Practicum 1

Debuggen op machineniveau

1.1 Inleiding

Het doel van dit practicum is om vertrouwd te raken met assemblercode en de debugger. We beschouwen een eenvoudig C-programma: `jacobsthal.c`. Dit programma genereert Jacobsthal-getallen. De Jacobsthal-getallen worden analoog gedefiniëerd als de wellicht beter bekende Fibonacci-getallen.

$$\begin{cases} Jacobsthal(0) &= 0 \\ Jacobsthal(1) &= 1 \\ Jacobsthal(n) &= Jacobsthal(n-1) + 2 * Jacobsthal(n-2) \quad ,\forall n \geq 2 \end{cases} \tag{1.1}$$

Gebruikmakend van de make-tool en gcc (compiler), zullen we de C-code compileren (.c file) naar assemblervorm. Vervolgens wordt de mnemonische assemblercode (.s file) getransformeerd naar de eigenlijke machinecode in binair formaat, in een zogenaamd objectbestand (.o file). Tot slot worden al deze objectbestanden samen gelinkt aan de noodzakelijke systeembibliotheken om tot een uitvoerbaar programma te komen.

Alle benodigde bestanden voor dit practicum bevinden zich in de `pract01` submap van de `computerarchitectuur_practica` map. We zullen naar deze map gaan in een terminal. Dit doe je als volgt: ga in het hoofdmenu linksbovenaan naar *Applications*, *Accessories*, en start *Terminal* op. In deze terminal kan je vanuit de home-directory van de gebruiker (deze map wordt aangegeven met een tilde: `~`) gaan naar de `pract01` map met het change-directory commando `cd` als volgt:
`cd computerarchitectuur_practica/pract01`

Om de code te compileren, moet je gebruik maken van de bijgeleverde makefile. Om `jacobsthal.c` te compileren typ je `make pract01` in op de commandolijn (in de `pract01` map). Om alle bestaande binaire bestanden te verwijderen kan je `make clean` uitvoeren. Het programma wordt op de commandolijn opgeroepen (`./jacobsthal arg1 arg2` – let op de "./" in het begin) met twee argumenten: de index van het Jacobsthal-getal dat berekend moet worden en het aantal keren dat deze berekening wordt uitgevoerd. Dit aantal iteraties kan je voor het eerste practicum steeds op 1 laten staan. Verifieer eerst en vooral dat de omgeving correct is opgezet door de code te compileren en het programma uit te voeren.

De code die door de gcc-compiler geproduceerd wordt, kan je terugvinden in `jacobsthal.s` (wordt gegenereerd tijdens `make pract01`). Je kan dit bestand openen met de meegeleverde tekst-editor, deze open je op een gelijkaardige manier als de Terminal: start via *Applications*, *Accessories* het programma *gedit Text Editor* op. Breng eventueel een aantal afgedrukken van `jacobsthal.s` mee naar het practicum.

1.2 Voorbereiding

1. Bestudeer de C-code van de recursieve implementatie. Bestudeer ook de bijhorende assemblercode (gegenereerd door `make pract01` uit te voeren). Zorg dat je van elke instructie in de code weet wat ze doet en hoe ze werkt. Ga eventueel op het internet op zoek naar informatie over gebruikte instructies die je niet kent. We verwachten dat je minstens alle gebruikte instructies kent.

2. Teken de controleverloopgraaf van de functie `jacobsthal`; let hierbij goed op de afgesproken conventies (zie oefeningenles assembler).

We verwachten dat je deze voorbereidingen hebt gemaakt en zullen steeksproefsgewijs de voorbereiding controleren tijdens het practicum. Doe dit voor uzelf; zonder een goede voorbereiding gaat er veel kostbare tijd verloren tijdens de practica, tijd die beter gebruikt kan worden om interessante vragen te stellen.

1.3 Opgaven

1. Voer het programma uit in de emacs-omgeving (zie handleiding). Kies als argumenten 9 en 1. Stap doorheen de code met de debugger en maak uzelf vertrouwd met de code en de emacs-omgeving. Besteed hier voldoende tijd aan. Tracht op zijn minst de volgende opdrachten uit te voeren:

   - Maak uzelf vertrouwd met de basisfunctionaliteit van een debugger. Leer breakpoints te zetten, de inhoud van registers/geheugen bekijken en doorheen de code te stappen. Stap bijvoorbeeld doorheen de main-functie zonder in de Jacobsthal-functie te stappen, verander de inhoud van een register en observeer de invloed op het programmaverloop, ...
   - Beschrijf het gedrag van de `leave` en `ret` instructies. Welke registers worden aangepast? Waarom?
   - Gebruik de stackpointer om het verloop van de stapel op- en afbouw te bestuderen.
   - Wat is de inhoud van register eax op het moment dat jacobsthal voor de 3e keer wordt opgeroepen? Beantwoord dezelfde vraag voor de 13e keer dat jacobsthal wordt opgeroepen. Zoek een efficiënte manier om dit te doen.
   - Met een debugger kan je niet alleen de inhoud van registers en variabelen bekijken, je kan die ook aanpassen. Gebruik de debugger om ervoor te zorgen dat de waarde die op het scherm wordt getoond voor jacobsthal(9) niet het 9^{de} Jacobsthal-getal is, maar het 12^{de}. Voer deze hack uit door at runtime de toestand van het programma aan te passen. Probeer deze oefening op een aantal verschillende manieren op te lossen.
   - Hoeveel instructies zullen er uitgevoerd worden door jacobsthal(12)? Beschrijf hoe je deze waarde hebt bepaald.

2. Teken de stapel op het ogenblik dat de instructie `jmp .L3` in de functie `jacobsthal(3)` (bij het oproepen van `jacobsthal` met de argumenten `3 1`) voor de derde keer uitgevoerd wordt. Geef nauwkeurig aan wat de betekenis is van de inhoud van de stapel, bijvoorbeeld *terugkeeradres main* en dus niet (enkel) de waarde die af te lezen is uit de emacs-omgeving. Denk goed na over het controleverloop van de recursieve functie: door eerst na te gaan in welke volgorde de verschillende oproepen naar `jacobsthal` gebeuren, kan je jezelf heel wat werk besparen. Maak deze oefening niet op papier, maar in een tekstverwerker of spreadsheet, dat werkt veel gemakkelijker.

3. In de opgavedirectory (*pract01*) staan twee kleine programma's die een paswoordcontrole uitvoeren. Gebruik jullie kennis van de debugger om het paswoord te achterhalen of om de beveiliging te omzeilen. Er zijn meerdere oplossingen mogelijk, probeer een verschillende techniek voor de twee programma's.