Nadal siedzimy w 16-bitowym trybie rzeczywistym, ale już niedługo

Dzisiaj posiedzimy chwilę w Bootsectorze na 16 bitach, konkretnie zajmiemy się wywoływaniem funkcji, drukowaniem stringów w formie w jakiej faktycznie powinniśmy to robić, spróbujemy się dostać do dysku, a potem się zobaczy

Najpierw jednak przyda nam się stos. Stos rozwiązywać nam będzie jeden, ale dość poważny problem – mamy za mało rejestrów ogólnego przeznaczenia, a nierzadko zdarzyć się może sytuacja, w której potrzebować będziemy trochę więcej przestrzeni, szczególnie, że na 16 bitach jest jej raczej niewiele. CPU ma dwie instrukcje, które obsłużą stos

push
pop

Odpowiednio, umieszczą nam one dane na szczycie stosu oraz zdejmą dane ze szczytu stosu. Stos jest zaimplementowany za pomocą dwóch rejestrów specjalnego przeznaczenia – bp i sp. Sp wskazywać nam będzie na szczyt stosu, tak, abyśmy zawsze odczytywali dane stamtąd. Bp ma trochę bardziej złożone działanie, na razie użyjemy go do ustawienia sp.

```
ORG 0x7C00

mov ah, 0Eh

mov bp, 0x8000
mov sp, bp

push 'A'
push 'B'
push 'C'

pop bx
mov al, bl
int 10h


times 510-($-$$) db 0
dw 0xaa55
```

W tym momencie warto wspomnieć o dwóch bardzo fajnych instrukcjach, które za chwilę bardzo nam się przydadzą: pusha i popa – one, odpowiednio wrzucą oraz zdejmą ze stosu wszystkie rejestry ogólnego przeznaczenia.

Funkcje

Czymże byłoby programowanie bez funkcji? Na poziomie CPU wywołanie funkcji to w zasadzie nic innego, jak skok do zadanego adresu oraz powrót po wykonaniu funkcji. W asm służyć do tego będą instrukcje call i ret. Tej pierwszej podajemy adres (etykietę) funkcji, którą chcemy wykonać, natomiast ta druga wróci wykonanie kodu na adres, z którego poszło wywołanie – tu też używany jest stos.

Do pliku asm możemy również załączyć inny, zewnętrzny plik. Posłuży do tego dyrektywa

```
%include nazwa_pliku.asm
```

Tak naprawdę zamieni nam to na etapie kompilacji dyrektywę na kod znajdujący się w zewnętrznym pliku. Możemy sobie w takim pliczku zamieścić zestaw instrukcji służący do drukowania np. stringa.

Stringi

Drukowanie stringów z dolarem na końcu było wywoływane przy pomocy przerwania DOS-a na WIA2. Teraz nie mamy takiego komfortu, ale cały czas możemy zapisać stringa w pamięci, tu się nie zmieniło absolutnie nic. Standardowym rozwiązaniem drukowania ciągu znaków przy pomocy jedynie BIOS-u będzie przypisanie adresu zdefiniowanej etykiety ze stringiem zakończonego zerem do rejestru BX, zawartości BX do AL i przy pomocy komparatora i skoków warunkowych za każdym razem, kiedy nie będzie znajdowało się tam '0' wydrukowanie znaku i zwiększenie BX, a gdy będzie znajdowało się tam '0' to zakończenie funkcji i powrót.

Zadanie 1

Wydrukuj na terminalu string „Booting HylaOS". Ale string, a nie literka po literce. To nam się przyda w przyszłości, więc warto zachować ten kod.

Dostęp do dysku

To może być niespodzianką, ale Systemy operacyjne zazwyczaj są odrobinę większe niż dostępne w bootsectorze 512 bajtów. Generalnie procesor nie wie, jak odczytywać jakiekolwiek dane z dysku, ale – na szczęście – BIOS to potrafi, dlatego go użyjemy. Dzisiaj wczytamy sobie do pamięci dane z dysku, posłuży nam do tego przerwanie 13h/AH=02h.

To konkretne przerwanie wymaga od nas jednak ustawienia kilku dodatkowych rejestrów, na których określimy, skąd konkretnie chcemy odczytywać dane.

```
mov ah, 02h    ; do przerwania

mov al, 02h    ; liczba sektorów do odczytania

mov dl, 0      ;odczyt z dysku 0 – to jest zależne od systemu –
0 i 1 to odczyt z dyskietek 0 i 1, 0x80 – hdd1, 0x81 – hdd2

mov cl, 02h    ; sektor – 0x01 to bootsector, 0x02 to pierwszy
dostępny sektor

mov ch, 00h    ; cylinder

mov dh, 00h    ; głowica

int 13h        ; przerwanie
```

Takie dane zostaną zapisane pod adresem wskazywanym przez [es:bx].

Teraz ważna sprawa: zawsze może coś nie pójść, a errorhandling to w tym wypadku dość ważna sprawa, jak nigdy, bo nie dostaniemy żadnej zwrotki o błędnym odczycie. Na szczęście zastosowane przerwanie ustawia nam w wypadku błędu CF (carry flag). Dlatego dobrym rozwiązaniem jest sprawdzenie czy dane zostały prawidłowo wczytane poprzez skontrolowanie flagi. Skok warunkowy

```
jc error
```

wykona skok do etykiety error w sytuacji, w której flaga jest ustawiona. Jeśli nie jest, to pójdzie dalej.

Zadanie 2

Napisz bootsector, który będzie wczytywał dane z dysku i będzie zawierał porządny error handling w postaci wydruku stringa o błędzie, gdy CF będzie ustawiony. Po przerwaniu 13h rejestr AL będzie zawierał informację o ilości odczytanych sektorów – tu też przydałoby się porównanie, czy faktycznie po wywołaniu przerwania odczytano tyle sektorów, ile kazaliśmy i ewentualne info o błędzie.