Rolf Vidar Mazunki Hoksaas
Institutt for Informatikk
Universitetet i Oslo

October 12, 2020

OBLIG 3 — Assemblerprogrammering

Del 1

Hvem er det som har ansvar for registrene, og hvilke registre, når det kalles en funksjon under ARM funksjonskallkonvensjon?

`R0-R3` er midertidlige variabler som blir brukt som argumenter. Disse har funskjonen som blir kalt på ansvar for. Om man trenger flere variabler en dette trenger vi en stack, og gir dermed tilgang til flere registere til en funskjon.

□

Hva skjer med input argumentet til fib funksjonen etter at fib returnerer?

`R0` blir brukt som return-verdien, og blir dermed slettet fra minne etter at `fib` returnerer.

□

Hvilken endring måtte du gjøre fra Del 1 slik at programmet avsluttes riktig og hvorfor måtte dette gjøres?

Jeg måtte wrappe main-funskjonen i en stack for å kunne lagre den midertidlig. Etter jeg var ferdig med å printe verdiene kunne jeg tømme stacken.

□

Del 3

Undersøk den produserte Assembler koden og sammenlign med din egen kildekode fra Del 2, er det noen store forskjeller mellom det du lagde og det GCC produserer?

Den største forskjellen mellom koden jeg har skrevet og den maskinen har skrevet er antall ekstra kode som egentlig bare er instillinger. I tilleg bruker den flere labels og hopper en del mer mellom forskjellige punkter i koden.

□

Endre -Os til -O2, hvordan påvirker dette den produserte koden? Prøv å endre til -O3, ser du noen forandring nå?

Det virker som om `O2` lager en mer effektiv kode med bekostning på plass, mens `O3` lager en kode som er vanskelig å forstå. Den ser mer effektiv ut, i og med at den allerede har kalkulert `233` allerede i kompilasjonen.

□

Hvilke argumenter er det for og i mot å bruke en kompilator sammenlignet med å skrive Assembler? Tenk spesielt på hvor mye jobb det ville være å oversette programsnuttene i Del 1, 2 og 3 til en annen prosessorarkitektur.

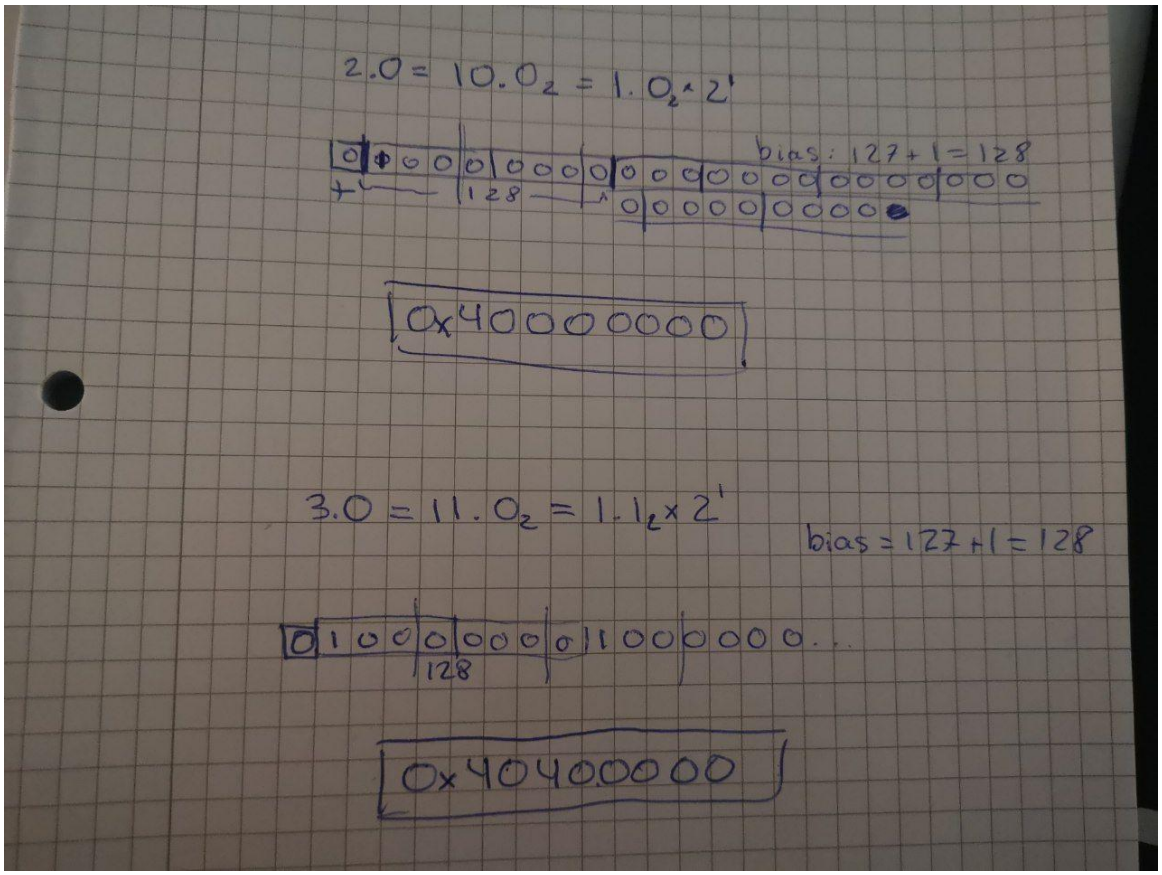
Å skrive Assembler-kode direkte er mer effektivt, men tar mye lenger tid å produsere. I tilleg er det mindre fornuftig for gjenbruk, i og med at det er ganske maskin-og-tids-avhengig kode. Det som fungerer nå vil sannsynligvis være unyttig kode om 10 år. I tillegg er kompilatorene ofte flinkere til å lage effektiv kode. Om et ledd i koden er ekstra treg kan man sjekke Assembler-koden som blir produsert av kompilatoren, og justere til målet vårt istedenfor.

□

Del 4

Oversett 2.0 til IEEE 754 32-biter flyttall.

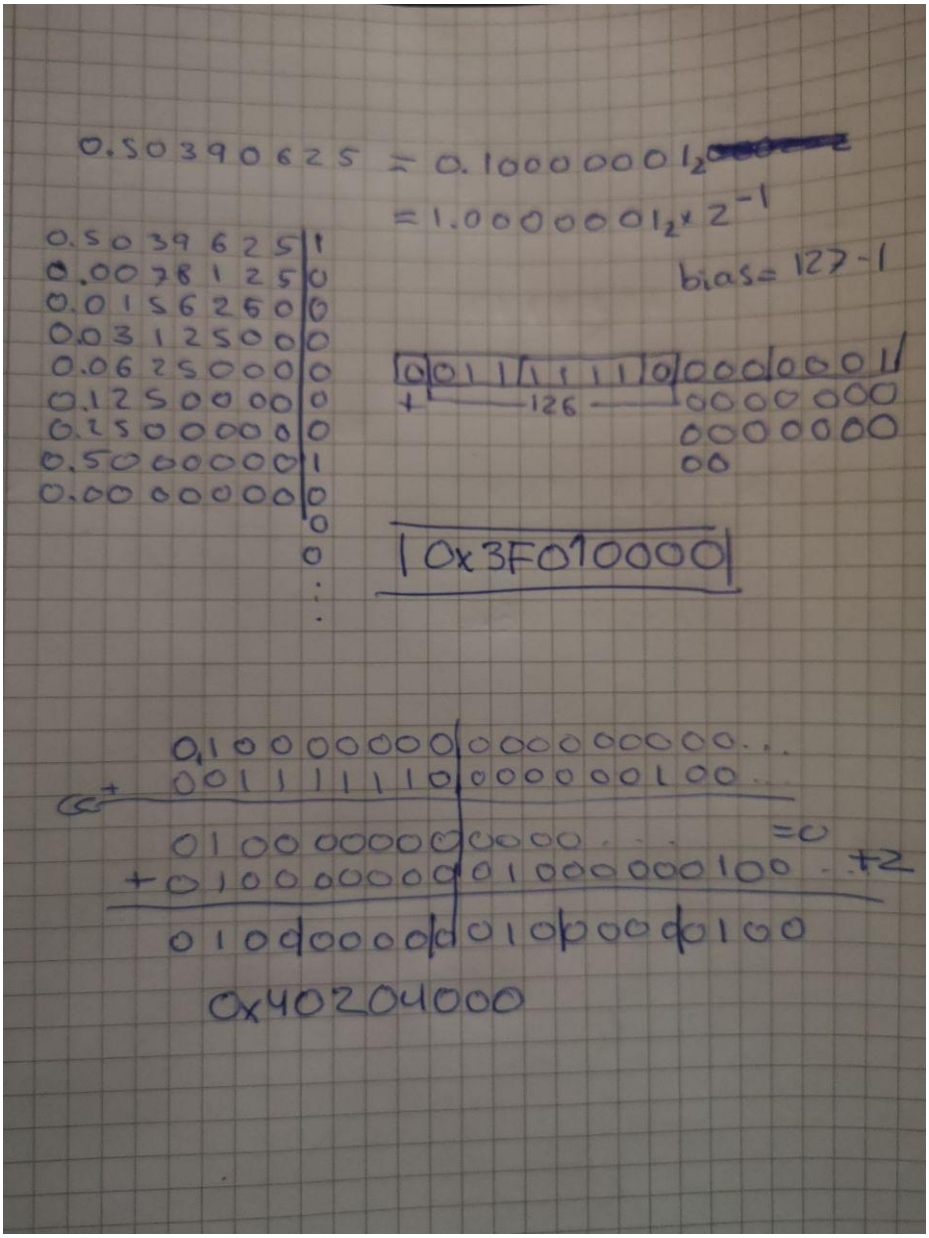
Oversett 3.0 til IEEE 754 32-biter flyttall.



□

Oversett 0.50390625 til IEEE 754 32-biter flyttall.

Legg sammen tallene 2.0 og 0.50390625 med samme fremgangsmåte som Figur 5.30 (du trenger ikke å ta hensyn til avrunding).



□