# Practicum 3

# De onderbrekingsroutine

In dit practicum zullen we de onderbrekingsroutine onder de loep nemen. Vertrekkend van een stukje assemblercode, zullen we een onderbrekingsroutine toevoegen voor het toetsenbord.

## 3.1 Inleiding

Gegeven het programma `keyboard.asm`. Dit programma zal worden geassembleerd en gelinkt tot een programma dat op de bootsector van een floppy of cdrom kan geschreven worden. Bij het opstarten van de (gesimuleerde) computer gaat de controle over naar dit programma en hebben we complete toegang tot alle onderdelen van de processor (omdat we ons op dat ogenblik in kernelmode bevinden).

De bootsector bestaat uit de eerste sector van de floppy of cdrom. Deze sector zal de volgende sectoren inlezen en het bijhorende programma uitvoeren. Om als bootsector herkend te worden, moet de eerste sector eindigen met het bitpatroon `55AA`. De eerste sector van de floppy of cdrom wordt door de BIOS automatisch geladen op adres `7c00h`. We laten de stapel vanaf adres `7c00h` groeien naar adres `0` toe – we volgen hierin de normale conventie waarbij de stapel op de x86 groeit naar de lagere adressen.

Nadat alle sectoren ingeladen zijn, wordt er omgeschakeld naar protected mode en wordt er begonnen met de uitvoering van 32-bit code.

Daarna wordt het hoofdprogramma uitgevoerd, dat bestaat uit het herhaald controlleren van alle registerwaarden. Ieder register heeft in dit controleprogramma een vooraf gedefinieerde constante inhoud die we in een oneindige lus zullen controlleren. Je interrupt handlers mogen namelijk nooit de staat van het programma aanpassen, aangezien ze op gelijk welk moment in de uitvoering van het programma kunnen optreden. Wordt de staat toch aangepast, zal het hoofdprogramma een foutmelding geven. Dit betekent dat je in de onderbrekingsroutines alle gebruikte registers zult moeten bewaren en herstellen.

Om het practicum te kunnen uitvoeren beschik je over de volgende bestanden:

- `keyboard.asm` : het hierboven beschreven programma
- `k.sh` : script dat keyboard.asm assembleert en omzet in MyBoot.bin
- `b.sh` : script dat de bochs emulator opstart met het bestand MyBoot.bin

Voor dit practicum zullen we ons niet belasten met het herhaaldelijk herstarten van de computer vanaf een floppy of cdrom. We maken gebruiken van een een emulator (bochs) die een Intel Pentium emuleert. In het configuratiebestand van bochs (`bochs.conf`) staat dat de inhoud van een floppy opgeslagen ligt in `MyBoot.bin`. Op die manier zal de emulator booten van `MyBoot.bin`. Let op!

Het gebeurt frequent dat men nog de oude `MyBoot.bin` gebruikt, omdat er iets faalt tijdens het compileren van `keyboard.asm`.

Je editeert `keyboard.asm` met jouw favoriete editor (b.v. met vim, emacs :), gedit, nano,... ), en na bewaard te hebben voer je in de terminal in de directory van dit practicum het commando `./k.sh` uit om het programma te assembleren, en nadien `./b.sh` om het in bochs uit te voeren.

Nostalgische mensen kunnen hier ook echt een bootfloppy van te maken. Dat kun je doen door het commando `sudo dd if=MyBoot.bin of=/dev/fd0` uit te voeren. Je kunt de computer vervolgens van die floppy booten. Je kan ook het bestand MyBoot.bin op een cdrom branden als *Bootable cd* en de computer vervolgens van cdrom booten. Je kunt je dan vergewissen van het feit dat deze bootfloppy of bootcdrom ook in het echt werkt.

Het bestand `MyBoot.bin` kun je indien gewenst disassembleren met het programma `ndisasm`. Je moet dit wel in twee keer doen: met de vlag `-b 16` voor de eerste sector en met de vlag `-b 32` voor de overige sectoren.

## 3.2 Voorbereiding

Bestudeer de listing met assemblerinstructies. Let op! Hier wordt een variant van de Intel-syntax gebruikt. Het eerste deel van de code (bootsector) moet je niet in detail begrijpen; lees echter wel het commentaar in de code. Probeer zeker de code vanaf *second stage section* tot aan het gedeelte met hulpfuncties door te nemen *voor* aanvang van het practicum. Zaken die je niet begrijpt kan je natuurlijk vragen tijdens het practicum zelf.

Het is daarenboven sterk aanbevolen om in de cursus het gedeelte over onderbrekingen en vooral het bijhorend voorbeeld i.v.m. de printeronderbreking grondig te bekijken.

Een handige assemblerinstructie die je kan gebruiken is de `ud2` instructie. Deze instructie zal er voor zorgen dat je processor een fout genereert. Je kan dit gebruiken bij het debuggen. Als je niet zeker bent of een functie opgeroepen wordt (dit kan bijvoorbeeld een interrupthandler zijn), plaats dan deze instructie eens in het begin van deze functie: als de emulator een fout geeft als deze instructie in je code zit, en als er geen fout optreedt als deze instructie niét in je code zit, kan je er zeker van zijn dat ze opgeroepen wordt. Een andere manier van debuggen is natuurlijk het gebruik van de debugger met breakpoints en het stappen over instructies. Hierbij is het handig om weten dat (als je niet aan de boot code zelf komt), je eenvoudig de debugger kan laten stoppen op de eerste instructie van de `main`-code als volgt: `break 0x7fac`.

## 3.3 Opgaven

1. Installeer de toetsenbordhandler. Gebruik hiervoor de routine install_handler. Ga na op welke manier je het nummer van de vector en het adres van de handler moet meegeven. Op welk vectornummer moet de toetsenbordhandler geïnstalleerd worden?

2. Pas het onderbrekingsmasker in poort 21h aan zodat de toetsenbordonderbreking aangeschakeld wordt. Welke bit in het masker moet hiervoor aangepast worden?

3. In poort 60h kan je de scancode van de ingedrukte toets ophalen. Schrijf de ingelezen scancode op het scherm aan de hand van de routine printscancode. Let erop dat de werking van het hoofdprogramma niet verstoord wordt door het tonen van de scancode, m.a.w. zorg ervoor dat de registers goed bewaard en hersteld worden (tip: gebruik pushad en popad om de registers te bewaren en terug te herstellen).

4. Pas je code aan zodat de scancode voor elke nieuw ingedrukte toets verschijnt. Hiervoor moet je de onderbrekingsregelaar laten weten dat hij nieuwe onderbrekingen mag doorsturen en zet de onderbrekingen nadien aan.

5. Bekijk de waarden van de make en break codes voor een aantal toetsen (make code = code bij het induwen van de toets, break code = code bij het loslaten van de toets). Wat is de relatie tussen de make en break codes? Hoe codeert men de shift toets? Hoe kan een onderscheid tussen kleine letters en hoofdletters gemaakt worden?

6. Maak nu gebruik van 4 make codes om een kruisje over het scherm te laten bewegen. Je kan niet zomaar karakters op het scherm laten bewegen, enkel karakters overschrijven. Als je dus een karakter van plaats wil veranderen, zal je dus de oorspronkelijke plaats moeten leeg maken, waarna je het karakter schrijft op zijn nieuwe plaats.

   Voeg de code hiervoor toe in toetsenbordhandler, waarbij je gebruik kan maken van variabelen (dit zijn geheugenlocaties met labels waarnaar je kunt verwijzen) die wij reeds aangemaakt hebben, zoals `huidige_positie_x` en `huidige_positie_y`. Gebruik voor het bewegen van je kruis de de getallen op het numerieke klavier (niet van de pijltjestoetsen).

## 3.4 Facultatieve opgaven

- Kun je ervoor zorgen dat door het indrukken van de `r`-toets, naast het tonen van de bijhorende scancode, ook de voornaamste registers getoond worden (met "call printad")? Let er opnieuw op dat de registers goed bewaard en hersteld worden. Ga na op welke manier de uit te printen registers moeten worden meegegeven aan printad. Zorg ervoor dat als instructiewijzer het terugkeeradres van het hoofdprogramma getoond wordt. Waar wordt deze op de stapel bewaard?

- Wat merk je als je het verloop van de instructiewijzer bekijkt (druk hiervoor enige keren op de `r`-toets)?

- Kun je de timerhandler gebruiken om de waarde van de registers automatisch op het scherm bij te werken? Je kunt de `r`-toets nu gebruiken om het actualiseren aan en af te zetten. Het hoofdprogramma moet gedurende al deze operaties gewoon verder blijven werken.

Succes!