Co to ten Asembler?

Sam chciałbym wiedzieć. Z technicznego punktu widzenia jest to program, który tworzy kod maszynowy na podstawie kodu źródłowego bazującego na podstawowych operacjach procesora. Sam procesor rozumie jedynie instrukcje maszynowe, zasadniczo ciąg zer i jedynek. Tutaj, cały na biało, wchodzi nam właśnie asembler. Tutaj jedno polecenie odpowiada jednemu rozkazowi procesora. Niezrozumiałe dla człowieka kody operacji w języku maszynowym zostały zastąpione mnemonikami, czyli składających się z kilku liter kodów, takich jak mov, div, czy mul.

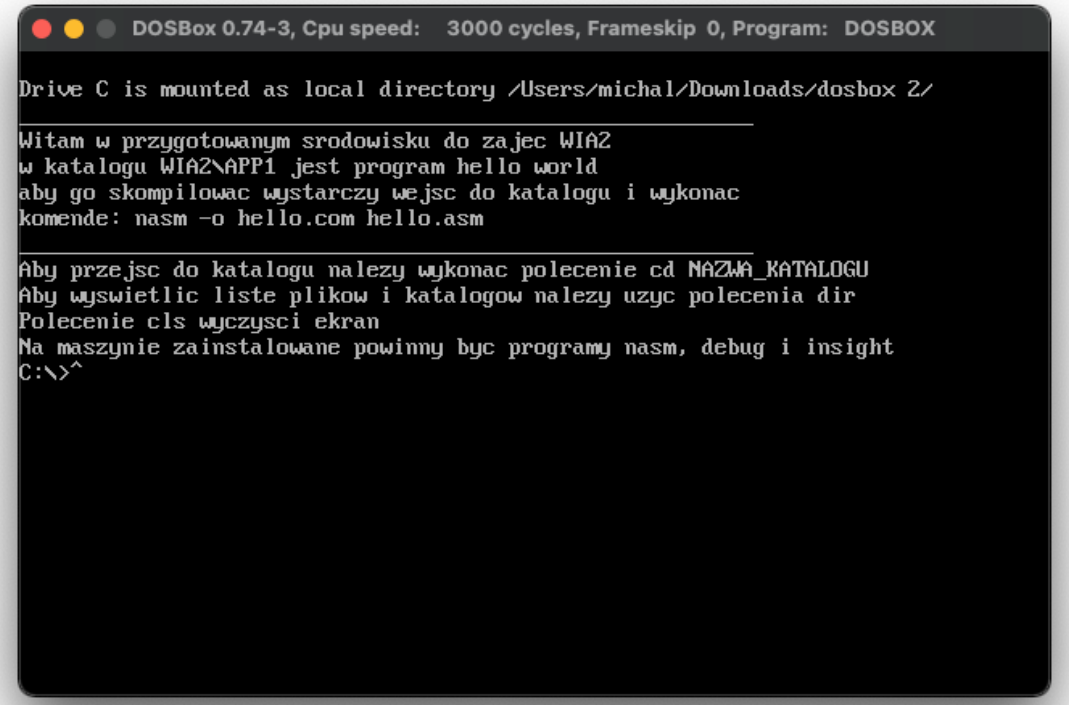
Wstęp do tego, jak sobie poradzić na zajęciach

Bez tego wszyscy polegniemy w pierwszych minutach zajęć. Niezależnie od wykorzystywanego środowiska, powinniśmy najpierw zainstalować, a następnie skonfigurować dosboxa. Kluczowe tu jest wskazanie lokalizacji, w której będzie znajdował się nasz „dysk c". Na potrzeby zajęć dzielę się gotową paczką – dysk z potrzebnymi programami dostępny tu – dosbox.zip. Tego zipa należy odpakować i umieścić gdzieś na dysku – na sprzęcie uczelnianym polecam dysk Z:. Pobrać należy również plik dosbox.conf. Zawartość pobranego pliku należy podmienić z zawartością domyślnego pliku konfiguracyjnego – w Windowsie dostępny z menu start jako DOSBox 0.74 (Options), na macOS w `/Users/[uzytkownik]/Library/Preferences/DOSBox 0.74-3 Preferences`.

W trzecim od końca wierszu pliku konfiguracyjnego należy wskazać ścieżkę do wypakowanego katalogu dosbox (zawierającego katalogi WIA2 i APP oraz plik autoexec.bat) i zapisać. Jeśli chcemy poczuć się jak nerd z lat 80., to alt+Enter przejdzie nam w tryb pełnoekranowy. Polecam również w pliku konfiguracyjnym ustawić:

`windowresolution=1600x800` oraz `output=ddraw` (`opengl` na linux I macOS)

Po uruchomieniu dosboxa naszym oczom powinno ukazać się to:



Wróć, jaki DOSBox?

Pracować będziemy na 16-bitowym systemie DOS, konkretniej w jego emulowanej wersji, stworzonej do odpalania starych gier na nowoczesnych komputerach. Sam asembler w wersji 32- lub 64-bitowej oczywiście różni się od tego, na czym będziemy pracować, natomiast koncepcja i sposób myślenia pozostaje ten sam, także przejście na wyższe wersje po opanowaniu tej powinno być względnie bezbolesne

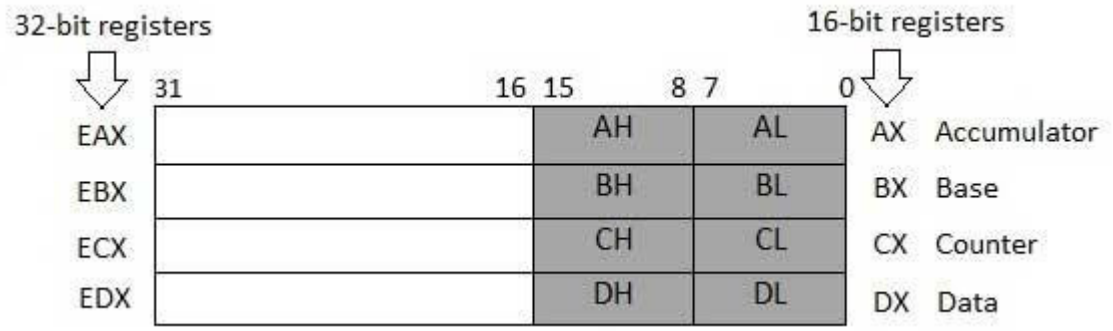
To teraz odrobinę dramatycznie suchej teorii

Procesor jaki jest każdy widzi. pracujemy na architekturze x86, jednak zasada zazwyczaj będzie taka sama. Procesor pobiera z pamięci rozkazy oraz dane i wykonuje je. Rejestr jest komórką pamięci na procesorze, do której ma on bezpośredni dostęp. Nie należy go mylić z pamięcią cache, która jest szybką pamięcią, do której procesor ma szyki dostęp. Rejestr traktujemy jako integralny element układu procesora. Rozróżniamy kilka podstawowych rodzajów rejestrów. Jakby ktoś się zastanawiał, skąd Citroen wziął nazwy modeli swoich samochodów w latach 70 i 80, to odpowiedź znajduje się poniżej.

Rejestry Danych

- AX – używany głównie do obliczeń, jeden z operandów przechowywany jest właśnie w AX • BX – Base Register – używany przy adresowaniu
- CX – Count register – Licznik. Nic ponad to.
- DX – Data Register – Rejestr danych, używany do operacji wejścia/wyjścia, czasem używany wspólnie z AX do większych operacji

Każdy z powyższych dzieli się na dwie 8- bitowe części AH i AL., BH i BL itd.



Rejestry wskaźnikowe

- IP – Instruction Pointer – Zawiera adres aktualnie wykonywanej operacji
- SP – Stack Pointer – wskaźnik stosu
- BP – Base Pointer – do adresowania pamięci

Rejestry indeksowe

- SI – Source Index – służy do indeksowania pamięci
- DI – Destinantion Index – jak powyżej, tylko w drugą stronę.

Rejestry Segmentowe

- CS – Code Segment – zawiera instrukcje do wykonania. Przechowuje adres początkowy segmentu kodu
- DS – Data Segment – zawiera dane. Przechowuje adres segmentu danych
- SS – Stack Segment – Segment stosu

Adresowanie

Pracować będziemy w trybie rzeczywistym. Jest to tryb pracy procesorów z lat 80 z rodziny x86. Tryb rzeczywisty nie zawiera zabezpieczeń pamięci przed użyciem przez inny proces, inaczej niż tryb chroniony. W trybie rzeczywistym dostępna jest 1-megabajtowa przestrzeń adresowa. Adres logiczny (programowy) składa się z dwóch liczb 16-bitowych: segmentu (numeru segmentu) oraz przemieszczenia względem początku segmentu (ang. *offset*). Adres fizyczny jest liczony jako *segment* * 16 + *przemieszczenie*. Ponieważ segmenty nie są rozłączne, wiele różnych adresów logicznych może odwoływać się do tej samej komórki pamięci (dokładnie – jeden adres fizyczny jest opisywany przez 4096 różnych adresów logicznych)[1]; na przykład:

```
   7400:1234 (zapis szesnastkowy), tak więc
segment = 7400h    offset  = 1234h

   74000h
    1234h
   -----
   75234h   - adres fizyczny.
```

Rozróżniamy różne tryby adresowania:

Adresowanie rejestrowe

```
mov ax, bx
```

Adresowanie bezpośrednia

```
mov ax, 1
```

Adresowanie pośrednie

```
mov ax, [102h] ; Adres końcowy to DS:0 + 102h
```

Adresowanie rejestrowe pośrednie

```
mov ax, [di] ; Adres komórki zawarty jest w rejestrze DI
```

Przerwania

Przerwania traktować możemy jako funkcję. Z technicznego punktu widzenia przerwanie jest sygnałem zmieniającym przepływ sterowania, niezależnie od wykonywanego programu. Najwięcej korzystać będziemy z przerwania 21h, zawierającego wiele interakcji sprzętowo-systemowych. Jeśli komuś nie przeszkadzają 25-letnie reklamy serwisów randkowych

to po pełną dokumentację zapraszam na najbardziej antyczną stronę w internecie: O tutaj.

Komendy

Komend oczywiście jest mnóstwo, ale głównie używać będziemy dwóch:

- `MOV {cel}, {źródło}` – Kopuje wartość źródła do celu.
- `INT` – Wywołanie przerwania

Segmenty

Segmentowy model pamięci, wykorzystywany właśnie w systemach DOS polega na tym, że program może mieć w pamięci operacyjnej 3 obszary: Kod, dane oraz stos. W segmencie kodu, jak można się domyślić przechowywany jest ~~stos~~ kod programu. W segmencie danych przechowywane są dane, a w stosie stos. Wymienione powyżej rejestry segmentowe będą właśnie przechowywać adresy w pamięci owych segmentów.