Odczyt z dysku, jądro, takie tam

Generalnie odczyt danych z dysku już raz robiliśmy, ale tym razem może się uda. Pracujemy jeszcze na 16 bitach, więc nadal mamy dostęp do przerwań, dlatego posłużymy się int 13h.

Qemu wywołane z opcją -fda ustawi nam rejestr dl na 0x00. DL w przerwaniu 13h/AH=02h określa numer dysku do odczytu. W teorii, jeśli ustawimy -boot c, to dl powinien zostać ustawiony na 0x80, ale tego mi się nie udało doprowadzić do działania, więc na razie odczytujemy z dyskietki. Potem już prosto:

- pusha
- AH = 02h (żeby przerwanie zadziałało)
- AL = 02h (2 sektory)
- CL = 02h (od którego sektora zaczynamy, 0x01 jest zajęty przez bootsector)
- CH = 00h (cylinder)
- DL powinien już być ustawiony
- DH = 00h (głowica)
- Przerwanie 13h
- Popa
- Ret

Teraz tak, wszystkie dane wczytywane są w bufor określony przez parę es:bx. ES się nie przejmujemy, ale BX przydałoby się ustawić, przy okazji tak, żeby nam się nic przy odczycie nie władowało w stos (a raczej, żeby stos się nie władował w to co wczytamy), więc np. jeśli ustawiamy w naszym bootsektorze stos na 0x8000, to wczytajmy dane na adres 0x9000.

Moja funkcja drukująca w 16 bitach przyjmuje adres początku stringa w rejestrze BX, więc przed wywołaniem print_string ustawiam BX na adres 0x9000.

Jądro systemu

Pisanie w C jest superprzyjemne, mamy przecież zarządzanie pamięcią, możemy pisać aplikacje wielowątkowe, podlinkować fantastyczne biblioteki. Przyjemność kończy się w momencie, w którym zorientujemy się, że w naszym systemie nie mamy absolutnie nic – nawet głupiego printa nie możemy zrobić, tylko musimy pracować z vga. Zacząć jednak musimy od wrzucenia do naszego bootsectora procedury, która pozwoli jądro załadować – po to nam było ładowanie danych z dysku.

Przed switchem na 32 bity należy wywołać procedurę, która załaduje jądro pod adres 0x1000 (dla bezpieczeństwa ze 4 albo 15 sektorów). Następnym krokiem jest napisanie samego jądra. Jeśli chcielibyśmy uzyskać jakieś potwierdzenie, że odczyt jądra się udał i teraz się ono wykonuje, to najprościej będzie umieścić w pamięci VGA jednego znaku. Jak się wyświetli to działa, jak nie, to nie.

Kernel.c:

```
void main(){
char* vid_mem = (char* 0xb8000);
*vid_mem = ‘A’;

}
```

I teraz tak: ładujemy jądro do postawionego systemu 32-bitowego, ale kompilować je będziemy na systemie 64-bitowym, 64-bitowym kompilatorem do 64-bitowego outputu. Żeby to zobaczyć proponuję wykonać polecenie:

```
Gcc -ffreestanding -c kernel.c -o kernel.o
```

A następnie

```
objdump -d kernel.o
```

Fajnie wygląda, natomiast nie mamy w naszym systemie dostępu do 64-bitowych rejestrów, więc przydałoby się to zmontować do 32 bitowego formatu. Jest niezerowa szansa, że powinniśmy przygotować cross-compilera, ale spróbujemy na razie sobie poradzić bez niego.

```
Gcc -ffreestanding -m32 -c kernel.c kernel.o
```

Puszczenie objdumpa pokaże, że teraz faktycznie mamy format elf_i386.