Practicum 1

Debuggen op machineniveau

1.1 Inleiding

In dit practicum is het de bedoeling dat je vertrouwd raakt met assemblercode alsook met de debugger. We beschouwen een eenvoudige C-programma: `lucas.c`. Dit programma genereert Lucas-getallen, de Lucas-getallen worden analoog gedefiniëerd als de wellicht beter bekende Fibonacci-getallen.

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

Gebruik makend van de make-tool en gcc (compiler) zullen we de C-code compileren (.c file) naar de assembler-vorm, om tot slot de mnemonische assemblercode (.s file) te transformeren naar de eigenlijke machinecode in binair formaat en het programma te linken aan de noodzakelijke systeembibliotheken om tot een uitvoerbaar bestand te komen.

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

De code die door de gcc-compiler geproduceerd wordt, kan je terugvinden in `lucas.s` (wordt gegenereerd bij compilatie). Breng eventueel een aantal afgedrukken van `lucas.s` mee naar het practicum.

1.2 Voorbereiding

1. Bestudeer de C-code van de zowel de iteratieve als 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 doen en hoe ze werken. Wees vooral niet bang om op het internet op zoek te gaan naar informatie over de gebruikte instructies die je mogelijks niet kent. We verwachten dat je minstens alle gebruikte instructies kent.

2. Teken de controleverloopgraaf van de functie `lucas` en ; 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 dat 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 argument 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 lucas-functie te stappen, verander de inhoud van een register en observeer de invloed op het programmaverloop, ...
   - Beschrijf het gedrag van de `leave` en `ret` instructie. 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 lucas voor de 3e keer wordt opgeroepen? Beantwoord dezelfde vraag voor de 13e keer dat lucas 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 lucas(9) niet het 9^{de} Lucas-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 lucas(12)? Beschrijf hoe je deze waarde hebt bepaald.

2. Teken de stapel op het ogenblik dat de instructie `jmp .L12` in de functie `lucas(3)` (bij het oproepen van `lucas` 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 `pell` gebeuren, kan u uzelf heel wat werk besparen. Maak deze oefening niet op papier, maar in een tekstverwerker of spreadsheet, dat werkt veel gemakkelijker.

3. In de opgaven 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 beide programma's.