Zadania - dodatkowe informajce

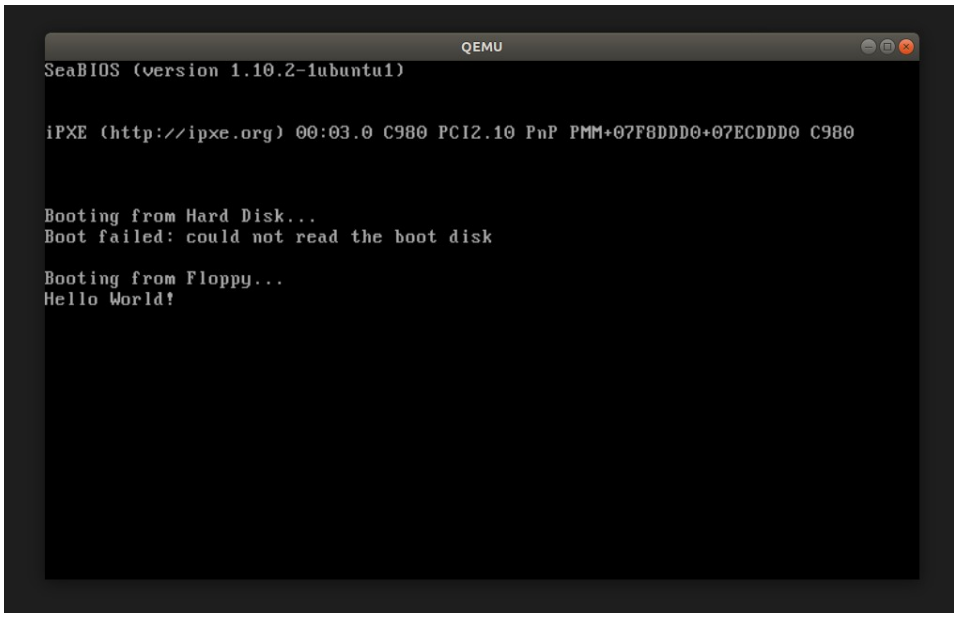
Aby skompilować i uruchomić zadania należy uruchomić skrypt *run.sh* znajdujący się w floderze z zadaniem.

ex1:

Pierwsze zadanie polega na wypisaniu na ekran łańcucha znaków "Hello World!". Należy to zrobić przy pomocy dwóch funkcji - printString(...) oraz printExclamationMark(). W pierwszej z nich użyta powinna zostać rozszerzona wersja asm(...) inline asembly, natomiast w drugiej wystarczy użyć wersji podstawowej.

- Do wypełnienia obu funkcji, niezbędne będą przerwania BIOS-u. Na stronie pod linkiem znajduje się "Interrupt Jump Table", w której można sprawdzić co powodują dane przerwania (w tym zadaniu najistotniejsze będzie przerwanie INT 10).
- Zawartośc funkcji main() powinna pozostać niezmienna.

Oto jak powinien wyglądać output:

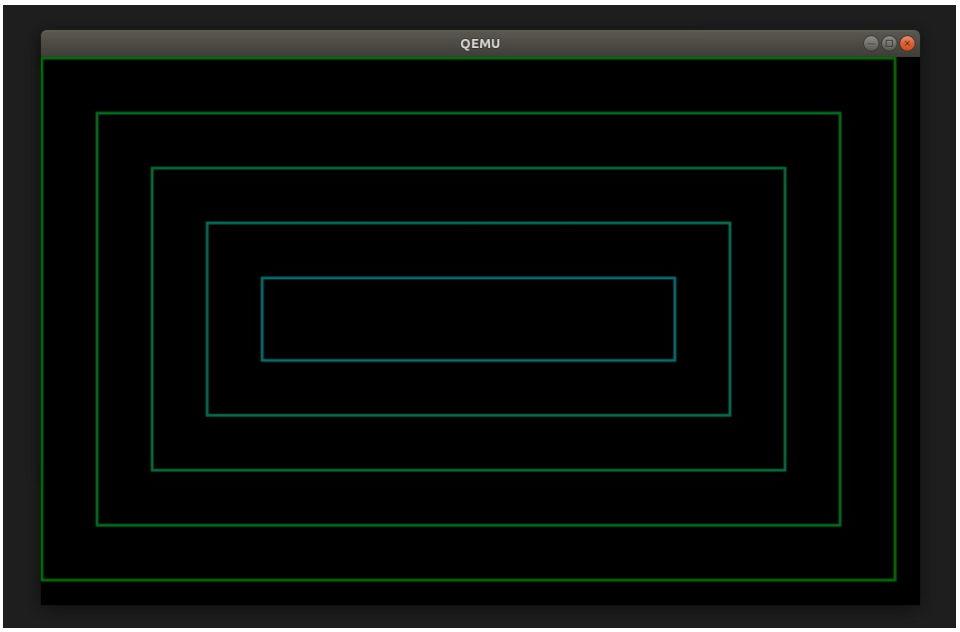


ex2:

Drugie zadanie polega na wyrysowaniu na ekran prostego rysunku. Należy to zrobić przy pomocy dwóch funkcji - setVideoMode(...) oraz draw(). Pierwsza z nich, jak sama nazwa wskazuje, odpowiedzialna jest za ustawienie w bootloaderze trybu video, natomiast zadaniem drugiej jest rysowanie.

- To co wyrysuje program nie ma większego znaczenia . Zadanie ma pokazać jak ustawić i poruszać się w trybie video bootloadera, a nie sprawdzić komu uda się stworzyć najładniejszy rysunek.
- Należy zwrócić uwagę na ograniczoną liczbę pamięci.
- Zadanie można sobie urozmaicić zmieniając kolor rysowanej figury.
- Również w tym zadaniu przydatna będzie tablica przerwań udostępniona w zadaniu pierwszym (również przerwanie INT 10).
- Istnieje wiele trybów video, lecz nie wszystkie działają, polecam wykorzystać tryb o numerze 13.
- Zawartośc funkcji main() powinna pozostać niezmienna.

Oto przykładowy output:



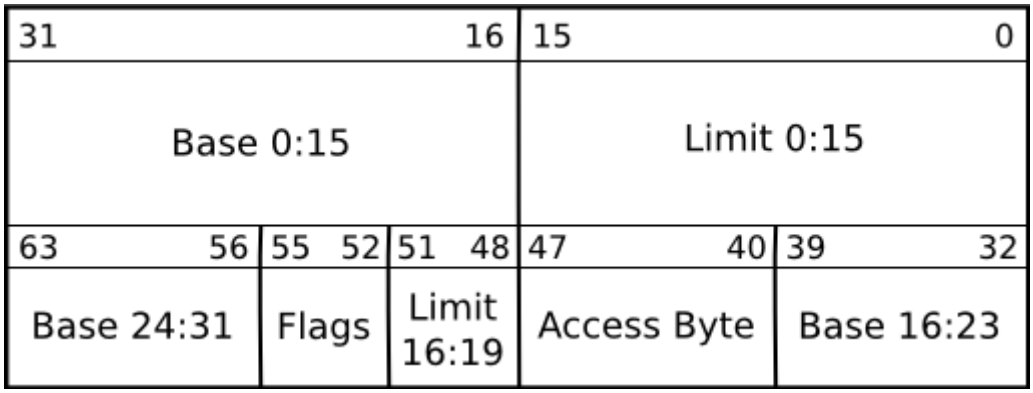
ex3:

Ostatnie zadanie polega na napisaniu mixed-stage bootloadera w języku asemblera, który zmieni tryb pracy procesora z rzeczywistego na chroniony.
Szkielet programu napisany został w języku NASM. W miejscach, w których należy uzupełnić program widnieją komentarze, które mówią co w danym miejscu powinno się znaleźć.
Moja prezentacja nie zawiera pełnych informacji dotyczących tego zadania, wobec czego zamieszczam dodatkowe informacje, które mogą okazać się pomocne w jego rozwiązaniu:

Zanim zmienimy tryb pracy procesora na rzeczywisty musimy wykonać 3 kroki:

- włączenie linii A20 - umożliwia to dostęp do pamięci adresowalnej, która w trybie rzeczywistym jest domyślnie nieosiągalna (4GB zamiast 1MB). Krok ten możemy wykonać używając przerwania biosu (INT 15 - znów odsyłam do tabeli przerwań).

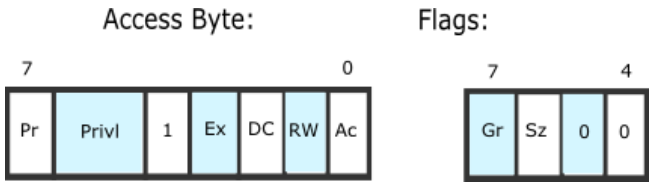
- ustawienie tabeli GDT - Global Description Table. Zawiera informacje dotyczące segmentów pamięci, potrzebne procesorowi do poprawnego działania. Szczegółowe informacje tutaj. W naszym programie zaimplementowane jest jej przekazywanie, natomiast należy ją jeszcze wypełnić. Należy to zrobić według określonego schematu. GDT składa się z dwóch części: code segment i data segment. Wypełnia się je niemal identycznie.



Offset (bits)	Name	Meaning
0..15	Limit	Lower 4 bytes of the descriptor's limit
16..31	Base	Lower 4 bytes of the descriptor's base address
32..39	Base	Middle 2 bytes of the descriptor's base address
40..47	Access byte	A group of bit flags defining who has access to the memory referenced by this descriptor
48..51	Limit	Upper 4 bits of the descriptor's limit
52..55	Flags	Four flags influencing segment size
56..63	Base	Upper 2 bytes of the descriptor's base address

Diagramy przedstawiają znaczenie bitów w GDT (zarówno code segment jak i data segment). Kolejne nazwy oznaczają:

- Base - zawiera adres początku segmentu do którego deskryptor się odnosi. Wszędzie ustawiamy na 0.
- Limit - zawiera rozmiar bloku pamięci do którego deskryptor się odnosi. Ustawiamy maksymalną wartość.
- Acces Byte i Limit - układ bitów i ich znaczenie:



Bit	Name	Description
Pr	*Present*	Selectors can be marked as “not present” so they can’t be used. Normally, set it to 1.
Privl	*Privilege level*	There are four privilege levels of which only levels 0 and 3 are relevant to us. Code running at level 0 (kernel code) has full privileges to all processor instructions, while code with level 3 has access to a limited set (user programs). This is relevant when the memory referenced by the descriptor contains executable code.
Ex	*Executable*	If set to 1, the contents of the memory area are executable code. If 0, the memory contains data that cannot be executed.
DC	*Direction (code segments)*	A value of 1 indicates that the code can be executed from a lower privilege level. If 0, the code can only be executed from the privilege level indicated in the Privl flag.
DC	*Conforming (data segments)*	A value of 1 indicates that the segment grows down, while a value of 0 indicates that it grows up. If a segment grows down, then the offset has to be greater than the base. You would normally set this to 0.
RW	*Readable (code segments)*	If set to 1, then the contents of the memory can be read. It is never allowed to write to a code segment.
RW	*Writable (data segments)*	If set to 1, then the contents of the memory can be written to. It is always allowed to read from a data segment.
Ac	*Accessed*	The CPU will set this to 1 when the segment is accessed. Initially, set to 0.
Gr	*Granularity*	For a value of 0, the descriptor’s limit is specified in bytes. For a value of 1, it is specified in blocks (pages) of 4 KB. This is what you would normally want if you want to access the full 4GB of memory.
Sz	*Size*	If set to 0, then the selector defines 16-bit protected mode (80286-style). A value of 1 defines 32-bit protected mode. This is what we want.

Privilege level - ustawiamy na 0.
Executable - jedyny bit który ustawiamy inaczej dla code i data segment! W pierwszym przypadku jest to 1, a w drugim 0.
Direction/Conforming - ustawiamy na 0
Readable/Writable - ustawiamy na 1
Reszta według tabeli powyżej.

- wyłączenie przerwań - Najprostszy z 3 kroków. Wystarczy wyczyścić flagę przerwań. Można to zrobić używając słowa kluczowego "cli" (clear interrupt flag).

Po wykonaniu tych kroków, procesor może przejść w tryb chroniony. Wykonujemy to ustawiając bit PE znajdujący się w rejestrze CR0 na 1. W tym celu wykonujemy następujące instrukcje:

```
mov    eax, cr0
or     eax, 1
mov    cr0, eax
```

W celu sprawdzenia, czy program działa prawidłowo, wypisuje on na końcu "Hello from 2nd stage bootloader". Znów więc ustawiamy tryb video (tak jak w zadaniu 2. lecz tym razem polecam tryb o numerze 2). Dodatkowo warto spojrzeć na zawartość wynikowego pliku binarnego. Pierwsze kilkadziesiąt bajtów powinno być wypełnione danymi(1st stage bootloader), a kolejna część (do 512 bajtu) zerami. Ostatnie bajty to dane z 2nd stage bootloadera.

W razie jakichkolwiek wątpliwości proszę o kontakt, a postaram się je wszystkie rozwiać.