Architektura komputerów
Laboratorium 2 w terminie 29 marca 2021

Jakub Superczyński 241381

26 kwietnia 2021

1 Cel laboratorium

Celem laboratorium było napisanie w języku assemblera w architekturze 32-bitowej na platformę Linux programu wczytującego dwie liczby szesnastkowe ze standardowego wejścia (zakładając bufor na 200 cyfr dla każdej znich), mnożącego przez siebie dwie podane liczby a następnie wyświetlającego wynik działania. Choć na liście rozkazów istnieją rozkazy mnożące, 200 cyfr szesnastkowych znacznie przewyższa długość słowa maszynowego, tzn. najdłuższą liczbę, jaką można w prosty sposób wymnożyć. Dlatego należało znaleźc sposób na obejście tego problemu.

2 Opis implementacji

Program można podzielić na dwa elementy:

2.1 Konwersja z ASCII na liczbę i odwrotnie

Po wczytaniu danych ze standardowego wejścia dane przechowywane są jako znaki w formacie ASCII. Należało więc najpierw przekonwertować format tekstowy na liczbę, przy okazji walidując poprawność wprowadzonych danych. Jeden znak kodowany jest na dwóch bajtach, tak więc na jego miejsce należało wpisać dwie cyfry szesnastkowe. Robiłem w to następujący sposób: iterując przez tekst, pobierałem kolejno jeden bajt. Odejmowałem od niego odpowiednią wartość, zależną od tego czy była to cyfra poniżej 10 czy powyżej (rozróżniając również sposób zapisu - wielką lub małą literą). Po odjęciu miałem pewność że pierwsze 4 bity są wyzerowane, dlatego przesuwałem bity o 4 w lewo, przez co zerowałem drugą połówkę bajtu. Pobierałem więc kolejny bajt z tekstu wejściowego i w ten sam sposób odejmowałem pewną wartość, znowu mając pewność że 4 lewe bajty są równe 0. Następnie dodawałem te dwa bajty, niejako łącząc je w jedno i następnie zapisywałem do odpowiedniego miejsca w pamięci. Następnie przesuwam docelowy adres w pamięci i przechodzę do kolejnej pary bajtów. Robię to tak długo aż natknę się na znak końca linii, co oznacza koniec wprowadzanego tekstu.

Odwrotna zamiana dokonuje się w sposób analogiczny, tym razem jednak musimy rozdzielić jeden bajt na dwa. Znowu wykorzystujemy do tego operację przesuwania bitów. Pobieram kolejno każdy bajt do dwóch rejestrów (w jednym rejestrze konweruję starszą połowę bajtu, w drugim młodszą). Starszą część przesuwam o 4 w prawo, następnie dodaję odpowiednią wartość i zapisuję do odpowiedniego miejsca w pamięci. Młodszą częśc najpierw przesuwam o 4 w lewo by wyzerować starszą część, a następnie z powrotem o 4 w prawo, by nie zmienić jej wartości i postępuję tak samo: dodaję wartość i zapisuję w pamięci. W przypadku tej konwersji nie jest już potrzebna walidacja. Warto też zauważyć, że o ile adres docelowy z każdą iteracją jest zwiększany, o tyle adres źródłowy się zmniejsza. Wynika to z algorytmu mnożenia i sposobu zapisu wyniku.

2.2 Algorytm mnożenia

Jest to główna część zadania, można ją rozwiązać na kilka sposób. Ja wybrałem mnożenie bajt po bajcie ponieważ wydawało mi się to najprostszą metodą do zrozumienia. Pierwszym krokiem jest znalezienie adresu najmniej znaczących bajtów obu liczb, ponieważ od tej strony rozpoczyna się mnożenie, natomiast konwersja jaką zastosowaliśmy zapisuje bajty od najbardziej znaczącego. Wykorzystuję więc informację o liczbie bajtów w każdej z liczb, którą otrzymałem po konwersji i przesuwam adresy początkowe o odpowiednią liczbę. Następnie przechodzę do samego mnożenia.
W pamięci rezerwuję miejsce na wynik, na początku zainicjowanej zerami. Mnożenie odbywa się w dwóch pętlach, wewnętrznej i zewnętrznej. Wewnętrzna pętla iteruje po bajtach mnożnej, zewnętrzna po bajtach mnożnika. W przebiegu pętli wewnętrznej mnożymy odpowiednie bajty a otrzymany iloczyn dodaje do całościowego wyniku, trzymanego w pamięci, oczywiście na odpowiedniej pozycji, tzn. odpowiednim adresie. Pamiętając również o konieczności ewentualnego propagowania przeniesienia. Adres ten otrzymujemy dość prosto, do początkowego adresu wyniku dodajemy iterator wewnętrzny oraz zewnętrzny. Dzięki temu po każdym przebiegu wewnętrznej pętli otrzymujemy częściowy iloczyn i od razu dodajemy go do wyniku. Gdy iterator zewnętrznej pętli osiągnie ilość bajtów mnożnika, mnożenie się kończy.

3 Problemy podczas implementacji

Niestety, program nie działa całkowicie dobrze. Działa tylko w dość konkretnym przypadku, tzn. gdy obydwie liczby mają parzystą liczbę cyfr. Wydaje się że jest to niedoskonałość wybranego algorytmu mnożenia. Gdy liczba ma nieparzystą ilość cyfr, ostatni bit ma postać *0x0?*. Na początku pisania programu wydawało mi się że nie powinno mieć to wpływu, ponieważ taki bajt ma dokładnie tę wartość o którą nam chodzi. Po uruchomieniu i testowaniu programu okazało się, że w takich sytuacjach wynik jest liczony dla liczby z dodatkowym zerem na pozycji znaczącej. Próbowałem obejść ten problem poprzez przesunięcie bitów w ostatnim bajcie a następnie przeskalowaniu ostatecznego wyniku, jednak to

również nic nie dało.
Zadanie było dosyć wymagające. Po raz kolejny konwersja wprowadzonych znaków stanowiła pewnien problem, mimo że było to nieco podobne do zadania poprzedniego. Dość długo zajęło jednak wymyślenie sposobu na podział jednego bajtu na dwa, prawdopodobnie wynika to z małego doświadczenia w stosowaniu takich metod. Napisanie, ale może przede wszystkim zrozumienie algorytmu mnożenia również było dość trudne. Domyślałem się że potrzebne będą dwie pętle umożliwiające pozyskanie iloczynów częściowych, jednak dość dużo czasu potrzebowałem by dojść (jak widać nie do końca skutecznie) w jaki sposób następnie połączyć te iloczyny częściowe.