Processoren 2014

Week 6: De stack en eerste assemblycode

Uiterste inleverdatum: 5 januari 2015

Lever de volgende opgaven in via email naar je studentassistent. Zorg dat [Proc]Week6 in het subject van je email staat zodat we die niet over het hoofd kunnen zien.

1) In deze opgave gaat het erom getallen te vergelijken en op basis daarvan een beslissing te nemen. Schrijf programmafragmentjes (in practicum-assembly) voor de volgende operaties. De vergelijkingen zijn signed bedoeld, tenzij expliciet aangegeven. De fragmentjes zijn bedoeld voor de practicum processor.

   Ofschoon we nog niet expliciet assembler op het college hebben behandeld (Dat is voor 5 januari, maar de slides daarvan zal ik deze week al op woensdag uploaden. Ook de practicum-assembler, beschikbaar als Unix tarball, hoop ik woensdag af te testen (inclusief Windows port)), zou je aan de hand van de beschrijving van de practicum processor toch al enig idee moeten hebben hoe zijn instructies met mnemonics (Makkelijk te begrijpen afkortingen in Assembler) op te schrijven zijn. Bovendien zullen we op het college van vandaag al de nodige assembler gebruikt hebbem.

   1. Als R12 ≤ -25, trek 1 van R4 af.
   2. Als R4 < 0, zet R11 op 42.
   3. Als R6 $= 7$, verhoog R9 met 2.
   4. Als R0 ≥ 4, kopiëer het naar R8.
   5. Als $43 \leq$ R7 ≤ 87 unsigned, zet R5 op $2^{24} + 2^{20}$.
   6. Als R4 oneven is, spring 24 achteruit.

   Note: Als je denkt dat het beter kan dan gesuggereerd, schrijf het dan op...

2) De practicum-processor kent geen instructies voor PUSH, POP, CALL en RET. Geef programmafragmenten die hetzelfde doen als deze operaties vans geavanceerdere processoren. Gebruik R14 als stack pointer.

3) Zij gegeven een 32 bits processor met stackoperaties PUSH, POP en flowinstructies CALL en RET zoals gebruikt in het hoorcollege. Functies geïmplementeerd op deze processor mogen recursief zijn. We roepen een functie aan met 3 argumenten die last argument first gepushed worden, waarbij de stack van hoog naar laag geheugen groeit. Deze functie heeft ook 2 locale variabelen. Neem aan dat de processor een vast register voor stack en frame pointer heeft. Je mag ze dus met SP en FP aanduiden.

   a) Teken een plaatje van de stack op het moment van binnenkomst van de functie en hoe SP en FP naar de stack wijzen.

b) Teken een plaatje van de stack op het moment dat de processor de code van de functie begint uit te voeren en hoe SP en FP dan naar de stack wijzen.

c) Aangezien deze processor niet de mogelijkheid heeft om de argumenten van de stack weg te halen bij een RET, hoe worden de argumenten dan van de stack weggehaald?

Commentaar van een compilerbouwer (Marc S): Het feit dat het laatste argument van de call het eerste op de stack wordt gepushed is nogal gebruikelijk. Dit zorgt ervoor dat de argumenten weer in volgorde van stijgend adres op de stack te vinden zijn. Het maakt het bijv. ook mogelijk om in functies met een variabel aantal argumenten (Jullie moeten in imperatief programmeren al de fprintf en de sprintf call van C ontmoet hebben) aan de hand van het eerste of tweede argument (die dan een vaste offset ten opzichte van de frame pointer in de aangeroepen functie hebben) te kunnen bepalen welke offset de andere argumenten tov. de frame pointer hebben. Op veel processoren waaronder de Intel x86 is dit de standaard calling conventie.