

INSTITUT FÜR INFORMATIK
Lehrstuhl für Rechnerarchitektur (Informatik 3)
Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen

29.09.2004

Klausur

zu

"Organisation und Technologie von Rechensystemen 4"

.....
Matrikelnummer Geb.-Datum Vorname Name

- Es sind keine elektronischen Hilfsmittel erlaubt.
- Legen Sie den Ausweis (mit Lichtbild!) griffbereit auf den Platz.
- Dieses Aufgabenheft umfasst 12 Seiten. Überprüfen Sie die Vollständigkeit.
- Gesondert beigelegte Blätter werden nicht bewertet!
- Schreiben Sie deutlich! Unleserliches wird nicht bewertet!
- Es darf nicht mit der Farbe rot geschrieben werden!

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausurunterlagen
- die Kenntnisnahme der obigen Informationen.

Erlangen, den 29.09.2004
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis der Klausur unter Angabe der Matrikelnummer veröffentlicht wird.

Erlangen, den 29.09.2004
(Unterschrift)

Aufgabe	1	2	3	4	5	Summe
max. Punktzahl	5	10	17	18	10	60
erreichte Punktzahl						

Aufgabe 1: Allgemeines**(5 Punkte)**

1a) Welche Aussagen sind wahr? (2 Punkte; Punktabzug bei falschen Antworten!)

- CPU-Exceptions sind synchron zu dem laufenden Programm.
- Jede CPU sorgt dafür, dass die Register eines Programmes nicht durch einen plötzlich auftretenden Interrupt verändert werden.
- Ein Interrupt Handler darf den Befehl `cli` ausführen.
- Bei einem Interrupt wird das Flags-Register gespeichert.
- Die Interrupt Vektor Tabelle muss im Hauptspeicher abgelegt werden.
- Die System Call Vektor Tabelle muss initialisiert sein, bevor ein Programm einen System Call ausführen kann.
- Ein System Call Handler muss alle zu verändernden Register sichern und vor dem Ausführen von `iret` wieder restaurieren

1b) Warum ist es an gewissen Stellen im Betriebssystem sinnvoll, die Interrupts kurzfristig abzuschalten? (1 Punkt)

1c) Beschreiben Sie, wofür die folgenden Prozessor-Flags beispielsweise Verwendung finden (2 Punkte):

i) Carry-Flag

ii) Zero-Flag, Negative-Flag

iii) Interrupt-Flag

Aufgabe 2: Speicher**(10 Punkte)**

- 2a) In einer Struktur werden die Elemente $i=-1$, $c=64$ und $n=4711$ gespeichert. Ein Speicherauszug sieht folgendermaßen aus (alle Bytes hexadezimal):

```
40 64 ff ff 67 12 00 00
```

Wie sieht die Struktur aus? Welche Byte-Order hat der verwendete Rechner? (3 Punkte)

- 2b) Warum sollte ein Assembler-Programmierer die Byte-Order (Little-Endian bzw. Big-Endian) seines Rechners kennen, während die Bit-Order des Rechners im allgemeinen keine Rolle spielt? (2 Punkte)

- 2c) Schreiben Sie ein Unterprogramm `is_big_endian`, das testet, ob der verwendete Rechner ein Big-Endian-Rechner ist! Das Test-Ergebnis soll als Boolescher Wert an das aufrufende Programm zurückgegeben werden. (5 Punkte)

Aufgabe 3: Arithmetik**(17 Punkte)**

- 3a) Welche Zahl steht nach Ausführung der folgenden Befehle im Register %eax? Welche Bedeutung hat diese Zahl? (2 Punkte)

```
movl $0, %eax
notl %eax
shrl $1, %eax
```

- 3b) Compilieren Sie die folgende Hochsprachen-Funktion in Assembler-Code! Benutzen Sie keine Multiplikations- bzw. Divisions- oder Modulo-Befehle! Optimieren Sie Ihr Programm! (7 Punkte)

Hinweis 1: der ‘%’-Operator ist in C, C++ und Java der Modulo-Operator.

Hinweis 2: die optimale Lösung für Intel-Prozessoren benötigt 3 Maschinen-Instruktionen.

```
unsigned int
odd(unsigned int x)
{
    if (x % 2 == 0) {
        return 0;
    } else {
        return 1;
    }
}
```

- 3c) Konvertieren Sie folgende Hochsprachen-Funktion in semantisch äquivalenten Hochsprachen-Code, so dass sich dieser möglichst leicht in Assembler-Code für eine Ein-Address-Maschine umwandeln lässt! (8 Punkte)

Hinweis: Ein-Adress-Maschine bedeutet, dass bei Berechnungen nur Ausdrücke der Form

Akku = Akku *op* Operand

erlaubt sind (*op* ist Element von {+, -, *, /}, Operand kann eine Variable oder eine Konstante sein.

```
int test(int n)
{
    int i;

    for (i = 0; i < 2 * n; i++) {
        if (func(n + i * i)) {
            return i;
        }
    }

    return -1;
}
```

Aufgabe 4: Unterprogramme**(18 Punkte)**

Gegeben sei folgendes Hochsprachen-Unterprogramm:

```
unsigned int
isqrt(unsigned int x)
{
    unsigned int i;

    i = 0;
    while (i * i < x) {
        i++;
    }

    return i;
}
```

Ein Compiler habe daraus den folgenden (lückenhaften) Assembler-Code generiert:

```
.align 4
isqrt: .globl isqrt
    movl 4(      ),%ecx
    xorl %edx,%edx
    cmpl %ecx,%edx
    jae .L4
    .align 4
.L5:

    movl %edx,%eax
    imull %eax,%eax

    jb .L5
.L4:
    movl      ,%eax
    ret
```

- 4a) Vervollständigen Sie den Assembler-Code! (5 Punkte; Punktabzug bei sinnlosen Einfügungen)
- 4b) Was könnten die beiden Pseudo-Befehle `.align 4` bedeuten? Wofür sind sie sinnvoll? Warum stehen sie genau an diesen Stellen? (2 Punkte)

- 4c) Schätzen Sie ab, wieviel Byte Speicherplatz das Assembler-Code-Stück belegt! Begründen Sie Ihre Antwort! (5 Punkte)

- 4d) Während der Initialisierung eines PCs wird u.a. folgender BIOS-Code (ROM) durchlaufen:

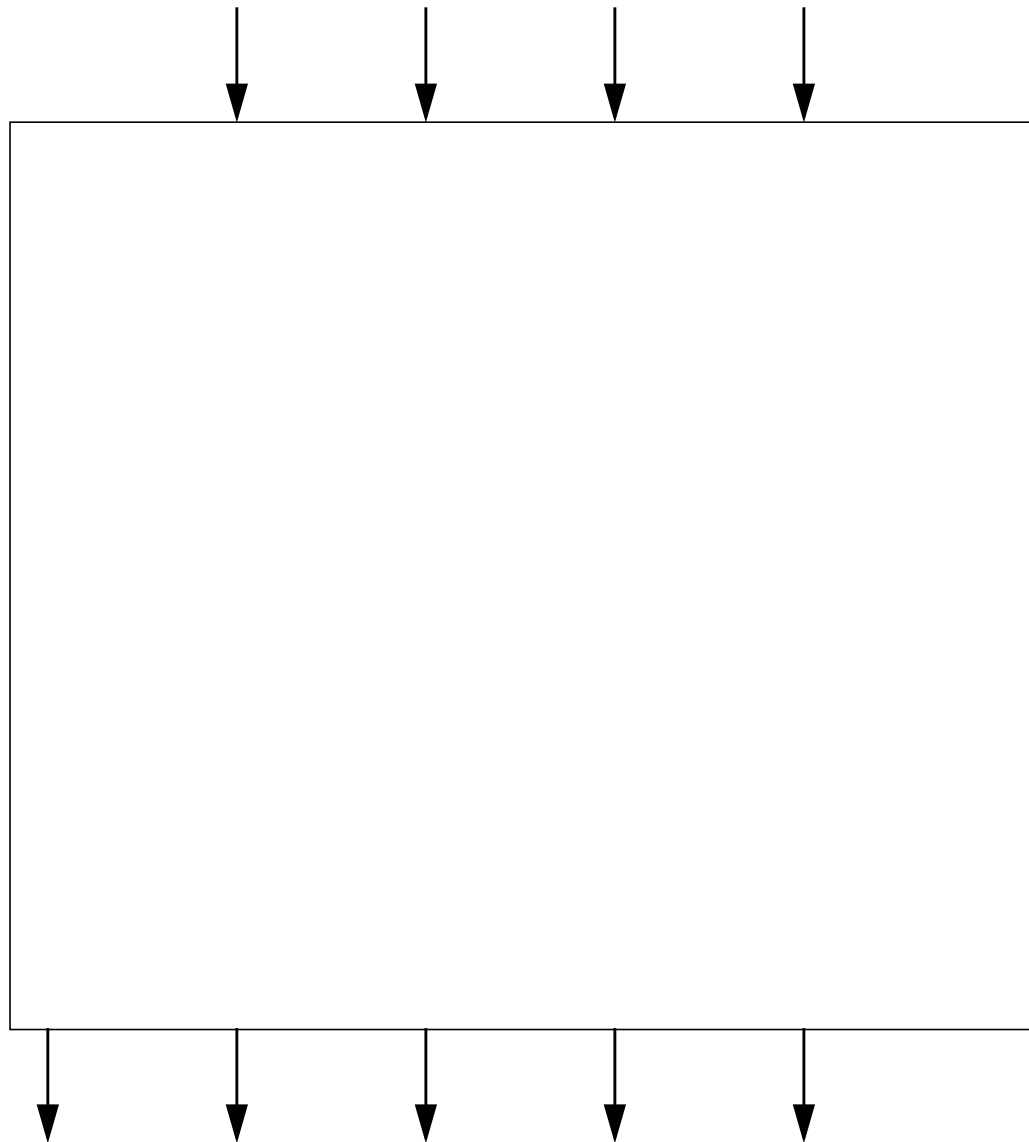
```
    ...  
    movw $L1, %sp  
    jmp L3  
L1:  .word L2  
L2:  ...  
  
L3:  ...  
    ret
```

Erklären Sie die Besonderheit dieses Code-Stückes! Was ist der Sinn dieser auf den ersten Blick umständlichen Programmierweise? (6 Punkte)

Aufgabe 5: Hardware**(10 Punkte)**

Es soll ein Inkrementierer in Hardware aufgebaut werden. Als Eingabe werden 4 Leitungen angenommen. Als Ausgabe sollen 4 Leitungen für die Signalisierung des normalen Ergebnisses dienen. Zusätzlich ist ein Übertrag über eine fünfte Leitung zu signalisieren.

Für die Implementierung des Schaltnetzes dürfen AND-, OR-, XOR- sowie NOT-Gatter verwendet werden.



Zusätzlicher Platz

Zusätzlicher Platz

Zusätzlicher Platz