lab_2 - Instrukcja do ćwiczenia

Teoria (lab_2.pdf):

Chcemy wyświetlić liczby w postaci hex bo inne reprezentacje są trudniejsze (dziesiętna) lub mniej przydatne (dwójkowa, ósemkowa).

Jeżeli liczba mieści się w jednym bajcie, to w zapisie uzyskamy dwa znaki (cyfry szesnastkowe). Każda cyfra odpowiada czterem bitom (połówkom bajtu – ang. nibble). Ponieważ najmniejszą porcją danych do której możemy się dobrać niezależnie od innych danych jest bajt, to w procesie konwersji bajt wejściowy zastąpimy dwoma bajtami, z których każdy będzie miał wartość odpowiadającą połówce bajtu oryginalnego. Mniej znaczącą połówkę uzyskujemy poprzez wykonanie operacji AND (iloczynu logicznego) oryginalnej wartości i liczby 0x0F (dec: 15). Bardziej znaczącą połówkę uzyskamy przez wykonanie operacji SHR (logiczne przesunięcie bitowe w prawo) z argumentem 4 (4 pozycje bitowe) – wtedy bardziej znaczące bity znajdą się w miejscu mniej znaczących, a w miejscu bardziej znaczących znajdą się zera (oryginalne mniej znaczące bity znikną).

Mając już dwie liczby zamieniamy je na odpowiednie znaki (cyfry szesnastkowe). Ponieważ wewnętrzną reprezentacją znaków są ich kody ASCII, to cały proces sprowadza się do zamiany liczby na inną liczbę (0 -> kod ASCII ‘0’, itp.). Niestety, nie można użyć jednej formuły do zamiany, bo kody ASCII liter A, B, …, F nie znajdują się bezpośrednio po kodach cyfr 0..9. Konieczne jest użycie jednej z dwóch różnych formuł w zależności od tego, czy liczba jest mniejsza od 10 (wtedy zamieniamy ją na znak ‘0’..’9’), czy nie (wtedy zamieniamy ją na znak ‘A’..’F’).

Znaki uzyskane po konwersji muszą znaleźć się w pamięci (bo do wyświetlenia potrzebny będzie adres łańcucha znaków) w odpowiedniej kolejności (dla liczby 15 napis ma mieć postać 0F, a nie F0)

Nowe instrukcje:

- **AND** value, register - register &= value;
- **CMP** value, register - result = register – value; dodatkowym działaniem oprócz odejmowania jest ustawienie różnych znaczników (flag) procesora. Kolejną operacją zwykle jest skok warunkowy uzależniony od stanu określonych znaczników
- **JB** – Jump If Below (połączenie obu instrukcji daje efekt: if(register < value) goto …
- **ADD** value, register - register += value;
- **SHR** value, register - register >>= value;

Praktyka (lab_2.s):

Dane:

- **hex_str** – łańcuch znaków wykorzystywany do wyświetlania liczb szesnastkowych (w miejsce zer zostaną zapisane znaki po konwersji, dodatkowa spacja dla rozdzielenia liczb w linii
- **new_line** – znak końca linii (LF=0x0A) do formatowania wyświetlanego tekstu

Kod:

- **r15** – licznik pętli (wyświetlamy 256 liczb – 0..255)
- **r14** – liczba do konwersji i wyświetlenia (licznik zmienia się w dół w zakresie 256..1, więc nie da się go wykorzystać wprost do uzyskania liczby zmieniającej się w górę w zakresie 0..255)
- **xor %r14, %r14** – to prostszy i szybszy sposób na uzyskanie w rejestrze wartości 0 niż instrukcja **mov $0, %r14**
- konwersją zajmuje się funkcja num2hex – jako argument dostaje liczbę w rejestrze **al** (przenosimy tam zawartość najmniej znaczącej części rejestru **r14** – bajt o wartości 0..255), zaś rezultat konwersji w postaci dwóch znaków zwracany jest w rejestrze **ax**, którego zawartość trzeba przenieść do łańcucha **hex_str** (w miejsce oryginalnych zer)
- wyświetlenie łańcucha znaków
- poprawiamy formatowanie danych poprzez wyświetlenie co 16 liczb znaku końca linii - jako źródło informacji o tym, czy wyświetlać znak czy nie, używamy rejestru **rax**, który zawiera kopię danych z **r14** (nie możemy użyć **r14** wprost, bo chcemy policzyć wartość **x%=16** (**x=x mod 16**), co redukuje wartość **x** do przedziału 0.15). Zerowa wartość oznacza konieczność wyświetlenia znaku końca linii, dla innych wartości wyświetlanie jest pomijane.
- zwiększenie **r14** (liczby do konwersji)
- zmniejszenie licznika pętli **r15**
- zakończenie działania po wykonaniu zadanej liczby powtórzeń pętli
- funkcja **num2hex** – dyrektywa **.type num2hex, @function** może być pominięta, istotne jest użycie etykiety wskazującej miejsce, od którego powinny być wykonywane kolejne instrukcje oraz instrukcji **ret**, skutkującej powrotem do kodu, w którym nastąpiło wywołanie funkcji (użycie instrukcji **call** powoduje zapamiętanie na stosie adresu powrotu – jest nim adres instrukcji znajdującej się bezpośrednio po instrukcji **call**)

Działania:

1. CLR (Compile, Link, Run)
2. Problem z wyświetlaniem - chcemy co jakiś czas dodać LF.
3. Usuwamy komentarze w liniach 54-57.
4. CLR
5. Jest prawie dobrze - poza pierwszym i ostatnim wierszem.
6. Przenosimy instrukcję **inc %r15** tak, aby znalazła się zaraz po **disp_str_64**
7. CLR
8. Wyświetlanie znaków w pętli jest OK.
9. Dodajemy konwersję liczby na hex - usuwamy komentarze w liniach 48-50.
10. C
11. Problem w linii 50 - zapis **MOVW %ax, $hex_str** oznacza **&hex_str = ax** (zmiana adresu), ale adresu nie da się zmienić!!! Tak naprawdę chcemy zmienić wartość czyli powinno być **MOVW %ax, hex_str**
12. CLR
13. Coś się dzieje, ale niezależnie od zawartości **r14**, liczba wygląda tak samo - szukamy błędów w funkcji **num2hex**
14. Zmienna **tmp** powinna przechowywać kopię argumentu funkcji czyli rejestru **al**
    movb tmp, %al # %al = tmp;
    movb %al, tmp # tmp = %al;
15. CLR
16. Dane się zmieniają, ale to nie są liczby hex! - szukamy dalej.
17. Jeśli połówka bajtu jest mniejsza od **10**, to skok i dodajemy kod ASCII znaku **'0'**. Jeśli nie, to dodajemy kod **'A'** i odejmujemy **10**, ale dziesiętnie! (0x10 = 16).
18. CLR
19. Mniej znacząca cyfra jest OK, problem z konwersją dotyczy bardziej znaczącej.
20. Ten sam problem co w punkcie 17.
21. CLR
22. Dalej coś jest źle - mamy dodać kod ASCII znaku **'0'** a nie **'O'**
23. CLR
24. Bardziej znacząca cyfra zmienia się zbyt szybko i skokowo (0,2,4,...) - do wyłuskania bardziej znaczącej połówki bajtu użyto instrukcji logicznego przesunięcia bitowego w prawo (**SHR**=Shift Right), ale bity mają być przesunięte o **4** pozycje w prawo a nie o **3**
25. CLR
26. Yes, yes, yes! Radość, euforia, wręcz ekstaza - u niektórych łzy wzruszenia - tak długo oczekiwany sukces wreszcie nadszedł!
27. Cieszymy się i w napięciu czekamy na kolejne zajęcia.