lab_3 - Instrukcja do ćwiczenia

Teoria (lab_3.pdf):

Chcemy wyświetlić zawartość wektora (tablicy jednowymiarowej) w postaci liczb dziesiętnych.

Algorytm zamiany liczby na jej postać dziesiętną polega na wielokrotnym dzieleniu całkowitym przez podstawę systemu (w tym przypadku 10). Kolejne reszty z dzielenia – po zamianie na znaki (cyfry dziesiętne) – utworzą zapis liczby. Iloraz uzyskany w danym kroku staje się dzielną w kroku kolejnym.

Przykład: liczba 357

Krok	Dzielna	Iloraz	Reszta	Znak
1	357	35	7	'7'
2	35	3	5	'5'
3	3	**0**	3	'3'

Warunkiem stopu w algorytmie jest wyzerowanie się ilorazu

Mając już sekwencję reszt z dzielenia zamieniamy je na odpowiednie znaki (cyfry dziesiętne). Ponieważ wewnętrzną reprezentacją znaków są ich kody ASCII, to cały proces sprowadza się do zamiany liczby na inną liczbę (0 -> kod ASCII '0', itp.) – w tym celu wystarczy dodanie kodu ASCII znaku '0', bo wszystkie reszty są liczbami z przedziału [0..9].

Znaki uzyskane po konwersji muszą znaleźć się w pamięci (bo do wyświetlenia potrzebny będzie adres łańcucha znaków) w odpowiedniej kolejności (dla liczby z przykładu napis ma mieć postać '357', a nie '753'). Ponieważ znaki - odpowiadające kolejnym resztom - pojawiają się w kolejności od najmniej znaczącej cyfry do najbardziej znaczącej ('7', '5', '3') a mają utworzyć napis '357' – trzeba je „odwrócić" (np. poprzez użycie stosu lub danej o dostępie swobodnym (tablicy).

Nowe instrukcje:

- **PUSH** register - zapisanie zawartości rejestru na stosie
- **POP** register – odtworzenie zawartości rejestru ze stosu
- **DIV** %ebx – dzielenie całkowite: %edx:%eax / %ebx = %eax i %edx
  - %edx:%eax - dzielna (64bit)
  - %ebx - dzielnik (32bit)
  - %eax - iloraz (32bit)
  - %edx - reszta (32bit)

- **LOOP** etykieta – instrukcja organizująca pętlę, wykorzystuje wewnętrznie rejestr %rcx jako licznik pętli, odpowiada następującej sekwencji operacji: { rcx--; if(rcx) goto etykieta }, sensowne użycie wymaga nadania wartości początkowej rejestrowi %rcx (liczba powtórzeń pętli)
- **MOV** address(,index,4), register - register = *(address+index*4);
- **MOV** register,(index) - *(index) = register;

Nowe dyrektywy:

- **.long** – rezerwacja miejsca na daną o rozmiarze 4 bajtów
- **.quad** – rezerwacja miejsca na daną o rozmiarze 8 bajtów

Praktyka (lab_3.s):

Dane:

- **vector** – wektor (tablica jednowymiarowa) liczb o rozmiarze 4 bajtów
- **count** – liczba elementów wektora (kropka w wyrażeniu oznacza adres bieżącej pozycji w trakcie kompilacji – znając adres pierwszego bajtu po wektorze i adres pierwszego bajtu wektora, można policzyć ile bajtów zajmuje wektor. Aby uzyskać liczbę elementów należy wielkość wektora podzielić przez rozmiar pojedynczego elementu
- **line_no** – łańcuch znaków wykorzystywany do wyświetlania liczby dziesiętnej (indeksu elementu wektora) - w miejsce spacji zostaną zapisane znaki po konwersji, przewidziano miejsce na maksymalnie 3 cyfry
- **number** – łańcuch znaków wykorzystywany do wyświetlania liczby dziesiętnej (wartości elementu wektora) - w miejsce spacji zostaną zapisane znaki po konwersji, przewidziano miejsce na maksymalnie 5 cyfr, na końcu łańcucha znajduje się znak końca linii (LF=0x0A) do formatowania wyświetlanego tekstu
- długości napisów (wartości stałych **item_len**, **FL_len**, **LF_len**) wyznaczono poprzez odjęcie od adresu pierwszego znaku po napisie adresu pierwszego znaku napisu

Kod:

- **program główny** (komunikat, wyświetlenie zawartości wektora w kolejności od pierwszego do ostatniego elementu, komunikat, wyświetlenie zawartości wektora w kolejności od ostatniego do pierwszego elementu, zakończenie działania programu)
- **disp_vector_FL** – wyświetlenie zawartości wektora w kolejności od pierwszego do ostatniego elementu, pętla powtarzana **count** razy, zbudowana z wykorzystaniem instrukcji **LOOP**. Licznikiem pętli jest rejestr **rcx**, rejestr **rsi** jest indeksem do kolejnych elementów tablicy (zmienia się w zakresie **0..count-1**). Zawartość jest wyświetlana linia po linii, w każdej linii pojawiają się dwie liczby: indeks elementu wektora (**esi** –mniej znacząca część **rsi**) oraz wartość elementu wektora (**ebx**)
- **disp_vector_LF** – wyświetlenie zawartości wektora w kolejności od ostatniego do pierwszego elementu, pętla powtarzana **count** razy, zbudowana z wykorzystaniem instrukcji **LOOP**. Licznikiem pętli jest rejestr **rcx**, rejestr **rsi** jest indeksem do kolejnych elementów tablicy (zmienia się w zakresie **count-1..0**). Zawartość jest wyświetlana linia po linii, w każdej linii pojawiają się dwie liczby: indeks elementu wektora (**esi** –mniej znacząca część **rsi**) oraz wartość elementu wektora (**ebx**)
- **make_string** – przygotowanie łańcucha znaków do wyświetlenia, funkcja wywołuje dwukrotnie procedurę konwersji liczby (indeksu elementu i jego wartości) na reprezentację dziesiętną wskazując dwa różne miejsca do zapisu uzyskanych ciągów znaków – indeks elementu zostanie zapisany w **line_no**, wartość w **number**. Ponieważ w procesie konwersji cyfry dziesiętne będą zapisywane w kolejności od prawej do lewej (aby odwrócić kolejność uzyskiwanych reszt/cyfr dziesiętnych) przekazywane są adresy ostatnich miejsc przewidzianych na zapis liczb w postaci znakowej: **line_no+2** (**line_no** zawiera miejsce na trzy cyfry: **line_no** albo **line_no+0** to pierwsza cyfra, **line_no+1** to

druga, a **line_no+2** to trzecia i ostatnia) i **number+4** (**number** zawiera miejsce na pięć cyfr, więc **number+4** to adres piątej i ostatniej cyfry)

- **num2dec** – funkcja dokonująca konwersji liczby (parametr w rejestrze **eax**) na postać dziesiętną, wykorzystuje dzielenie przez **10** umieszczone wewnątrz pętli **do..while()**. Warunkiem stopu jest zerowa wartość ilorazu. Kolejne reszty z dzielenia (liczby **0..9** zawarte w rejestrze **edx/dx/dl**) po konwersji na znaki są zapisywane w pamięci pod adresem znajdującym się w rejestrze **rdi**. Znaki są zapisywane w kolejności od prawej do lewej, po zapisie znaku zawartość **rdi** jest zmniejszana o **1** (przesunięcie w lewo). Liczba do konwersji jest 32-bitowa (dzielnik=**10** też), ale żeby uzyskać iloraz i resztę w tej samej postaci, konieczne jest wykorzystanie dzielnej w postaci liczby 64-bitowej zawartej w parze rejestrów **edx** (bardziej znacząca część) i **eax** (mniej znacząca). Aby taka dzielna miała identyczną wartość jak liczba do konwersji, należy wyzerować rejestr **edx**. Zerowanie realizowane jest w pętli, bo po każdym dzieleniu rejestr **edx** zawiera najpierw resztę z dzielenia, a potem znak (cyfrę dziesiętną) uzyskany po konwersji. Wykorzystywane w trakcie dzielenia rejestry **ebx** i **edx** są na początku funkcji zapisywane na stosie w celu ich ochrony, a na końcu odtwarzana jest ich oryginalna zawartość.

Działania:

1. CLR (Compile, Link, Run)
2. Naruszenie ochrony pamięci, ale pierwszy element wektora został wyświetlony prawidłowo, co oznacza, że sama konwersja i przygotowanie łańcucha znaków do wyświetlania działa poprawnie. Błędu/błędów trzeba szukać w funkcji **disp_vector_FL**.
3. Problem polega na użyciu rejestrów **rcx** i **rsi** – te same rejestry są wykorzystywane przez funkcję systemową **write_64**. Konieczna jest ochrona tych rejestrów na czas wywoływania funkcji systemowej w makrze **disp_str_64**. Uzyskujemy to poprze dodanie instrukcji **push** i **pop** w odpowiednich miejscach:

   push %rcx
   push %rsi
   disp_str_64 …
   pop %rsi
   pop %rcx

4. Identyczne zmiany wprowadzamy w kodzie funkcji **disp_vector_LF**.
5. CLR
6. Nie ma już błędu „Naruszenie ochrony pamięci”, ale wyświetlanych jest zbyt wiele elementów wektora. Sama pętla wykonywana jest poprawnie, więc problem związany jest z nadaniem wartości początkowej licznikowi pętli. Instrukcja **mov count, %rcx** przenosi do rejestru **rcx 8** bajtów z pamięci począwszy od adresu **count** – znamy zawartość tylko **4** pierwszych bajtów (zmienna **count**), pozostałe to już napis „**Item**”.
7. Zmieniamy deklarację **.long** na **.quad** – teraz zmienna **count** ma rozmiar **8** bajtów.
8. CLR
9. Wyświetlanych jest **20** elementów wektora – faktycznie jest ich **10**. Problem polega na błędnym obliczaniu liczby elementów przy nadawaniu wartości zmiennej **count** – różnica adresów (czyli wielkość wektora w bajtach) powinna być podzielona przez rozmiar pojedynczego elementu (**4** bajty) lub przesunięta o **2** pozycje w prawo (jeśli użyto operatora przesunięcia, a nie dzielenia).
10. Poprawiamy na: **(. – vector) >> 2**.
11. CLR
12. Wyświetlanych jest już **10** elementów, ale nigdzie nie widać wartości **0** chociaż w wektorze jest – są za to dwie liczby **20** przy wyświetlaniu zawartości od pierwszego elementu do ostatniego i dwie liczby **30** przy wyświetlaniu od ostatniego elementu do pierwszego. Problem polega na tym, że znaki po konwersji zapisywane są w tym samym miejscu (zmienna **number**) – liczba **0** po konwersji to tylko jedna cyfra dziesiętna (znak), która nie nadpisuje w pełni poprzedniej zawartości zmiennej **number** – stąd widoczne są znaki, które zostały zapisane wcześniej. Konieczne jest „czyszczenie” (wypełnianie spacjami) miejsca przeznaczonego na zapis znaków czyli zmiennej **number**. Teoretycznie do „czyszczenia” mamy pięć znaków (co mogłoby stwarzać kłopoty, bo prosto i szbko można zmodyfikować **1**, **2**, **4** lub **8** bajtów), ale wystarczy „czyścić” tylko **4** pierwsze znaki – ostatni zawsze jest nadpisywany!).

13. Na początku funkcji **make_string** dodajemy instrukcję: **movl $0x20202020, number**- w ten sposób zmodyfikujemy **4** pierwsze bajty zmiennej number, nadając każdemu z nich wartość **0x20** (**32** dziesiętnie) co jest tożsame z wstawieniem tam znaków spacji.
14. CLR
15. Pojawia się już wartość **0** – wygląda na to, że wszystko jest OK.
16. Dodajemy do wektora w dowolnym miejscu jedną lub dwie nowe liczby (nieujemne i zawierające mniej niż **5** cyfr).
17. CLR
18. Okazuje się, że problem z brakiem nadpisywania się znaków dotyczy też indeksów elementów wektora (zmiennej **line_no**) – wcześniej wszystkie indeksy były jednocyfrowe, więc błąd nie był widoczny. „Czyścimy” - tym razem tylko dwa znaki (w trzyznakowej zmiennej **line_no**).
19. Na początku funkcji **make_string** dodajemy instrukcję: **movw $0x2020, line_no** – w ten sposób zmodyfikujemy dwa pierwsze bajty zmiennej **line_no**, nadając każdemu z nich wartość **0x20** (**32** dziesiętnie) co jest tożsame z wstawieniem tam znaków spacji.
20. CLR
21. Teraz jest OK i będzie tak, dopóki dane będą spełniać założenia: indeksy maksymalnie trzycyfrowe, co oznacza nie więcej elementów niż **1000**, oraz maksymalnie pięciocyfrowe i nieujemne wartości samych elementów
22. Świętujemy kolejny sukces!