Marcin Stępniak

Architektura systemów komputerowych
Laboratorium 7
Symulator SMS32
Stos, instrukcje skoku i pętle

1. Informacje

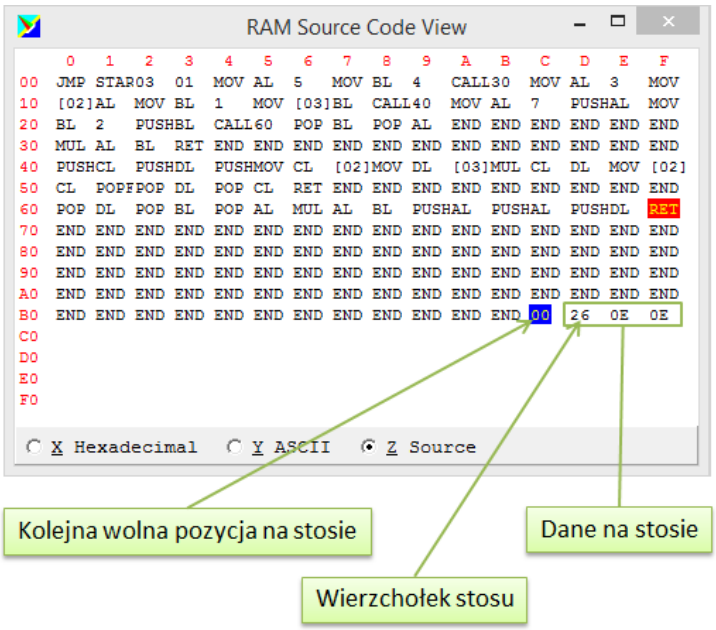
1.1. Stos

Stos jest strukturą danych, w której dane dokładane są na wierzch stosu (zwany też wierzchołkiem stosu) i z niego są też pobierane. Jest to bufor typu LIFO, Last In, First Out; ostatni na wejściu, pierwszy na wyjściu). Stos stosowany w informatyce można porównać do stosu książek wkładanego do ciasnego pudełka. Książki można odkładać tylko na wierzch stosu i da się je brać jedynie z wierzchu stosu. Chcąc wyjąć książkę znajdującą się niżej, konieczne jest wcześniejsze podniesienie książek leżących na niej.

W mikroprocesorach, adres kolejnego wolnego elementu na stosie jest zapisany w rejestrze SP (ang. stack pointer). Większość implementacji stosu, po dodaniu elementu zmniejsza adres znajdujący się w rejestrze SP. W symulatorze SMS32 jest to rozwiązane właśnie w ten sposób, a początkowy adres w SP to *BF*.

Rysunek 1 przedstawia widok pamięci symulatora SMS32. W tym przypadku mamy stos, na którym odłożono 3 elementy (bajty). Komórka o adresie z rejestru SP jest oznaczona niebieskim tłem.

Tabela 1 zawiera objaśnienie czterech instrukcji manipulacji stosem, które udostępnia symulator SMS32.



Rysunek 1. Stos w symulatorze SMS32

Tabela 1. Instrukcje operujące na stosie

Instr.	Operand	Działanie
PUSH	rejestr ogólnego przeznaczenia	odkłada (kopiuje) wartość rejestru na sos
POP	rejestr ogólnego przeznaczenia	zdejmuje wartość ze stosu i zapisuje do rejestru
PUSHF		odkłada wartość rejestru flag na stos
POPF		pobiera wartość rejestru flag ze stosu

1.2. Instrukcje skoku

Instrukcje skoku zmieniają wskaźnik instrukcji w rejestrze IP. Powoduje to, że po wykonaniu skoku program kontynuuje pracę w innym miejscu. To miejsce ustalane jest za pomocą etykiety.

Etykiety dodawane są do programu w postaci „*nazwa_etykiety:*". Ten zapis wskazuje miejsce w kodzie programu, do którego można skoczyć na podstawie nazwy etykiety. Instrukcje skoku w języku asemblera zapisujemy w postaci „*nazwa_instrukcji_skoku nazwa_etykiety*". W tabeli 2 zostały opisane dostępne instrukcje skoku.

Warunkowe instrukcje skoku korzystają z rejestru flag SR (ang. Status Register). Rejestr ten jest modyfikowany przez operacje arytmetyczne i logiczne. Rejestr SR zawiera następujące flagi:

— Flaga "Z" (zera) jest ustawiana na jeden, jeśli obliczenia dały wynik zerowy.
— Flaga "S" (znaku) jest ustawiana na jeden, jeśli obliczenia dały wynik ujemny.
— Flaga "O" (przepełnienia) jest ustawiana, jeśli wynik był zbyt duży, aby zmieścić się w rejestrze. Gdy wartość rejestru osiągnie $7F_{(16)}$ lub $127_{(10)}$, to następna powinna być liczba 128, ale ze względu na sposób, w jaki numery są przechowywane w systemie binarnym, następną liczbą jest minus 128. Efekt ten nazywany jest przepełnieniem.
— Flaga "I" (przerwania) jest ustawiana, jeśli przerwania są włączone.

Tabela 2. Instrukcje skoku symulowanego mikroprocesora

Instr.	Działanie
JMP	skok bezwarunkowy; zawsze skacze do wskazanej etykiety
JZ	skacze do wskazanej etykiety jeżeli flaga "Z" jest ustawiona
JNZ	skacze do wskazanej etykiety jeżeli flaga "Z" nie jest ustawiona
JS	skacze do wskazanej etykiety jeżeli flaga "S" jest ustawiona
JNS	skacze do wskazanej etykiety jeżeli flaga "S" nie jest ustawiona
JO	skacze do wskazanej etykiety jeżeli flaga "O" jest ustawiona
JNO	skacze do wskazanej etykiety jeżeli flaga "O" nie jest ustawiona

Czasami zachodzi konieczność porównania dwóch wartości, a nie chcemy wykonywać operacji arytmetycznej ani logicznej. W takiej sytuacji można wykorzystać instrukcję CMP, która działa tak jak SUB, ustawia odpowiednie flagi, ale nie zapisuje wyniku w żadnym rejestrze. Działanie instrukcji CMP zostało zaprezentowane w tabeli 3.

Tabela 3. Przykłady działania instrukcji CMP

Instrukcja	Działanie
CMP AL,BL	Ustawia flagę 'Z' jeżeli AL = BL Ustawia flagę 'S' jeżeli AL < BL
CMP BL,13	Ustawia flagę 'Z' jeżeli BL = 13 Ustawia flagę 'S' jeżeli BL < 13
CMP CL,[20]	Ustawia flagę 'Z' jeżeli CL = [20] Ustawia flagę 'S' jeżeli CL < [20]

Listing 1 zawiera kod programu w języku asemblera wraz z komentarzami objaśniającymi działanie poszczególnych instrukcji skoku. Zrozumienie działania tych instrukcji będzie niezbędne do rozwiązania zadań. Program w

zamieszczonej wersji nigdy się nie kończy. Po ostatnim teście następuje skok bezwarunkowy do testu pierwszego.

Listing 2 zawiera kod programu z przykładami wyjaśniającymi działanie instrukcji CMP. Podobnie jak w przykładzie pierwszym, program nigdy się nie kończy. Po ostatnim teście następuje skok bezwarunkowy do testu pierwszego.

Listing 1. Program demonstrujący instrukcje skoków

```
   CLO      ; zamknij wszystkie okna.

test1:
   MOV   AL, 7
   SUB   AL, 7
   JZ test2    ;skacze do nastepnego testu
            ;jezeli ustawiona jest flaga zera
   JNZ   end_label   ;skacze na koniec
            ;jezeli nie jest ustawiona flaga zera

test2:
   MOV   AL, 6E      ;110
   ADD   AL, 2C      ;44
   JO test3    ;skacze do nastepnego testu
            ;jezeli ustawiona jest flaga przepelnienia
   JNO   end_label   ;skacze na koniec
            ;jezeli nie jest ustawiona flaga przepelnienia

test3:   ;przepelnienie dla liczb ujemnych
   MOV   AL, 89      ;-119
   SUB   AL, 2C      ;44
   JO test4    ;skacze do nastepnego testu
            ;jezeli ustawiona jest flaga przepelnienia
   JNO   end_label   ;skacze na koniec
            ;jezeli nie jest ustawiona flaga przepelnienia
test4:
   MOV   AL, 2C      ;44
   SUB   AL, 6E      ;110
   JS unconditional  ;skacze do etykiety unconditional
            ;jezeli ustawiona jest flaga znaku
   JNS   end_label   ;skacze na koniec
            ;jezeli nie jest ustawiona flaga znaku
unconditional:
   JMP   test1    ;skacze bezwarunkowo do etykiety test1
end_label:
   END
```

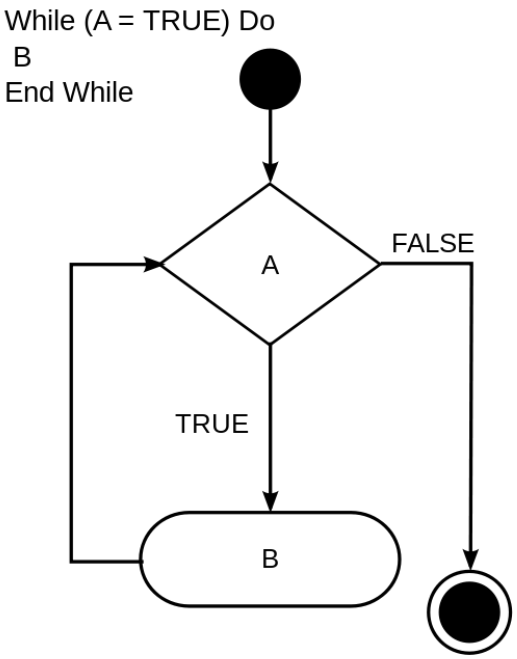
Listing 2. Program demonstrujący instrukcje porównania (CMP) i skoków warunkowych

```
   CLO      ; zamknij wszystkie okna.

test1:
   MOV   AL, 5
   CMP   AL, 5
   JZ test2    ;skacze do nastepnego testu jezeli
            ;ustawiona jest flaga zera (warunek 5=5)
   JNZ   end_label   ;skacze na koniec jezeli
            ;nie jest ustawiona flaga zera (niespelniony warunek 5!=5)
test2:
   MOV   AL, 5
   CMP   AL, 7
   JS test3    ;skacze do nastepnego testu jezeli
            ;ustawiona jest flaga znaku (warunek 5<7)
   JNS   end_label   ;skacze na koniec jezeli
            ;nie jest ustawiona flaga znaku (niespelniony warunek 5>=7)
test3:
   MOV   AL, 5
   CMP   AL, 3
   JZ end_label   ;skacze na koniec jezeli
            ;jest ustawiona flaga znaku (niespelniony warunek 5=3)
   JNS   unconditional  ;skacze do etykiety unconditional jezeli
            ;nie jest ustawiona flaga znaku
            ;(spelniony warunek 5>=3)
            ;razem z warunkiem na zero mamy 5>3
   JMP   end_label   ;skacze na koniec jezeli poprzednie
            ;instrukcje warunkowe nie zostaly wykonane
unconditional:
   JMP   test1    ;skacze bezwarunkowo do etykiety test1
end_label:
   END
```

1.3. Dodatkowe informacje

— http://www.softwareforeducation.com/sms32v50/sms32v50_manual/index.htm



Rysunek 2. Diagram przepływu dla pętli while

2. Zadania

2.1. Zadanie 1

Napisać program zawierający pętlę zmniejszającą rejestr AL o $2_{(10)}$ w każdej iteracji. Początkową wartość rejestru należy ustawić na $11_{(10)}$. Warunkiem wyjścia z pętli jest wartość rejestru AL mniejsza od 0 (AL<0).

2.2. Zadanie 2

Napisać program zawierający pętlę zwiększającą rejestr AL o $3_{(10)}$ w każdej iteracji. Początkową wartość rejestru należy ustawić na 0. Warunkiem wyjścia z pętli jest wartość rejestru AL większa od $63_{(10)}$ (AL>$63_{(10)}$).

2.3. Zadanie 3

Napisać program, który zawiera zaimplementowaną pętlę typu while w postaci while(x<$71_{(10)}$){x++}. Zmienna x powinna być ustawiana przed pętlą. Należy wziąć pod uwagę fakt, że pętla może nie wykonać się ani razu.

2.4. Zadanie 4

Napisać program umieszczający w rejestrze AL kolejne liczby ciągu Fibonacciego (pierwszy wyraz jest równy 0, drugi jest równy 1, każdy następny

jest sumą dwóch poprzednich). Program powinien zakończyć swoją pracę po wystąpieniu przepełnienia w rejestrze AL. W zadaniu należy wykorzystać stos.