowymi słowami. Zatem dekodowanie dyrektyw wykonawczych specjalnych polega na sprawdzeniu, czy podwójne słowo zbioru roboczego adresów zawiera liczbę większą od $fffff (największa wartość, którą można określić dwudziestoma bitami). Jeżeli tak, to należy wykorzystać do adresacji młodsze 20 bitów, a starsze bity, po przesunięciu w prawo o 20 bitów podwójnego słowa, wskażą bezpośrednio na numer identyfikujący typ sygnału generowanego sprzętowo po wykonaniu rozkazu, którego ostatni bajt jest adresowany podwójnym słowem zbioru roboczego adresów z informacją o dyrektywie wykonawczej.

Zbiory typu "object" z istniejących makroasemblerów [3,4,5] są w pełni relokowalne, co z punktu widzenia hard-asemblera powoduje kłopot z przypisaniem kodów do sztywnych, bezwzględnych adresów.

Drugim problemem jest kodowanie dyrektyw wykonawczych, np. pod postacią komentarza i dekodowanie ich w trzecim dodatkowym przebiegu asemblera.

Trzeci problem przedstawiono za pomocą następującego przykładowego fragmentu programu:
wystąpił rozkaz skoku do pewnego rozkazu, a następnie dyrektywa ORG, ustawiająca licznik rozkazów na adres pierwszego bajtu tego rozkazu. Standardowy zbiór typu "object" będzie zawierać wszystkie niewykorzystane bajty programu pomiędzy ostatnim bajtem rozkazu skoku, a bajtem na którego adres wskazuje argument dyrektywy ORG, natomiast zbiór roboczy kodów hard-assemblera zawierać ich nie będzie.
Omówione zagadnienia powodują, że przeróbki istniejących makroasemblerów i próby wykorzystania ich do tworzenia zbiorów roboczych wydają się bardziej pracochłonne, niż stworzenie od nowa programu hard-asemblera.
Specjalizowany program hard-asemblera posiada własny, prosty edytor tekstowy umożliwiający pisanie programu źródłowego, poprawianie linii programu, wstawianie dodatkowych linii do utworzonego już programu i usuwanie istniejących linii. Edycję programu kończy się dyrektywą END, która jednak niczego do programu nie wnosi i w programach źródłowych pisanych pod innym edytorem jest zbędna. Program jest przerywany dyrektywą BYE, której działanie opisano w części publikacji dotyczącej generatora sekwencji testującej. Każda instrukcja musi być pisana w osobnej linii i zakończona znakiem "ENTER" (kodem ODh). Linia pusta jest dekodowana jako błąd składniowy. Nadmiarowe spacje (oprócz niezbędnych) są w trakcie asemblacji pomijane. Własny edytor automatycznie zamienia małe litery na duże, którymi są wpisane wzorce mnemonik w odpowiednich tablicach asemblera.
Program asemblera może wywołać program NORTON EDITOR (NE) i użytkownik może pisać program źródłowy za pomocą tego popularnego edytora tekstowego. Po wstępnym zapisaniu na dysku, program testu zostaje ponownie odczytany, małe litery zamienione na duże i tak zmodyfikowany program ostatecznie zapisany.

Hard-asembler może również odczytywać programy pisane przy użyciu innych edytorów tekstu. Pierwsze menu hard-asemblera pozwala na wybór edycji za pomocą własnego edytora, użycie NE lub odczyt z dysku istniejącego już zbioru tekstowego. Z każdej z tych opcji w przypadku bezbłędnego ich wykonania przechodzi się do drugiego menu. Wykrycie błędu powoduje powrót do pierwszego menu. Drugie menu zawiera wymienione już opcje własnego edytora, opcję zapisu zbioru źródłowego na dysk pod nazwą wybraną przez operatora (istnieje nazwa domyślna przekazywana jako parametr przy wywoływaniu programu hard-asemblera), opcję uruchomienia programu hard-asemblera i opcję zapisu na dysk zbiorów roboczych (kodu, adresów i parametrów kolejnych dyrektyw wykonawczych) do zbiorów o nazwach określonych przez program. Powrót do pierwszego menu następuje po wciśnięciu klawisza "ESC", co równocześnie

powoduje zwolnienie wszystkich zmiennych dynamicznych, a w konsekwencji usunięcie z RAM utworzonego bądź odczytanego zbioru źródłowego. Wynik asemblacji w postaci tablic kodów, adresów i parametrów dyrektyw wykonawczych pozostaje niezmieniony aż do drugiego przebiegu hard-asemblera lub wyjścia z programu - przekazanie sterowania (za pomocą klawisza "ESC") do programu zarządzającego. Hard-asembler skonstruowano jako asembler dwuprzebiegowy [6,7]. W pierwszym przebiegu rozszyfrowane zostają kody mnemoniczne rozkazów, ustalone adresy pierwszych bajtów tych rozkazów na podstawie ustalonych liczb bajtów poprzedzających je rozkazów i rozszyfrowanie występujących dyrektyw ORG) oraz utworzona zostaje tablica nazw symbolicznych (etykiet, stałych i nazw zmiennych), tablica związanych z nimi ilości bitów (0: dla etykiet i stałych, 1,2,4: dla danych, inna wartość spoza zbioru [0,1,2,4] np 100 przy braku deklaracji nazwy symbolicznej występującej w programie) oraz tablica wartości dla danych (ponieważ są one reprezentowane przez adres, liczbę zajmowanych bitów i wartość). Etykiety mogą być typu "near" (adres umieszczony w 2 bajtach - wykorzystany w większości instrukcji jako przesunięcie (ang. displacement) do poruszania się wewnątrz segmentu zadeklarowanego w odpowiednim rejestrze segmentowym), lub typu "far" (adres umożliwiający poruszanie się po całej przestrzeni adresowej - standardowo [3,4,5] umieszcza się go w 4 bajtach). W pierwszym przebiegu interaktywnie (z wprowadzeniem na bieżąco poprawek w trakcie pracy programu) sprawdzana jest zarówno poprawność składni mnemonik instrukcji, wartość adresów bajtów instrukcji (czy nie wykraczają poza $fffff) jak i wartość operandów liczbowych, itp. - sprawdzane są wszystkie informacje uzyskane z asemblowanej aktualnie mnemoniki instrukcji i ją poprzedzających instrukcji oprócz podstawienia uzyskiwanych dopiero w pierwszym przebiegu wartości liczbowych powiązanych z nazwami symbolicznymi. Wartości liczbowe (adresy lub operandy natychmiastowe) muszą być deklarowane w kodzie heksadecymalnym poprzedzonym znakiem "$". Znak "$", po którym jest przynajmniej jedna spacja, lub znaki "+", "-" w dyrektywie ORG, oznacza bieżącą wartość licznika rozkazów (czyli adres ostatniego bajtu kodu plus 1). Deklaracja nazwy symbolicznej wymaga znaku ":" po etykiecie (nie licząc spacji), znaku "=" po stałej, łańcucha "DB" lub "DD" po nazwie stałej. W instrukcjach, w których istnieje rozróżnienie operandu jako adresu pamięci lub operandu natychmiastowego [2] - np. operand będący źródłem (ang. source operand) w instrukcji MOV, adres pamięci występuje w nawiasach kawadratowych "[...]", a operand natychmiastowy bez nich. W innym przypadku, gdy operandem instrukcji może być jedynie adres pamięci, operand może być w nawiasach lub nie. Końcowym wynikiem pierwszego przebiegu asemblera jest

kolejka rekordów składających się ze wstępnego kodu rozkazu (tablica trzynastu liczb całkowitych stanowiących numeryczny wynik deszyfracji mnemoniki), wartość ewentualnie występującego operandu liczbowego oraz adres pierwszego bajtu rozkazu, a także opisane już tablice określające użyte nazwy symboliczne oraz tablica parametrów kolejnych dyrektyw wykonawczych. Na pierwszej pozycji wstępnego kodu rozkazu znajduje się numer instrukcji (zgodnie z tablicą pierwszych słów mnemonik hard-asemblera), na drugiej - liczba bajtów rozkazu. Pozycje 5,6,7,8 dotyczą pola pierwszego operandu (przed przecinkiem), pozycje 9,10,11,12 dotyczą pola drugiego operandu (po przecinku). Na pozycji 5 lub 9: 0-oznacza brak nazwy symbolicznej, n < 0-oznacza numer nazwy symbolicznej występującej jako adres pamięci w nawiasach "[...]", n > 0-oznacza n-ty numer nazwy symbolicznej występującej bez nawiasów. Na pozycji 6 lub 10: -2 oznacza występowanie liczbowego operandu jako adresu pamięci w nawiasach "[...]", -1 oznacza występowanie liczbowego operandu bez nawiasów, 0 - brak nazwy symbolicznej lub operatora liczbowego, 1 - występowanie nazwy symbolicznej o dwubajtowym bliskim adresie (ang. near), 2 - występowanie nazwy symbolicznej o dalekim adresie (ang. far) z czterema bajtami zarezerwowanymi na adres. Na pozycji 7 lub 11: 0 - oznacza brak rejestru indeksowego lub bazowego w pierwszym nawiasie, n > 0 - oznacza numer takiego rejestru z odpowiedniej tablicy w pierwszych nawiasach. Pozycje 8 lub 11 dotyczą odpowiednio tych rejestrów w drugich nawiasach. Numer na pozycji 13 identyfikuje wersję rozkazu o numerze z pozycji 1. Pozycje 3 i 4 zmieniają znaczenie zależnie od typu rozkazu.

Tak zdefiniowany wstępny kod rozkazu zawiera wprawdzie nadmiarową informację, lecz dzięki temu umożliwia napisanie krótszego i szybciej wykonywalnego programu drugiego przebiegu asemblacji.

Błąd w trakcie pierwszego przebiegu asemblacji powoduje żądanie przez program napisania poprawnego symbolu mnemonicznego. Komunikat na monitorze wskazuje na numer błędnej instrukcji i typ błędu. Program wstawia ponownie napisaną instrukcję na miejsce błędnej i rozpoczyna automatycznie pracę od początku. Po pozytywnym przejściu przez pierwszy przebieg asemblacji wyświetlona zostaje kontrolnie zawartość opisanej uprzednio kolejki rekordów, a następnie operator z klawiatury uruchamia drugi przebieg asemblacji. W drugim przebiegu asemblacji ze wstępnych kodów rozkazów, wartości operandów liczbowych i adresów pierwszych bajtów rozkazów oraz trzech opisanych tablic identyfikujących nazwy symboliczne tworzy się bajty kodu maszynowego, zgodnie z [1,2,3,4,5,7,8,9], sprawdzając czy wszystkie występujące w rozkazach nazwy

symboliczne zostały zadeklarowane (czy w tablicy liczby bajtów pod indeksem nazwy jest liczba 0,1,2 lub 4, a nie umowna liczba 100 oznaczająca brak deklaracji etykiety) oraz sprawdzana jest prawidłowość deklaracji nazwy symbolicznej dla danego typu rozkazu (liczba bajtów adresu pamięci lub bajtów zmiennej, długość skoku, wartość adresu ustawianego dyrektywą ORG $+nazwa symboliczna itp.). W trakcie drugiego przebiegu asemblacji wpisywane są wartości do tablicy kodów i adresów.

Błąd wykryty w drugim przebiegu asemblacji przerywa ją, a wyświetlany komunikat wskazuje na numer błędnej instrukcji i rodzaj błędu. Wciśnięcie dowolnego klawisza powoduje przejście do drugiego menu, gdzie znajdują się opcje poprawy programu źródłowego, a ewentualną ponowną asemblację trzeba rozpocząć od pierwszego przebiegu.

Bezbłędne przejście drugiego przebiegu jest sygnalizowane wyświetleniem na ekranie wpisanych w trakcie asemblacji elementów tablicy kodów i adresów z automatyczną sygnalizacją miejsca w programie testu, w którym nastąpi generacja umieszczonej w programie dyrektywy wykonawczej. Po wyświetleniu ciągu adresów i kodów wciśnięcie dowolnego klawisza powoduje wyświetlenie drugiego menu.

Ostatnią czynnością przed przejściem do dalszej części programu testowania procesora jest zapisanie tablic kodów i adresów oraz tablicy parametrów dyrektyw wykonawczych na dysk w postaci zbiorów roboczych.

PROGRAM SYMULATORA MIKROPROCESORÓW INTEL 8086/88

Opisany powyżej program hard-asemblera przygotowuje dane do symulacji urządzeń zewnętrznych za pomocą drugiego programu SIM, który jest generatorem mikroprogramu wymuszeń testujących dla mikroprocesorów MCY 78C86/88. Wersja 1.0 programu umożliwia generowanie wymuszeń testujących dla procesorów pracujących w trybie maksymalnym.

Programowy symulator SIM służy do generacji mikroprogramu wymuszeń testujących dla mikroprocesorów MCY 78C86/88. Program ten realizuje następujące zadania:

1. Generowanie sekwencji zerującej procesor.
2. Obserwacja słowa stanu procesora w celu określenia cykli jego dostępu do urządzeń zewnętrznych.
3. Symulacja urządzeń zewnętrznych.
4. Generowanie sygnałów zewnętrznych dla procesora zgodnie z programem testu.

Symulator jest przeznaczony do generowania mikroprogramu testującego zgodnie z metodą LIT. Po wygenerowaniu nowej linii mikroprogramu testującego test jest wykonywany od początku. Odpowiedzi zebrane w ostatnim kroku testu stanowią podstawę do wygenerowania nowego słowa pobudzeń testujących. Program uwzględnia na ich podstawie konieczność wygenerowania odpowiednich sygnałów sterujących, podania danych na magistralę procesora, a także konieczność zmiany poziomu na linii zegara mikroprocesora.

Zmienne w rekordzie zajmujące 4 bajty stanowią tzw. długie adresy w postaci dwóch bajtów przesunięcia (ang. displacement) oraz dwóch bajtów adresu segmentu. Dane te reprezentują podwójne słowo.

Tablica adresów RAM zawiera listę adresów wszystkich komórek pamięci, w których będzie pamiętana informacja podczas wykonywania testu. Lista ta tworzona jest podczas pracy programu hard-asemblera. Każdy adres jest pamiętany jako 20-bitowy adres bezwzględny. Struktura słowa listy adresów została omówiona w opisie programu asemblera.

Zmienna "Długość obszaru RAM" określa wielkość zadeklarowanego obszaru pamięci (w bajtach).

Tablica zawartości RAM posiada strukturę bajtową. Są w niej zapisane zarówno program jak i dane dla testu procesora. Aby uzyskać dostęp do bajtu pod określonym adresem, symulator przeszukuje listę adresów zawartą w tablicy adresów w celu wyznaczenia przesunięcia elementu listy, odpowiadającego szukanemu adresowi, od początku listy. Przesunięcie to wskazuje na pozycję bajtu w tablicy zawartości RAM. W przypadku gdy lista adresów nie zawiera poszukiwanego wzorca symulator uznaje, że pamięć o danym adresie nie istnieje. Oznacza to, że w przypadku próby zapisu informacji przez procesor do danej komórki pamięci jest ona tracona, natomiast informacja odczytana będzie równa Offh.

Tablica pobudzeń powstaje w wyniku działania programu symulatora. Test to dynamicznie tworzony mikroprogram testujący w standardowym formacie testera MASTER-86. Długość testu jest określona przez zmienną "Liczba linii testu". Określa ona długość mikroprogramu testującego w bajtach.

Dwie tablice zawierające odpowiedzi na test zawierają informację uzyskaną na wyprowadzeniach testowanego mikroprocesora w wyniku wykonania testu. Informacja ta jest kodowana na dwóch bitach w celu odróżnienia stanu HZ od stanów L i H.

Zmienna "Wskaźnik błędu" jest słowem statusu programu opisanym w punkcie poprzednim. Tablica sygnałów sterujących opisuje sekwencję użycia sygnałów sterujących pracą mikroprocesora. Tablica ta jest tworzona podczas asemblacji testu programem hard-asemblera. Sygnały, które mogą być użyte to: INT, NMI,

RQØ, RQ1, $\overline{TEST}$, MN/$\overline{MX}$, tablica ta podzielona jest na 6 sekcji, z którycl każda opisuje użycie jednego z sygnałów. Każda pozycja tablicy zawiera trz parametry: rodzaj sygnału,. czas w taktach zegarowych od momentu pobrani rozkazu, w którym występuje znacznik użycia sygnału zewnętrznego do moment jego ustawienia, czas trwania sygnału. Znacznik sygnału jest ustawiany najstarszym bajcie elementu listy adresów.

SPOSÓB PRZEPROWADZENIA SYMULACJI

Sygnałem rozpoczęcia symulacji w kolejnym kroku jest wykonani programowego przerwania 61h. Jeżeli symulacja ma być zainicjowana o początku, zewnętrzny program wywołujący symulator powinien zapisać wartość do zmiennej "Długość testu". Przed wywołaniem symulatora powinny być takż ustalone adresy poszczególnych obszarów danych, które zostały omówion powyżej. Wywołanie symulatora z ustawioną na 0 zmienną "Długość testu spowoduje, że wynikiem pracy symulatora będzie 20 słów mikroprogram opisujących sekwencję zerowania testowanego mikroprocesora. Następn wywołania każdorazowo będą tworzyły od 1 do dwóch słów mikroprogramu. Dw słowa będą tworzone w przypadku konieczności zmiany konfiguracji wyprowadze magistrali testowanego procesora. Po zakończeniu kroku symulacji symulato zwraca sterowanie do programu obsługi testera. Program ten powinien wykona test za pomocą testera MASTER-ITE oraz odczytać odpowiedzi mikroprocesora n test uzyskane w wyniku przeprowadzenia testu i zapisane w pamięci testera Długość testu jest określona przez zmienną o tej samej nazwie. Odczytan odpowiedzi stanowią podstawę do określenia kolejnych linii mikroprogram testującego podczas następnego wywołania symulatora.

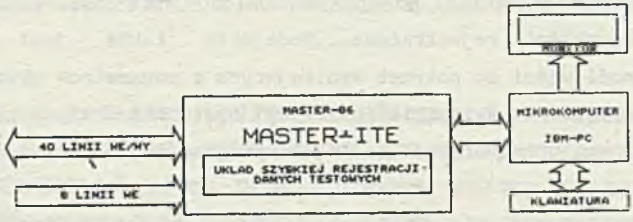
Na jeden takt zegara testowanego mikroprocesora przypadają 4 słow mikroprogramu.

OPIS IMPLEMENTACJI SPRZĘTOWEJ

W celu zaimplementowania opisywanej metody konieczny był układ tester zapewniający sekwencyjne pobudzanie mikroprocesora i rejestrację odpowiedz na tyle szybką, aby zapewnić pracę mikroprocesora w czasie rzeczywistym Rozwiązania takie znane są w świecie, jednak koszt ich jest znaczny zwłaszcza, że są to urządzenia unikalne o niezywkle wysokim poziomi technicznym i technologicznym. Dowodem tego są ceny testerów w krajac zachodnich.

Próbą wyjścia naprzeciw tym problemom jest poszukiwanie przez znane firmy rozwiązań o charakterze "cost-effective". Przykładem tego jest tester Wayne Kerr serii 900. Producent zapowiada, że urządzenie to pozwala osiągnąć 80% możliwości rozbudowanego systemu ATE nakładem 20% kosztów takiego systemu. Mimo, że wspomniany tester nie ma takich możliwości, jak np. system HP3065AT, to jednak jego popularność wydaje się być uzasadniona szerokimi możliwościami jego zastosowania w procesie produkcyjnym.

Wzorując się na nowoczesnych, przodujących firmach podejmuje się próby rozwiązania poszczególnych zadań w dziedzinie testowania także w warunkach krajowych. Przykładem tego jest stanowisko testowo-uruchomieniowe MASTER-ITE, opracowane na zlecenie Instytutu Technologii Elektronowej w Warszawie. Stosunkowo niewielkim nakładem kosztów uzyskano interesujące wyniki. Bazą konstrukcyjną testera MASTER-ITE jest opracowany w ramach CPBR 8.7 w Instytucie Elektroniki Politechniki Śląskiej tester MASTER-86. Rozwiązanie konstrukcyjne ilustruje załączony schemat blokowy - rys. 1.



Rys. 1. Schemat blokowy stanowiska testowo-uruchomieniowego MASTER-ITE
Fig. 1. Block - schema of test - developing system MASTER-ITE

Podstawowym rozszerzeniem, stwarzającym nowe możliwości, jest układ szybkiej rejestracji danych, stanowiących rezultaty pobudzania obserwowanego układu. Zasada działania prezentowanego stanowiska MASTER-ITE polega na równoległym zadawaniu wielobitowego pobudzania i rejestracji słowa odpowiedzi w każdym kroku testowym układu. Swą uniwersalność stanowisko testowo-uruchomieniowe zawdzięcza temu, że każde wyprowadzenie może być traktowane indywidualnie w danym kroku testowym. Spośród 40 równoległych końcówek każda może być wejściem albo wyjściem. Jeśli wyprowadzenie ustawione jest w stan wyjścia (tzn. jest końcówką pobudzającą badany układ) wówczas w danym kroku testowym przyjmuje ono stan binarny zadeklarowany odpowiednio w zbiorze

pobudzeń. Zmiana wartości binarnych na wyprowadzeniach pobudzających wymaga podania kolejnego słowa pobudzeń. W przypadku natomiast, gdy zamiarem jest zmiana charakteru wyprowadzeń dynamicznie w trakcie testu, w zbiorze testowym powinno się znajdować nowe, aktualne tzw. słowo konfiguracji, ustalające kierunki przepływu sygnału na poszczególnych końcówkach. Kierunki te obowiązują od chwili pojawienia się pierwszego słowa pobudzeń po aktualnym słowie konfiguracji.

Tester został wyposażony w osiem dodatkowych wejść równoległych. Rozszerzenie to pozwala zwiększyć pole obserwacji do 48 wyprowadzeń. Takie podejście uzasadnione było koniecznością podłączenia dodatkowych linii pomiarowych.

Tak więc, za pomocą odpowiedniego oprogramowania użytkownik jest w stanie pobudzić badany układ do realizacji dowolnych funkcji, przy założeniu, że jest to układ cyfrowy, o poziomach logicznych kompatybilnych z TTL.

Obserwacja reakcji badanego układu jest możliwa dzięki zastosowaniu szybkiego układu rejestrującego wyjściowe stany logiczne, łącznie ze stanem wysokiej impedancji. Rozwiązanie to przypomina swą konstrukcją analizator stanów logicznych. Różnica polega zasadniczo na znacznym ograniczeniu maksymalnej szybkości rejestratora. Podejście takie jest podyktowane dopasowaniem możliwości do potrzeb wynikających z parametrów układu zadawania pobudzeń. Skompletowane w trakcie realizacji testu rezultaty mogą być poddane dalszemu dowolnemu przetwarzaniu na drodze programowej.

LITERATURA

[1] Misiurewicz P.: Układy mikroprocesorowe, WNT, Warszawa 1983.

[2] The 8086 Family User's Manual, Santa Clara, Calif., Intel Corporation 1979.

[3] Microsoft, Macro Assembler, User Guide, Microsoft Corporation 1984.

[4] Microsoft, Macro Assembler, Reference Manual, Microsoft Corporation 1984

[5] Macro Assembler by Microsoft, Microsoft Corporation 1981.

[6] Tannenbaum Andrew S.: Organizacja maszyn cyfrowych w ujęciu strukturalnym WNT, Warszawa 1980.

[7] Waite William M., Goos Gerard: Konstrukcja kompilatorów, WNT, Warszawa 1989.

[8] Systemy mikroprocesorowe, zeszyt 20, Przemysłowy Instytut Elektroniki, Warszawa 1986.

[9] Liu Yu-Cheng, Gibson Glenn A.: Microcomputer Systems: the 8086/8088 Family. Architecture, Programming, and Design, Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632 1984.

[10] Mitas A., Hławiczka A., Polok D.: Automatyczny tester cyfrowych układów scalonych MASTER-86, Pomiary, automatyka, kontrola, VI., 1987.

[11] Polok D.: The STAR system: Integrated Software Environment for Functional Test, Proc. XIII Fault Tolerant Systems and Diagnostics Conference, Varna 1990.

Recenzent: Prof. dr hab. Andrzej Kobus

Wpłynęło do Redakcji 1.03.1990 r.

МЕТОД ТЕСТИРОВАНИЯ МИКРОПРОЦЕССОРОВ С ПРИМЕНЕНИЕМ ПРОЦЕССОВ МОДЕЛИРОВАНИЯ. ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ И ОБОРУДОВАНИЕ

Р е з ю м е

В статье представлен оригинальный способ тестирования микропроцессоров, а также его применение для конкретных аппаратных средств. Примененный метод генерирования тестов позволяет преобразовывать мнемоническое состояние теста, записанного в кодах ассемблированной программы, в состояние последовательности тестируемых векторов, подаваемых на выводы тестируемого микропроцессора. Сущность метода заключается в чередовательном повторении двух процессов: модуляции работы микропроцессора с помощью соответствующей программы и возбуждения образцового микропроцессора последовательностью полученной во время моделирования. Результаты тестирования являются основой для очередного процесс моделирования. Тест может быть применен для быстрого компаративного тестирования процессоров. Все рассмотренные процесс проходят в специально запроектированной однородной програмной среде сориентированной на тестирование.

METHOD OF TESTING OF MICROPROCESSORS USING SIMULTATION PROCESSES. SOFTWARE AND HARDWARE

Summary

A new unique approach to microprocessors testing and its hardware implementation are presented in this paper. The used test pattern generation method allows to convert test programs, written as mnemonical procedures of microprocessor instructions, into the set of testing vectors driving the inputs of tested processor. The general idea of the presented method is iterative repetition of two processes: simulation of microprocessor by means of appropriate software and executing the test of reference processor using the test sequence obtained in simulation. The results of testing create data set to the next simulation pass. Test obtained in the such process, may be used to quick comparative testing of microprocessors. All the described processes are executed inside the integrated test-oriented software environment.