Rechnerorganisation
SS 2005
Übungsblatt Nr. 4

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin

Ausgabe am 3.06.2005 — Abgabe spätestens 17.06.2005, 12:00 Uhr

Bitte bei der Abgabe alle Namen/Matr.Nr. der Gruppe, NUMMER DER ÜBUNG/TEILAUFGABE und DATUM auf den Lösungsblättern **nicht vergessen!** Darauf achten, dass die Lösungen beim richtigen Tutor/der richtigen Tutorin abgegeben werden.
Zu spät abgegebene Lösungen werden nicht mehr berücksichtigt!

---

1. Aufgabe: Doppelt verkettete Liste (20 Punkte)

Schreiben Sie IA -32-Assembler-Routinen zur Manipulation doppelt verketteter Listen. Listenelemente seien 4-byte-Integers. Jedes Element hat einen Zeiger auf seinen Vorgänger und einen auf seinen Nachfolger. Das erste Element zeigt auf sich selbst als Vorgänger, das letzte auf sich selbst als Nachfolger. In C würden Sie diesen Datentyp

`dolili` (double linked list) wie folgt deklarieren:

```
struct dolili {
  int data;
  struct dolili *next_ptr;
  struct dolili *prev_ptr;
};
struct dolili *elem_ptr = NULL; // Erzeugen eines Zeigers auf ein Element der Liste
```

Damit Sie dynamisch neue Elemente einer Liste hinzufügen können, müssen Sie einen Teil des Speichers verwenden, den Sie selbst „verwalten“. Das Löschen von Elementen können Sie durch das logische Ausfügen der Elemente aus der Liste realisieren. Dadurch wird natürlich der Speicherplatz des jeweiligen Elements nicht freigegeben (es liegt ja noch im Speicher, ist nur nicht mehr durch Zeiger verbunden). Zur Verwaltung Ihres Speichers müssen Sie also noch zusätzlich z.B. eine Tabelle mitführen, welche Ihnen belegte/freie Speicherplätze angibt. Es genügt für diese Aufgabe, wenn Ihr Speicher inklusive Verwaltung für 64 Elemente ausgelegt ist. Erstellen Sie folgende Routinen:

```
void create_list(char *data_ptr, int data);
//Erzeugen einer Liste im leeren Speicher mit data als erstem Element
void insert(char *data_ptr, int data);
//Einfügen eines Elements data an das Ende der Liste
void del(char *data_ptr, int nr);
//Löschen des Elements nr aus der Liste; ohne Effekt, falls nr größer als die An-
zahl
der Elemente in der Liste
int get_next(char *data_ptr, int nr);
//Liefert den Inhalt des Elements nach dem Element nr; -1 falls nr das letzte Ele-
ment
ist
```

```
int get_previous(char *data_ptr, int nr);
//Liefert den Inhalt des Elements vor dem Element nr; -1 falls nr das erste Element
ist
```

Zum Schaffen eines initialen Speicherbereichs kann einfach mit Hilfe von `malloc` Speicher reserviert werden (im Hauptprogramm). Dazu müssen vorher die Bibliotheken `#include <stdlib.h>` und `#include <malloc.h>` eingebunden werden:

```
char *data_ptr; // Zeiger auf den Speicherbereich
data_ptr = (char *)malloc(1024); // Reservierung eines Speicherbereichs von 1 kbyte
```

2. Aufgabe: Quicksort (8 Punkte)

Schreiben Sie eine IA-32-Assembler-Routine zum Sortieren eines Vektors beliebiger Länge len mit Integer-Zahlen als Elementen. Als Algorithmus soll hierfür Quicksort rekursiv eingesetzt werden (siehe z.B. http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/quick/quick.htm). Vektoren werden in gewohnter Weise definiert, die Funktion soll wie folgt aufgerufen werden und den Vektor aufsteigend sortieren.

```
void quicksort(int *vec, int len)
```

3. Aufgabe: Pipelining (2 Punkte)

Gehen Sie im Folgenden von einer einfachen 5-stufigen Pipeline aus: Befehl holen (IF), Befehl dekodieren (ID), Operanden holen (OF), Ausführung (EX), Rückspeichern (WB). Weiterhin liegt eine reine Load/Store-Architektur ohne architekturelle Beschleunigungsmaßnahmen (z.B. forwarding, reordering etc.) oder Hardware zur Erkennung von Hemmnissen vor. Operanden können erst dann aus Registern geholt werden, nachdem sie zurück gespeichert wurden. ADD X, Y, Z steht für Z:= X+Y. Welche Aussagen lassen sich dann treffen?

A: Die Befehlsfolge ADD R2, R3, R1; NOP; ADD R1, R5, R4 ist ausführbar.
B: Die Befehlsfolge ADD R2, R3, R1; ADD R1, R5, R4 ist nicht ausführbar.
C: Die vollständige Abarbeitung von Z:=X+Y; A:=Z+B dauert insgesamt 10 Takte.
D: Die vollständige Abarbeitung von Z:=X+Y; A:=Z+B dauert insgesamt 8 Takte.
E: In dieser 5-stufigen Pipeline kann es keine Hemmnisse durch Datenabhängigkeiten geben.
F: Im Idealfall führt diese Pipeline zu einer 5-fachen Beschleunigung der Befehlsarbeitung eines Programms durch überlappende Bearbeitung.
G: Die vollständige Abarbeitung von Z:=X+Y; A:=Z+B dauert insgesamt 6 Takte.
H: Der Compiler muss mögliche Pipeline-Konflikte auflösen.