# Practicum 2

# Optimaliseren

## 2.1 Inleiding

Dit practicum bouwt verder op het eerste practicum: we vertrekken van de uitvoer van de compiler (`jacobsthal.s`). Als je de assemblercode aandachtig bestudeerd hebt, zul je wellicht gemerkt hebben dat de gegenereerde code verre van efficiënt is, laat staan dat ze optimaal is. In dit practicum is het de bedoeling om deze code met de hand te optimaliseren. Je kunt hiervoor de assemblercode bewerken en vervolgens het resulterende bestand omvormen tot een uitvoerbaar bestand met de commando's die verder worden gegeven.

## 2.2 Indienen

Dit practicum wordt gequoteerd. Je moet zowel de verschillende versies van je code als een verslag indienen vóór **26 maart 2014, 22:00:00** via `https://indianio.ugent.be`. Vergeet niet dat de systeemklok van indianio ongeveer 15 minuten kan voorlopen!

Let op! We verwachten per groep één *enkel* zip-bestand waarin de onderstaande bestanden zijn opgenomen. Beslis onderling wie het bestand zal indienen, dit hoeft niet voor ieder practicum dezelfde persoon te zijn.

Uw zip-bestand bevat de volgende bestanden:

- `practicum2_verslag.pdf`
- `practicum2_vraag2.s`
- `practicum2_vraag3.s`
- `practicum2_vraag4_opt_1.s` tot `practicum2_vraag4_opt_`$k$`.s`, waarbij $k$ het aantal versies is die je voor vraag 5 hebt geïmplementeerd.
- `practicum2_vraag4_opt.s`
- `practicum2_vraag8.s`

**BELANGRIJK. Alle .s bestanden mogen enkel de code van je jacobsthal functie bevatten, dus niet de volledige assember! Op de eerste regel moet de eerste instructies van de jacobsthal functie staan, inclusief het label (jacobsthal:).**
**Opgelet! De jacobsthal functie moet voldoen aan de standaard i386 oproepconventies (zie slides). Indien niet aan deze conventies voldaan wordt, kunnen we jullie code niet testen, en kunnen we ze dus ook niet quoteren.**

Voor vragen die je niet wil/kan oplossen, dien je lege bestanden in. Gebruik geen submappen, hoofdletters, andere bestandsformaten, enz. We gebruiken een geautomatiseerd systeem om jouw code te evalueren en het verslag te verwerken. Dat betekent dat enkel zipbestanden die strikt aan dit formaat voldoen, worden verbeterd.

## 2.3 Voorbereiding

Tijdens dit practicum zal je (grote) stukken van de functie jacobsthal(int) herschrijven. *Practicum 1: Debuggen op machineniveau* dient als voorbereiding voor dit practicum. Het is uitermate belangrijk dat je reeds vertrouwd bent met de code en goed weet wat de betekenis is van elke instructie.

Om de prestatie van een programma te meten, wordt vaak de uitvoeringstijd gebruikt. De uitvoeringstijd van eenzelfde programma kan echter (sterk) variëren tussen verschillende uitvoeringen, onder andere omdat de applicatie niet het enige proces is op de computer. Bovendien zijn moderne processors dermate complex en geavanceerd dat het ook niet altijd direct duidelijk is wat de relatie is tussen een optimalisatie en de resulterende versnelling. Allerhande factoren — die slechts duidelijk zullen zijn na deze cursus (en de vervolgcursussen) — maken het zeer moeilijk voor jullie om een optimalisatie te verifiëren gebruik makende van de gemeten uitvoeringstijd. Daarom zullen we tijdens dit practicum een vereenvoudigd uitvoeringsmodel gebruiken: alle niet-geheugeninstructies hebben een uitvoeringstijd van 1 klokcyclus en geheugenoperaties hebben een latentie van 153 klokcycli.

Het kan handig zijn om de hoofdstukken uit de cursus betreffende assemblercode en optimalisaties door te nemen vooraleer het practicum op te lossen.

## 2.4 Opgaven

1. Genereer eerst het bestand `jacobsthal.s` met behulp van de makefile (`make assembly`). *Let op dat je in de verdere stappen dit bestand niet opnieuw genereert, of je werk wordt overschreven*! Compileer vervolgens het uitvoerbaar bestand `jacobsthal` met behulp van de makefile (`make pract02`). Roep het programma aan met als eerste argument 9. Het tweede argument (het aantal iteraties) werd in alle scripts ingesteld op 1000. Om een tijdsmeting uit te voeren, kan je het bijhorende script `timing.sh` gebruiken door op de commandolijn `./timing.sh 9` uit te voeren.

   Hoeveel klokcycli heb je nodig om het `jacobsthal(9)` 1000 keer te berekenen? Deze waarde zal doorheen het practicum gebruikt worden als referentiewaarde. Let op, we vragen naar het aantal cycli dat de functie nodig heeft, dus niet de totale uitvoeringstijd. Hoe kan je de kost van de rest van het programma (argumenten inlezen, argumenten omzetten naar int,...) bepalen?

2. Tijdens practicum 1 zal je ongetwijfeld gemerkt hebben dat de gereserveerde stapelruimte in de functie `jacobsthal(int)` slechts beperkt gebruikt wordt. Pas de assemblercode aan zodat er geen onnodige stapelruimte wordt gereserveerd. Om fouten te vermijden ga je best stap voor stap te werk:

   - Compacteer de data op de stapel door deze te verplaatsen. Gebruik jouw oplossing van practicum 1 om het werk te vereenvoudigen. Verifieer steeds dat het programma nog werkt zoals voorheen.
   - Verminder nu de gereserveerde stapelruimte. Hoeveel bytes stapelruimte heb je minimum nodig per call naar de functie `jacobsthal`?

   **Tips**:

   - Om deze vraag op te lossen, hoef je enkel offsets en constanten aan te passen binnen de functie `jacobsthal`. Je hoeft dus geen instructies te vervangen of weg te laten!

- Hou steeds de uitvoer van jouw programma in de gaten: het is goed mogelijk dat jouw code correct assembleert en bijgevolg wel uitvoert, maar dat de berekeningen niet meer correct zijn. Ook hiervoor kan je gebruik maken van `timing.sh`. Controleer ook of de berekening nog correct is voor andere invoerwaarden (bijvoorbeeld 0 en 1, hiervoor kan je `timing.sh` ook best aanpassen).
- Wanneer je in de problemen komt (verkeerde berekening, programmacrash, ...) kan je je kennis van het vorige practicum gebruiken om jouw code te debuggen.

Wat is de uitvoeringstijd nu (gebruik steeds hetzelfde argument als voor vraag 1)? Bespreek deze optimalisatie.

Sla jouw code op als `practicum2_vraag2.s`

3. Code die zeer vaak wordt uitgevoerd noemen we hete code. Identificeer de heetste code van de functie `jacobsthal`. Met welk uitvoeringspad (argumentwaarden) komt dit overeen? Waarom is dit de heetste code?

   Tracht dit uitvoeringspad zo sterk mogelijk te optimaliseren. Aangezien dit de heetste code is van de functie zal de impact groot zijn, spendeer voldoende aandacht aan deze opdracht (tip: enkele instructies volstaan om dit uitvoeringspad te implementeren).

   Wat is de uitvoeringstijd nu?

   Sla je code op als `practicum2_vraag3.s`

4. Er zijn nog heel wat optimalisaties mogelijk. Optimaliseer de functie `jacobsthal` verder en probeer de uitvoeringstijd naar beneden te krijgen. De functie dient hierbij wel recursief gehouden te worden (beide calls naar `jacobsthal` moeten blijven staan).

   Doe de optimalisaties stap per stap en kijk of ze een positief effect hebben. Indien het effect negatief is, herstel de wijzigingen dan. Blijf er goed op letten dat het resultaat nog steeds correct is (voor alle mogelijke argumenten).

   Beschrijf voor de 3 optimalisatie die de grootste impact hebben gehad op de uitvoeringstijd:

   (a) De resulterende uitvoeringstijd (relatief ten opzichte van de originele uitvoeringstijd).

   (b) Waarom heb je deze optimalisatie toegepast?

   (c) Een bondige beschrijving van de optimalisatie: wat heb je precies gedaan? Let op: geen instructie-per-instructie log van de aanpassingen maar een korte beschrijving zoals "*instructie weggelaten wegens overbodig*" of "*instructie vervangen door sneller alternatief*".

   Sla bij deze vraag telkens je code op als `practicum2_vraag4_opt_`$Y$`.s`, waarbij $Y = 1, 2$ en 3 het optimalisatienummer is. Let erop dat jouw code compileert, de juiste code bevat (enkel de body van de functie `jacobsthal`) en correct werkt.

   Sla daarnaast je finale versie nog op als `practicum2_vraag4_opt.s`

   Beschrijf bondig welke stappen je hebt ondernomen om tot deze laatste versie te komen (dit zullen er dus allicht meer zijn dan de drie optimalisaties uit vraag 4), waarom je die stappen hebt gedaan en wat het effect was op de uitvoeringstijd (ook de pogingen die een negatief resultaat opleveren). De bedoeling is dat je ons duidelijk maakt hoe je te werk bent gegaan en welke redeneringen jullie gemaakt hebben.

   Deze opgave kan tijdrovend zijn. Spendeer voldoende aandacht aan deze opgave tijdens het practicum zodat je goed op weg bent om later verder te werken.

5. Teken de controleverloopgraaf van jouw uiteindelijk functie `jacobsthal`. Besteed voldoende aandacht aan de gehanteerde conventies! (volle lijnen, stippellijnen)

6. Jullie hebben ondertussen twee practica besteed aan de analyse en het optimaliseren van deze machinecode. Bespreek de keuze van het algoritme. Voor welke implementatie zouden jullie kiezen? Geef de code weer aan de hand van pseudo-code, C-code, Java-code, ...

De volgende opgaves zijn niet noodzakelijk om 20/20 te kunnen behalen, maar kunnen ingediend worden om extra punten te verdienen (je kan echter nooit meer dan 20/20 verdienen).

7. Je hebt nu manueel het werk uitgevoerd van een optimaliserende compiler. Gebruik nu de compiler om een geoptimaliseerd programma te genereren door het commando `make pract02_opt` uit te voeren. Bekijk in de makefile ook de argumenten die hiervoor aan de compiler worden meegegeven. Voer het resulterende programma (`jacobsthal_opt`) uit. Kijk hierbij naar hoe `timing.sh` de uitvoeringstijd bepaalt van jouw code om zelf de uitvoeringstijd van `jacobsthal_opt` te meten.

   Wat is de uitvoeringstijd nu? Verklaar gebruik makende van de assemblercode in `jacobsthal_opt.s`?

8. Implementeer jullie algoritme van vraag 6 in assembler en vervang onze implementatie van `jacobsthal` door jullie implementatie. Wat is nu de uitvoeringstijd? Bespreek.

   Sla jouw code op als `practicum2_vraag8.s`

   Tip: maak geen gebruik van vlottende-komma implementaties of benaderingsformules. Dien geen compiler-gegenereerde code in. Als we een vermoeden hebben dat die het geval is, zullen we jullie ondervragen over de details van de code als we de verbeteringen teruggeven.

9. Ga na welke implementatie (onze vs. jullie) efficiënter is. Maak hiervoor een duidelijke grafiek van de uitvoeringstijd van onze implementatie (niet-geoptimaliseerd en geoptimaliseerd) en en jullie implementatie. Laat het argument variëren van 0 tot 22. Let op de asbenoemingen, schaalkeuze, ...

   Bespreek — gebruik makend van deze grafiek(en) — jullie conclusies i.v.m. dit practicum.

Succes!