Powtórzenie

Na kolokwium pojawić się mogą zagadnienia z całego przedmiotu. Jestem w pełni świadom, że będziecie podczas kolokwium korzystali z Internetu, nawet nie zamierzam z tym walczyć, więc spoko, można.

Ten PDF zawiera informacje, na których szukanie podczas kolosa jest już zdecydowanie za późno. Są tu skompresowane podstawy tego, co przerabialiśmy na zajęciach.

Rejestry

Rejestry są komórkami pamięci bezpośrednio na procesorze. W naszym wypadku, bo pracujemy na systemie 16-bitowym, wyposażonym w emulowany procesor 80486 z 89 roku, mamy dostęp do rejestrów, w których możemy umieścić do 16 bitów danych. Rejestry, z których będziemy najczęściej korzystać to **AX**, **BX**, **CX** i **DX**. Każdy z tych rejestrów dzieli się jeszcze na dwie części, górną zawierającą starszy bajt (**H**igh) i dolną zawierającą młodszy bajt (**L**ow). Tak więc rejestry 8-bitowe to odpowiednio **AH, AL, BH, BL, CH, CL, DH, DL.**

Adresowanie

Tryby adresowania określają sposób, w jaki dobieramy się do adresu, czyli położenia operandu (argumentu).

Adresowanie rejestrowe

```
Mov AX, BX
```

Przenosi zawartość rejestru BX do rejestru AX

Adresowanie bezpośrednie

```
Mov AX, 10h
```

Ustawi rejestr AX na 0010

Adresowanie pośrednie

```
Mov AX, [0100h]
```

Ustawi rejestr AX na wartość znajdującą się w pamięci pod adresem 0100 (i 0101)

W temacie pamięci pozostając

Odpalając insighta widzimy, że pamięć (to na dole) jest reprezentowana przez parę liczb. Pierwsza z nich wskazuje na segment, druga na offset, czyli przesunięcie adresu względem segmentu. Na potrzeby tych zajęć nie jest potrzebne manipulowanie segmentem, natomiast offset bardzo się przyda, bo będzie wskazywał na konkretne dane w pamięci. Instrukcja

```
Mov AX, string
```

Ustawi rejestr na adres wskazywany przez etykietę *string*. Natomiast coś takiego:

```
string db „kocham asemblera”
```

ustawi nam w segmencie danych (wczytany zaraz za programem) napis „kocham asemblera” i zapamięta adres jego początku, abyśmy mogli odnosić się do niego przy użyciu etykiety string

Jeśli chcemy ustawić na jakimś adresie jakąś daną, musimy podać jeszcze jej rozmiar:

```
mov byte [string], AH
```

Taka instrukcja wstawi na adresie wskazywanym przez etykietę string zawartość rejestru AH. Jeśli chcielibyśmy ustawić 2 bajty za jednym razem, to jest to możliwe

```
mov word [string], AX
```

Przerwania

Przerwania możemy porównać do funkcji. Każda rodzina przerwań zajmuje się czym innym, dla przykładu przerwanie 21h to rodzina przerwań systemu DOS, przerwanie 10h to przerwania karty wideo, 16h to przerwania klawiatury. Aby przerwanie się wykonało, należy wykonać instrukcję

```
int 21h
```

To, co wykona nam przerwanie zależy od tego, jak ustawimy odpowiednie rejestry, zazwyczaj zachowanie po wywołaniu przerwania określa nam rejestr AH, czasem AX. Cała lista przerwań dostępna jest tu: http://www.ctyme.com/intr/int.htm

Niektóre przerwania do prawidłowego zadziałania wymagać będą ustawienia jeszcze innych rejestrów, a jeszcze inne po wykonaniu w odpowiednim rejestrze umieszczą jakieś dane. Dla przykładu, aby wypisać znak na standardowym outpucie użyjemy przerwania INT 21h/AH=02h. http://www.ctyme.com/intr/rb-2554.htm

```
mov AH, 02h
mov DL, 41h
int 21h
```

To wydrukuje nam literkę ‘A’ na terminalu.

Arytmetyka

```
INC AX          ;zwiększa AX o 1
DEC AX          ;zmniejsza AX o 1
ADD AX, 10h     ;dodaje do AX 10h – można też podać rejestr
SUB AX, 10h     ;odejmuje od AX 10h – można też podać rejestr
MUL BX          ;mnoży BX przez AX
DIV BH          ;dzieli AX przez BH.
```

Pętle

Pętle są w założeniu i realizacji bardzo proste. Do wykonania pętli posłuży instrukcja loop wskazująca na etykietę opisującą pętlę. Za każdym razem, w którym program podczas wykonywania dotrze do instrukcji loop, zmniejszy rejestr CX o 1 i dokona skoku do etykiety opisującej pętlę

```
Mov CX, 10
petla:
;jakiś kod do wykonania
loop petla
```

Taka kombinacja wykona pętlę 10 razy.

Skoki warunkowe i bezwarunkowe

Są dwa rodzaje skoków: warunkowe i bezwarunkowe. Skok wykonujemy do etykiety, a dokładniej do adresu wskazywanego przez etykietę. Skoki bezwarunkowe określamy instrukcją

```
jmp jajco
```

Teraz za każdym razem, jak podczas wykonywania program napotka taką instrukcję to przeskoczy do etykiety jajco.
Skoki warunkowe natomiast mają inną specyfikę, bo ich wykonanie zależy od ustawionych flag. Najpierw musimy wykonać polecenie porównujące, np.

```
cmp AX, 08h
```

Zależnie od wyniku takiego porównania skoki się wykonają lub się nie wykonają i program przejdzie z wykonaniem dalej. https://pl.wikibooks.org/wiki/Asembler_x86/Instrukcje/Skokowe#jmp