Instytut Informatyki Politechniki Śląskiej

Projekt PK3

Rok akademicki:	Rodzaj studiów:	Temat:	Grupa:	Sekcja:
2018/2019	SSI	Maszyna W	6	1

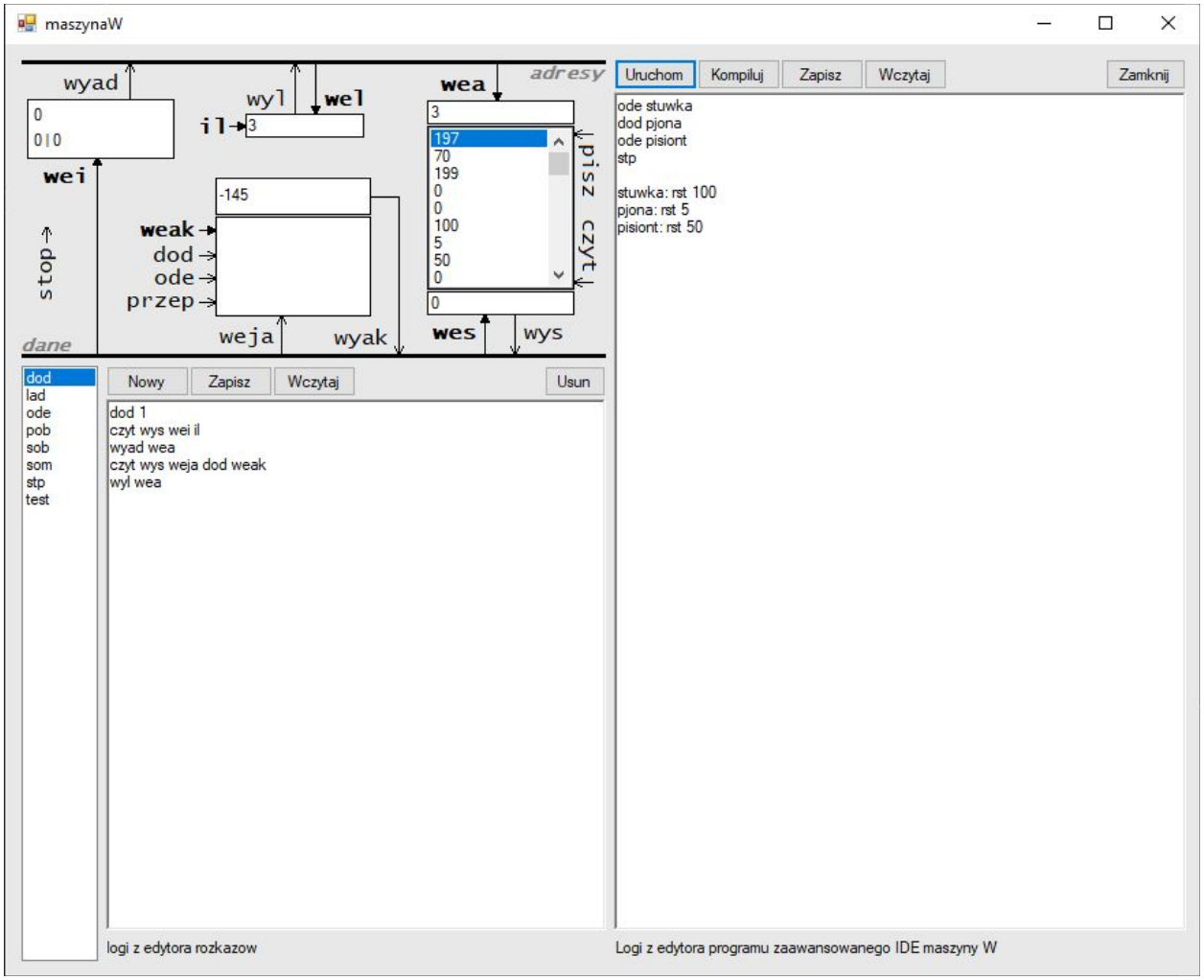
Maszyna W

Piotr Bosowski

1. Temat projektu i jego analiza

Tematem projektu jest program, który symuluje zachowanie Maszyny W - wirtualnego, dydaktycznego komputera, który można programować uproszczonym językiem asembler. Symulator umożliwia nie tylko pisanie kodu wykonywanego przez Maszynę W, ale także edycję istniejących i projektowanie nowych rozkazów, które Maszyna będzie w stanie wykonać. Projekt jest dopiero podstawową wersją Maszyny W, stworzony dotychczas program nie obsługuje przerwań, operacji wejścia-wyjścia ani sterowania ręcznego, będzie jednak przeze mnie rozwijany do momentu w którym dorówna funkcjonalnością i prześcignie wydajnością pierwowzór.

2. Specyfikacja zewnętrzna



W lewym górnym rogu znajduje się wizualizacja Maszyny W. Na diagramie znajdują się wszystkie sygnały sterujące - pogrubione zostały sygnały wykonywane na końcu taktu. Graficzne przedstawienie strzałek zostały mocno uproszczone w stosunku do oryginału. Interpreter rozkazów umieszczony jest wyżej, a licznik rozkazów nieco w prawo. Dzięki tym operacjom grafika jest zwarta, mimo to wciąż zawiera komplet informacji potrzebnych do łatwego pisania kodu.

Pod wizualizacją znajduje się edytor rozkazów. Za pomocą listy możemy przełączać się między dostępnymi rozkazami i dowolnie je edytować/podglądać, przyciskiem “nowy” tworzymy nowy rozkaz do edycji, a następnie zapisać go jednym kliknięciem w przycisk “zapisz” - rozkaz zostanie zwalidowany, automatycznie zostanie mu przypisana nazwa, będzie od teraz dostępny także po zrestartowaniu programu. Przyciskiem “wczytaj” możemy wczytać tekst rozkazu z pliku, a “usuń” kasuje aktualny rozkaz.

Cała prawa strona okna to edytor kodu programu - ponownie mamy możliwość wczytywania z pliku i zapisywania programu. Po naciśnięciu przycisku “kompiluj” kod programu zostaje przesłany do Kompilatora Maszyny W, który poddaje go procesowi asemblacji i wprowadza do pamięci operacyjnej Maszyny. Przycisk “uruchom” rozpoczyna właściwą pracę Komputera - Interpreter sczytuje pierwszy rozkaz, wprawiając tym samym w ruch całą wirtualną machinę.

3. Specyfikacja wewnętrzna

Maszyna W została przeze mnie napisana w poszanowaniu pełnej rozdzielności logiki od interfejsu - kompiluje się ją do postaci biblioteki klas i nie ma w niej żadnych odwołań do widocznego na zdjęciu interfejsu napisanego w nieco przestarzałej już technologii WindowsForms. Komunikacja między interfejsem a logiką następuje tylko poprzez interfejs IPublic.

Cały kod bibilioteki zamknięty jest w klasie MaszynaW (która jest jedyną polską nazwą w projekcie) która implementuje wspomniany interfejs IPublic. Klasa ta zawiera (kompozycja) wszystkie komponenty Maszyny: klasę Accumulator, Counter, Interpreter, Memory, AddrBus (szyna adresowa), DataBus (szyna danych), BusConnector (łącznik między szynami A. i D.). Mimo tylu składników jedynymi zadaniami klasy MaszynyW są: komunikowanie się z GUI (poprzez IPublic) oraz przechowywanie komponentów.

Schemat dziedziczenia przedstawia się następująco: Wszystkie elementy, z których zbudowana jest Maszyna, dziedziczą po klasie Component. Urządzenia (w przeciwieństwie do magistral) dziedziczą dodatkowo po klasie Device. W projekcie występuje jeszcze kilka mniejszych dziedziczeń, np. klasa Counter dziedziczy po klasie

Register (bo tak naprawdę licznik jest rejestrem rozbudowanym o funkcję inkrementacji).

Warto w tym miejscu wspomnieć, że instancjalizując Maszynę W podajemy tylko dwa parametry - WordLength i AddrLength, które warunkują, na ilu bitach pracować będzie Maszyna i jak duża będzie przestrzeń adresowa - ze względu na ograniczenia języka C# nie da się w prosty sposób stworzyć więcej niż 29 bitowej Maszyny - wynika to z ograniczenia wielkości pojedynczego obiektu (lub kolekcji) - przy 30 bitach adresowych pamięć RAM maszyny osiągnęłaby rozmiar prawie 4GB.

W projekcie znajdują się techniki takie jak wyjątki, wzorce, kontenery, iteratory, regexy.

4. Testowanie i uruchamianie

Program (zarówno biblioteka klas jak i interfejs) zostały przeze mnie przetestowane i działają poprawnie. Podstawowe rozkazy MaszynyW oraz rozkazy tworzone przez użytkownika wykonują się prawidłowo (o ile są poprawne). Programy kompilują się, asemblacja działa. Przez mój błąd nie została zaimplementowana funkcja skoku warunkowego - przeoczyłem fakt, że kod rozkazu SOM znacznie różni się strukturą od innych (formuła @JEŻELI… TO…), a obsłużenie takiego kodu wymaga zmiany struktury walidatora rozkazów, na co niestety nie mogłem sobie pozwolić z powodu braku czasu. Z tego powodu nie działają pętle (ale podprogramy już tak - instrukcja SOB działa poprawnie). W wersji beta 0.2 naprawię swój błąd i zaimplementuję obsługę instrukcji SOM :P

5. Uwagi i wnioski

Starałem się stworzyć oprogramowanie, które nie będzie wyręczać użytkownika w znajdowaniu błędów w kodzie - aby podnieść wartość dydaktyczną programu, obrałem za cel napisanie symulatora, który “przełknie wszystko” co wpisze w niego użytkownik - Maszyna nie wyświetlania komunikatów o błędach w kodzie, konfliktach na magistrali i niepoprawnych/powtarzających się sygnałach sterujących, nie prowadzi użytkownika za rękę tak jak nowoczesne IDE - każdy błąd należy znaleźć samemu. Ciekawym przykładem jest rozwiązywanie (a raczej jego brak) wspomnianych konfliktów na magistrali.

Czym jest konflikt na magistrali? Jeśli dwa urządzenia, np. rejestry, próbują wyemitować sygnał na tę samą magistralę w tym samym takcie, dochodzi do konfliktu. Oczywiście program wykrywa taką sytuację i mógłby zasygnalizować błąd, jednak tego nie robi. Dlaczego? Tak jak pisałem wyżej: przede wszystkim chciałbym, aby użytkownik świadomie pisał kod, zamiast pisać cokolwiek i drugie tyle czasu poprawiać wyskakujące coraz to nowe błędy. W poszukiwaniu alternatywy wyświetlania komunikatu o konflikcie sięgnąłem do sąsiedniej branży - elektroniki. Jak zachowałaby się prawdziwa, namacalna Maszyna W? Aby poznać wartość sygnału na skonfliktowanej magistrali, musielibyśmy zANDować sygnały - gdyby jedno urządzenie emitowało na danym przewodzie stan niski, a drugie wysoki, stan niski “ściągnąłby” do zera stan wysoki. Tak też zachowuje się symulator - gdy jedno urządzenie wyśle na magistralę sygnał 1100 (12) a drugie 0110 (6), otrzymamy wyjściowo 0100 (8) i żadnego komunikatu o błędzie. Elektroniczna Maszyna W jest tak niskopoziomowym urządzeniem, że na pewno nie wyświetliłaby żadnego okienka z ostrzeżeniem.

Dużą trudnością okazało się dla mnie wybranie odpowiedniej struktury dziedziczenia - wahałem się między trzema różnymi modelami i żaden nie wydawał mi się odpowiedni, każdy miał jakieś wady.

Na pewno przerobię przez wakacje cały projekt i będę go dalej rozwijał. Marzy mi się napisanie Maszyny W i wrzucenie jej na własny serwer, ale w taki sposób, aby kod programu był pobierany i wykonywany na komputerze klienta, a nie na serwerze. Niestety jedyną znaną mi technologią, która na coś takiego pozwala, jest JavaScript, a to skutecznie zniechęca mnie przed realizacją tego planu.