64-041 Übung Rechnerstrukturen WS 15/16

Aufgabenblatt 10 Ausgabe: 16.12., Abgabe: 06.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 10.1 (Punkte 10+10+10+10)

Collatz-Vermutung: Wir betrachten einen Spezialrechner für die Collatz-Vermutung (Prof. Collatz war lange Jahre Direktor des Instituts für Angewandte Mathematik an der Uni Hamburg). Für Details siehe zum Beispiel: https://de.wikipedia.org/wiki/Collatz-Problem.

Man startet mit einem positiven ganzzahligem Eingabewert $x_0 > 0$ und überprüft, ob der Wert gleich Eins ist. In diesem Fall endet die Berechnung. Ansonsten wird der nächste Wert x_{i+1} gemäß einer einfachen Iteration berechnet, nämlich $x_{i+1} = 3 \cdot x_i + 1$ wenn x_i ungerade ist, und $x_{i+1} = x_i/2$ wenn x_i gerade ist. Die Iteration wird beendet, sobald der Wert $x_i = 1$ erreicht wird. Die bis heute nicht endgültig bewiesene Vermutung besagt, dass der Wert 1 (bzw. die Folge 4-2-1) für jeden beliebigen Eingabewert x_0 erreicht wird.

Ein Beispiel für eine Folge ist $11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$.

(a) Entwerfen Sie eine Hardwarestruktur für die Berechnung der Collatz-Iteration für n-bit Zahlen. Man benötigt offenbar ein n-bit Register X zur Speicherung des aktuellen Werts x_i und einen Eingang A für den Startwert x_0.

Überlegen Sie sich ein möglichst effizientes Rechenwerk (ALU) für die Berechnung des Nachfolgewerts x_{i+1}. Außer den arithmetischen Operationen sind dabei natürlich auch alle Shift- und logischen Operationen zugelassen. Die ALU soll abhängig vom Eingabewert selbständig die jeweils notwendige Berechnung (also entweder $3 \cdot x + 1$ oder $x/2$) ausführen.

Außer dem eigentlichen Rechenergebnis sollen auch die beiden Statussignale *is_one* und *is_odd* ausgegeben werden. Wie berechnen Sie diese?

Zeichnen Sie Ihre Hardwarestruktur mit der vollständigen ALU und allen ggf. notwendigen Multiplexern.

(b) Entwerfen Sie das Steuerwerk mit dem Startzustand *Start* (zum einmaligen Einlesen des Eingabewerts x_0) und dem Endzustand *One*, sowie allen weiteren Zuständen, die Sie benötigen. Der Automat bekommt das externe Steuersignal *start* sowie die beiden Statussignale *is_one* und *is_odd* als Eingaben. Zusätzlich zu den für Ihre Hardware benötigten

Steuerleitungen soll der Automat einen Ausgabewert *ready* liefern, um das Ende der Berechnung anzuzeigen. Zeichnen Sie das Zustandsdiagramm und geben Sie die logischen Gleichungen für das δ- und λ-Schaltnetz an.

(c) Schreiben Sie ein Java oder C Programm für die Collatz-Vermutung und erläutern Sie Ihren Algorithmus. Der Startwert x_0 wird als Kommandozeilenargument übergeben. Zählen Sie zusätzlich die Iterationsschritte i mit und geben Sie in jedem Schritt sowohl i und das Zwischenergebnis x_i aus.

(d) Für welchen Wert von $1 < x < 100$ ergibt sich die längste Folge? Nehmen Sie auch Ihr Geburtsdatum (in der Form Jahr Monat Tag, also zum Beispiel 19951131) als Eingabe des Programms und geben Sie die Folge der Zwischenergebnisse an.

Aufgabe 10.2 (Punkte 4·8 + 8)

Befehlsformate: Vergleichen Sie 0-, 1-, 2- und 3-Adress Maschinen, indem Sie für jede Architektur ein Programm zur Berechnung des folgenden Ausdrucks schreiben:

$$R = (A * B - C)/(E * F + D)$$

Für die unterschiedlichen Maschinentypen sind die jeweils verfügbaren Befehle unten angegeben. Bezeichner M und N stehen für 16-bit Speicheradressen und MEM[M] ist der Inhalt des Speichers an der Adresse M. Mit X, Y und Z werden 4-bit Registernummern codiert.

0-Adress Maschine mit einen unbegrenzten Stack (TOS "top of stack")

Mnemonik	Bedeutung
`PUSH M`	push; TOS = MEM[M]
`POP M`	MEM[M] = TOS; pop
`ADD`	tmp = TOS; pop; TOS = tmp + TOS
`SUB`	tmp = TOS; pop; TOS = tmp − TOS
`MUL`	tmp = TOS; pop; TOS = tmp ∗ TOS
`DIV`	tmp = TOS; pop; TOS = tmp / TOS

1-Adress Maschine: Akkumulatormaschine mit genau einem Register

Mnemonik	Bedeutung
`LOAD M`	Akku = MEM[M]
`STORE M`	MEM[M] = Akku
`ADD M`	Akku = Akku + MEM[M]
`SUB M`	Akku = Akku − MEM[M]
`MUL M`	Akku = Akku ∗ MEM[M]
`DIV M`	Akku = Akku / MEM[M]

2-Adress Maschine: benutzt nur Speicheroperanden

Mnemonik	Bedeutung
`MOV M,N`	MEM[M] = MEM[N]
`ADD M,N`	MEM[M] = MEM[M] + MEM[N]
`SUB M,N`	MEM[M] = MEM[M] − MEM[N]
`MUL M,N`	MEM[M] = MEM[M] ∗ MEM[N]
`DIV M,N`	MEM[M] = MEM[M] / MEM[N]

3-Adress Register-Maschine: *load-store* RISC-Architektur, 16 Universalregister

Mnemonik	Bedeutung
`LOAD X,M`	X = MEM[M]
`STORE M,X`	MEM[M] = X
`MOV X,Y`	X = Y
`ADD X,Y,Z`	X = Y + Z
`SUB X,Y,Z`	X = Y − Z
`MUL X,Y,Z`	X = Y ∗ Z
`DIV X,Y,Z`	X = Y / Z

(a) Schreiben Sie vier (möglichst kurze) Programme für die Berechnung des Ausdrucks $R = (A * B - C)/(D + E * F)$ auf den verschiedenen Maschinen. Dabei bedeuten $A \ldots F$ und R die Speicheradressen der Operanden bzw. des Resultats. Verwenden Sie, falls nötig, die Speicheradressen von $G \ldots Q$ für Zwischenergebnisse.

(b) Wenn die Befehlscodierung jeweils 8-bit für den Opcode verwendet (und natürlich 16-bit für eine Speicheradresse bzw. 4-bit für eine Registernummer), wie viele Bits werden dann für jedes der obigen vier Programme benötigt?

Welche Maschine hat also die kompakteste Codierung (gemessen an der Programmgröße in Bits) für dieses Programm?

Aufgabe 10.3 (Punkte 10+10)

Installation und Test der GNU-Toolchain: Ziel dieser Aufgabe ist es, dass Sie selbst Zugang zu einem C-Compiler und den zugehörigen Tools haben. Wir empfehlen die *GNU Toolchain* mit dem `gcc` C-Compiler und Werkzeugen. Diese ist auf den meisten Linux-Systemen bereits vorinstalliert, so dass Sie die Befehle direkt ausführen können.

Für Windows-Systeme könnten Sie die sogenannte Cygwin-Umgebung von `cygwin.com` herunterladen und installieren. Im Setup von Cygwin dann bitte den `gcc`-Compiler und die Entwickler-Tools auswählen und mit installieren. Alternativ können Sie auch einen anderen C-Compiler verwenden, Sie müssen sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.

Als dritte Alternative können Sie auf die Rechner in den PC-Poolräumen unseres RZ zurückgreifen, die PCs sind als Dual-Boot Systeme auch mit einer Linux Distribution (Ubuntu 14.04) ausgestattet.

Als vierte Alternative sei auf die von Andreas Mäder angebotene virtuelle Maschine hingewiesen.

Für einen ersten Test tippen Sie bitte das folgenden Programm ab oder laden Sie sich die Datei `aufg10_3.c` von der Webseite herunter. Ändern Sie dann die Datei, indem Sie ihre eigene Matrikelnummer eintragen. Übersetzen Sie das Programm und schauen Sie sich den erzeugten Assembler- und Objektcode an.

```
/* aufg10_3.c
 * Einfaches Programm zum Test des gcc-Compilers und der zugehörigen Tools.
 * Bitte setzen Sie in das Programm ihre Matrikelnummer ein und probieren
 * Sie alle der folgenden Operationen aus:
 *
 * Funktion         Befehl                                  erzeugt
 * ----------------+---------------------------------------+---------------
 * C -> Assembler:  gcc -O2 -S aufg10_3.c                  -> aufg10_3.s
 * C -> Objektcode: gcc -O2 -c aufg10_3.c                  -> aufg10_3.o
 * C -> Programm:   gcc -O2 -o aufg10_3.exe aufg10_3.c     -> aufg10_3.exe
 * Disassembler:    objdump -d aufg10_3.o
 * Ausführen:       aufg10_3.exe
 *
 * 32bit Code auf 64bit System: gcc -m32 ...
 */

#include <stdio.h>

int main( int argc, char** argv )
{ int matrikelNr = 123456;

  printf( "Meine Matrikelnummer lautet %d (0x%x)\n",
          matrikelNr, matrikelNr );
  return 0;
}
```

(a) Machen Sie sich mit dem Compiler und den Tools vertraut. Probieren Sie die vorgeschlagenen Befehle aus und sehen Sie sich die Ausgaben an.

Hinweis: Auf x86-64 Systemen (64bit Linux) sollten Sie die `gcc`-Compileroption `-m32` verwenden, um 32bit Code zu erzeugen.

(b) Schicken Sie den Quellcode sowie den erzeugten Assemblercode und die Ausgabe des Befehls `objdump -d` (GNU Toolchain) an Ihren Gruppenleiter.
Wenn Sie sowohl 64bit Code als auch 32bit Code einschicken und kurz auf die Unterschiede eingehen, gibt es als Weihnachtsgeschenk 10 Punkte extra!

Bei Verwendung anderer Compiler und Tools bitte ebenfalls die entsprechenden Ausgabedateien generieren und einschicken.

Hinweis: Der erzeugte Programmcode (`aufg10_3.exe`) soll nicht mit abgegeben werden. Verschiedene Mailserver halten Mails mit angehängten ausführbaren Programmen wegen eventuell enthaltener Viren automatisch zurück.