Hochschule

Ravensburg-Weingarten

Prof. Dr.-Ing. F. Brümmer

Leistungsnachweis

im Fach

Rechnertechnologie

05.07.2006
(SS 2006)

Name: Vorname: Matr. Nr.

Studienrichtung:

Wichtige Hinweise:

- Dauer der Klausur: 90 Minuten
- Zugelassene Hilfsmittel: alle
- Bitte tragen Sie auf dem Deckblatt ihren Namen, Vornamen, Matrikelnummer und Ihre Studienrichtung ein
- Versehen Sie jedes Lösungsblatt mit Ihren Namen
- Die Aufgabenblätter und das Deckblatt sind mit der Klausur abzugeben
- Geben Sie die zur Erlangung Ihrer Ergebnisse notwendigen Zwischenschritte an. Nur so erzielen Sie die maximalen Punktezahlen.
- Die Unterlagen zur Klausur setzen sich zusammen aus:
  1 Deckblatt
  4 Seiten Aufgabenblätter

1) Rechner lassen sich nach unterschiedlichen Architekturprinzipien aufbauen, die für verschiedene Anwendungszwecke konzipiert wurden.
   1.1 Welche Auswirkungen sind für den Anwender sichtbar, wenn die zugrunde liegende Architektur der von-Neumann-Architektur entspricht?
   1.2 Welche Auswirkungen sind zu beobachten, wenn eine Havard-Architektur zum Einsatz kam?
   1.3 In welcher Prozessorgattung (Mikroprozessor, Mikrocontroller, Signalprozessor) ist die jeweilige Rechnerarchitektur überwiegend anzutreffen (Begründen Sie Ihre Ansicht!)

2) In Rechnersystemen stellt der Speicheraufbau ein wesentliches, die Leistungsfähigkeit des Rechnersystems charakterisierendes Merkmal dar.
   2.1 Erläutern Sie, welche Speichertypen (DRAM, SRAM, EPROM oder Varianten) an welcher Stelle im Rechner anzutreffen sind und warum gerade sie eingesetzt werden um die gewünschte Funktion zu realisieren!
   2.2 Welche Vor- und Nachteile sind zu erwarten, wenn ein Prozessor mit einer Vielzahl von Registern aufwartet?

3) Beim Entwurf eines Rechnersystems spielen neben der Wahl des Architekturprinzips auch Gestaltungsgrundsätze eine wesentliche Rolle.
   3.1 Welche Auswirkungen sind für den Anwender zu erwarten, wenn der eingesetzte Prozessor eine virtuelle Speicherverwaltung aufweist?
   3.2 Wie wirkt sich die dynamische Erweiterbarkeit, die einem Befehlssatz eines Prozessors zugrunde liegen möge, für den Anwender aus?

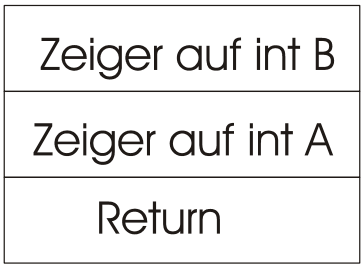
4) In einem Pentium lassen sich unterschiedliche Datenformate verwenden, die teilweise unmittelbar an der Definition von „C“ anknüpfen.
   4.1 Zählen Sie die Datenformate zur Darstellung von vorzeichenlosen Zahlen auf und erläutern Sie, wie diese Zahlen implementiert sind (Anzahl Bytes, Wertigkeit der einzelnen Bitstellen)!
   4.2 Zählen Sie die Datenformate zur Darstellung von vorzeichenbehafteten Zahlen auf und erläutern Sie, wie diese Zahlen implementiert sind (Anzahl Bytes, Wertigkeit der einzelnen Bitstellen)!
   4.3 Zählen Sie die Datenformate zur Darstellung von gebrochen rationalen Zahlen auf und erläutern Sie, wie diese Zahlen implementiert sind (Anzahl Bytes, Wertigkeit der einzelnen Bitstellen)!

5) Wenn Sie den Befehlssatz des Pentiums hinsichtlich der arithmetischen Befehle betrachten so fällt auf, dass für die Addition und Subtraktion nicht zwischen vorzeichenbehafteten und vorzeichenlosen Zahlen unterschieden, bei der Multiplikation und der Division jedoch eine Unterscheidung getroffen wird.
   5.1 Ist es dann noch sinnvoll bei der Definition von Zahlen in der Hochsprache „C“ eine Unterscheidung zwischen vorzeichenlose und vorzeichenbehaftete vorzunehmen, wenn diese Zahlen nur durch Additionen oder Subtraktionen verknüpft werden? (Begründen Sie Ihre Ansicht!)
   5.2 Ist bei der Multiplikation und der Division auch eine Mischung zwischen vorzeichenlosen und vorzeichenbehafteten Zahlen in einem Befehl möglich? (Begründen Sie Ihre Ansicht!)

6) Bei der Assemblerprogrammierung unterscheidet man zwischen verschiedenen Adressierungsarten.

6.1 Welche Unterschiede bestehen zwischen der direkten und der indirekten Adressierung?
6.2 Lassen sich in einem Befehl zwei direkte Adressen verwenden? (Begründen Sie Ihre Ansicht!)

7) Es soll ein Assemblerprogramm entwickelt werden, mit dem sich die Werte von zwei Parametern vertauschen lassen. Die Werte sind in der Hochsprache „C“ als Integer definiert und werden dem Assemblerprogramm als Adressen übergeben. Beim Aufruf des Assemblerprogramms ist der Stack wie folgt vorbereitet:



Jeder Eintrag auf dem Stack umfasst 4 Byte. „Zeiger auf int A“ und „Zeiger auf int B“ sind die beiden Adressen, deren Werte vertauscht werden sollen. Notieren Sie die Assemblerbefehle, sodass die gewünschte Funktion erfüllt wird!

8) Mit einem Assemblerprogramm soll ein Vektor (vec_a) negiert werden, was bedeutet, dass jedes Element des Vektors negiert wird. Die einzelnen Elemente sind Long-Integerzahlen (64 Bit), wobei die Anzahl (number) als weiterer Parameter übergeben wird. Die negierten Zahlen sollen in einem zweiten Vektor (vec_b) abgelegt werden, dessen Anfangsadresse gleichfalls als Parameter übergeben wird. In dem aufrufenden C-Programm ist die Funktion wie folgt deklariert:

```
extern  void PRE_CDECL NEGATE( long  *vec_a,  long  * vec_b,
int number) POST_CDECL;
```

8.1 Nach dem Aufruf des Programms stellt sich leider eine Fehlfunktion heraus.
8.2 Suchen Sie die Fehler und begründen Sie die Fehlfunktion! Korrigieren Sie das Programm, sodass sich die gewünschte Funktion einstellt!

```
 *      Unterprogramm NEGATE
1*      Funktion: Negation eines Vektors, der eine angebbare Anzahl von
2*      Elementen enthält. Die einzelnen Elemente sind Long-Integer. Das Ergebnis wird in
3*      einem weiteren Vektor angelegt. Das Programm gibt einen „void-Wert als Rückgabe-
4*      parameter zurück.
5*      Übergabeparameter: PRE_CDECL skalar( int  *vec_a,  int * vec_b,  int number )
6*              POST_CDECL; (entsprechend C -Parameter calling convention)
7*      Rückgabeparameter: void (entsprechend C -Parameter calling convention)
8
9       segment .text
10      global  NEGATE
11
12      %define vec_a                   [ebp+8]
```

```
13      %define vec_b           [ebp+12]
14      %define number          [ebp+16]
15
16 NEGATE:
17              enter   0,0             ;setup routine
18              pusha                   ;Register retten
19              mov ecx,number          ;Anzahl Elemente
20              mov esi,vec_a           ;EBX zeigt auf Vektor A
21              mov edi,vec_b           ;ESI zeigt auf Vektor B
22
23 Schleife:    mov eax,[esi]           ;Element laden, niederwertiger Part
24              mov edx,[esi+4]         ;Element laden, höherwertiger Part
25              not eax                 ;Komplement EAX
26              not edx                 ;Komplement EDX
27              add eax,1               ;Zweier-Komplement
28              adc edx,0               ;Carry-behandlung
29              mov [edi],eax           ;negiertes Element speichern, niederwertiger Part
30              mov [edi+8],edx         ;negiertes Element speichern, höherwertiger Part
31              add esi,8               ;nächste Element Vektor A
32              add edi,8               ;nächste Element Vektor B
33              push ecx                ;Schleifenzähler retten
34              loop Schleife           ;alle Elemente berechnen
35
36              pop ecx                 ;Schleifenzähler restaurieren
37              popa                    ;Register restaurieren
38              leave
39              ret                     ;Zurück zu aufrufendem Programm
```