IN2060 Oblig3

Are Tov

October 2021

Del 2

1.

Det er den påkallte tar vare på verdiene i registre r4-r11, lr, og stacken. Den kallende tar vare på verdiene i registre r0-r3.

2.

Input argumentet til fib funsjonen går først ned til 0, så putter vi returverdien inn i r0 siden vi er ferdige også returnerer vi den.

3.

i første funksjonen gjorde vi alt i main, mens i denne lagrer vi registeret også input verdi, deretter går vi inn i en funksjon som gjøre utregningen fra del 1, og så går vi ut av den og printer ut returverdien.

Del 3

1.

Ser man bortifra alle flaggene som settes fungerer, og ser koden min og den kompilerte c koden veldig likt.

2.

Forskjell med min kode og O2: er ikke mye, men sammenlikningen som blir gjort i fib er annerledes, og stedet koden regner det ut er annerledes. Det virker som om koden har blitt flyttet rundt for å potensielt unngå mye branching.
Forskjellen mellom min kode of O3: Hovedgreia her er også at koden har blitt delt opp og flyttet rundt som de gjør av en eller annen grunn.
Begge de kompilerte versjonene gjør mye av det samme i det at de flytter ting rundt og gjør utregningene forskjellige steder fra min assembly kode.

3.

For kompilator: Fungerer på forskjellig maskiner og krever da ikke at man skriver om programmet for en ny maskin.
For assembler: Man må ha det. På et eller annet tidspunkt må man ha skrevet assembly. Kompilatoren er skrevet i assembler. Datamskinen forstår ikke c kode eller andre språk, en cpu forstår ikke assembly heller men assembly er en shorthand for å få kontakt med CPUen.
Mot kompilator: Det blir mange linjer i assembler koden.
Mot Assembler: Det suger. Det er kjedelig og knotete gi meg python any day fam.

Del 4.

1.

$2.0_{10} = 10.0_2 = 1.0_2 * 2^1$ Som medfører at biased exponent regnes ut via å konvertere $127 + 1 = 128 = 10000000_2$. Fraction blir alt bak decimal tegnet i det binære tallet, så vi får da tallet

$$\underset{signbit}{[0]} \quad \underset{BiasedExponent}{[10000000]} \; \underset{Fraction}{[1.00000000000000000000000]}$$

2.

$3.0_{10} = 11.0_2 = 1.1_2 * 2^1$ Som medfører at biased exponent regnes ut via å konvertere $127 + 1 = 128 = 10000000_2$ Fraction blir alt bak decimal tegnet i det binære tallet, vi får da tallet

$$\underset{signbit}{[0]} \quad \underset{BiasedExponent}{[10000000]} \; \underset{Fraction}{1.00000000000000000000000]}$$

3.

$0.50390625_{10} = 0.10000001_2 = 1.0000001_2 * 2^{-1}$ Som medfører at biased exponent regnes ut via å konvertere $127 - 1 = 126 = 01111110_2$ Fraction blir alt bak decimal tegnet i det binære tallet, vi får da tallet

$$\underset{signbit}{[0]} \quad \underset{BiasedExponent}{[01111110]} \; \underset{Fraction}{1.00000010000000000000000]}$$

4.

[0][10000000][00000000000000000000000]
[0][01111110][00000010000000000000000]
Shift by adding 10 and get
[0][10000000][00000000000000000000000]
[0][10000000][01000000100000000000000]
add together and get
[0][10000000][01000000100000000000000]