Autor: Sebastian Sossalla

Disclaimer: Das sind Lösungsvorschläge, keine Musterlösungen.

Aufgabe 1: Allgemeines(9 Punkte)

1a)Welche Aussagen sind wahr? (4 Punkte; Punktabzug bei falschen Antworten!)

<x> Variablen können im Speicher oder in Registern liegen.
<x> Register können Variablen mit und ohne Vorzeichen aufnehmen.
<x> Es können Variablen mit und ohne Vorzeichen mit „add“ addiert werden.
< > Alle Rückkehradressen von Unterprogrammen werden in Registern gespeichert.
<x> Mit „add“ können beliebig große Variablen addiert werden.
<x> Auf einigen Architekturen brauchen „int“- und „float“-Werte genauso viel Platz im Speicher.
< > Mit „add“ können sowohl „int“- als auch „float“-Werte addiert werden.
<x> Die Fließkomma-Division ist meist schneller als die Integer-Division.

1b) An welchen Stellen im Betriebsssytem ist es sinnvoll, die Interrupts kurzfristig abzuschalten? (1 Punkt)

Lösung:
Innerhalb Zeitkritischer-/Nebenläufigkeits-kritischer Abschnitte. Zum Beispiel zur Ausführung eines Interrupt-handlers darf kein erneuter Interrupt auftreten.

1c) Beim Auftreten eines Interrupts, eines System-Calls oder einer Exception werden die Bits, die anzeigen, ob der Prozessor zum Auftrittszeitpunkt im Supervisor-Modus war, gesichert. Warum? (1 Punkt)

Lösung:

Es muss nach dem Auftreten eines Interruts/System-Calls/Exception wieder der vorher herrschende Zustand hergestellt werden.Sonst kann das privilege-Level so wie vor dem Auftrittszeitpunkt nachher nicht mehr hergestellt werden. Interrupt/System-Call/Exception könnte ja im Supervisor-Modus aufgetreten sein.

1d)Welche Aussagen sind wahr? (3 Punkte; Punktabzug bei falschen Antworten!)
< > Segmentierung und Paging ist das gleiche.
<x> Segmentierung und Paging kosten Performance.
<x> Segmente haben eine variable Größe.
< > Es gibt genauso viele Segmente wie Pages.
< > Mehrstufige Page-Tabellen beschleunigen den Hauptspeicherzugriff.
<x> Eine MMU kann den Zugriff auf bestimmte Speicherzellen hardware-mäßig verhindern.

Aufgabe 2: Register(10 Punkte)

Die 32-Bit-CPUs der Motorola-M68000-Serie stellen je 8 32-Bit-Datenregister (%d0-%d7) und 8 32-Bit-Adressregister (%a0-%a7) dem Programmierer zur Verfügung. Nur mit den Datenregistern kann mit den Rechenoperationen (add, sub, and, or, usw.) gerechnet werden. Für die Adressierungsarten „Register

indirekt“, „Register indirekt mit Displacement“ sowie „Register indirekt mit Pre-/Post-Increment/Decrement“ können nur Adressregister verwendet werden. Zusätzlich gibt es mov-Befehle, die Daten/Adressen zwischen den Registern hin- und herkopieren können.

2a) Die M68000-CPUs kennen keine Adressierungsart „Register indirekt mit Index und Scale-Faktor“. Wie könnte man diese Adressierungsart mit Hilfe der zur Verfügung stehenden nachbilden? Schreiben Sie den folgenden Befehl so um, dass er nur die verfügbaren Adressierungsarten „immediate Operand“, „direkte Adresse“, „Register“ bzw. „Register indirekt“ verwendet (Datenregister %d0 sowie Adressregister %a0 dürfen verändert werden).(4 Punkte)

movw (%a1, %d1, 2), %d2

Lösung:

```
movw %a0, %d0      /* base*/
addw, %dl, %d0     /* base + Index*/
addw %dl, %d0      /* base + 2 * Index*/
movw %d0, %a0
movw (%a0), %d2
```

2b)Welchen Vorteil hat die Aufteilung in Daten- und Adressregister? Welchen Nachteil besitzt sie?(6 Punkte)

b) Vorteile:

- Kompaktere Codierung der Befehle
- Paralleler Zugriff auf Adressen & Datenregister
- Optimierterer Zugriff da Daten- von Adress-Modul getrennt!
- Code ist leichter zu lesen

Nachteile:

- Umständlicher da man ständig hin und her kopieren muss
- Mehr Befehle notwendig
- 16 Register vorhanden aber nur 8 zur Berechnung verfügbar

<u>Aufgabe 3:</u> Speicher, Adressierungsarten(7 Punkte)

Folgendes Intel-386-Assemblerprogramm laufe im Rechner ab:

1. **movl $0x10000, %esp**
2. **movl $32, %eax**
3. **movl $16, %ebx**
4. **pushl %eax**
5. **movl %ebx, 4(%esp)**
6. **movb %eax, 36(%eax, %ebx, 2)**

Hinweise (Intel-386):Verwendet wird das Little-Endian-Format.Der Stack-Pointer zeigt auf das zuletzt abgelegte Datum.Der Stack wächst von oben nach unten.

3a) Welchen Wert haben die Register %eax, %ebx, %esp nach Ausführung des Programmstückes?(3 Punkte)

1. esp = 0x10000
2. eax = 32
3. ebx = 16
4. esp = 0x0FFC, (%esp) = 32 => (0x0FFC) = 32
5. (0x10000) = 16
6. 32 + 36 + (2 * 16) = 68 + 32 = 100, 100 = 64, also: (0x64) = eax = 0x20

Also:
%eax = 32
%ebx = 16
%esp = 0x10004

3b)Welche Bytes im Speicher wurden beschrieben? Nennen sie ihre Adressen und ihren Inhalt nach der Ausführung des Code-Stückes!(4 Punkte)

Lösung:

16 = 0x00 00 00 10
32 = 0x00 00 00 20
100 = 0x00 00 00 64

0x10000 = 0x10
0x10001 = 0x00
0x10002 = 0x00
0x10003= 0x00

0x0FFC = 0x20
0x0FFD = 0x00
0x0FFE = 0x00

0x0FFF = 0x00

0x0064 = 0x20

100 = 64 + 32 + 4 = 6 * 16 + 4

Aufgabe 4: Arithmetik, Kontrollstrukturen, Unterprogramme(

Gegeben sei folgendes Programm:

```
int getnum(FILE *fp){
        int sign;
        int num;
        int c;
        c = fgetc(fp);
        if (c == '-') { sign = -1;  c = fgetc(fp);  }
        else {
                sign = 1;
        }

        num = 0;
        while ('0' <= c && c <= '9') {
                num = num * 10 + c - '0';
                c = fgetc(fp);
        }

        ungetc(c, fp);
        return sign * num;
}
```

Hinweis: `FILE *fp` definiert eine Variable, die die Adresse einer FILE-Struktur enthält.

4a) Konvertieren Sie die Hochsprachen-Funktion in semantisch äquivalenten Hochsprachen-Code, so dass sich dieser möglichst leicht in Assembler-Code für eine Ein-Address-Maschine umwandeln lässt! (12 Punkte)

Hinweis: Ein-Adress-Maschine bedeutet, dass bei Berechnungen nur Ausdrücke der FormAkku = Akku *op* Operanderlaubt sind (*op* ist Element von {+, -, *, /}, Operand kann eine Variable oder eine Konstante sein.

Lösung:

```
_getnum(FILE *fp) {
        int sign;
        int num;
        int c;
        int akku;
        c = fgetc(fp);

        if (c != '-') then goto _else;
```

```
        sign = 1;
        c = fgetc(fp);

_else:  sign = 1;

        num = 0;

        goto _text;
_loop:  akku = num;
        akku = akku * 10;
        akku = akku + c;
        akku = akku - '0';
        num = akku
        c = fgetc(fp);

_test:  if (c > '0') goto _next;
        if (c <= '9') goto _loop;
_next:

        ungetc(c, fp);
        akku = sign;
        akku = akku * num;
        return akku;
}
```

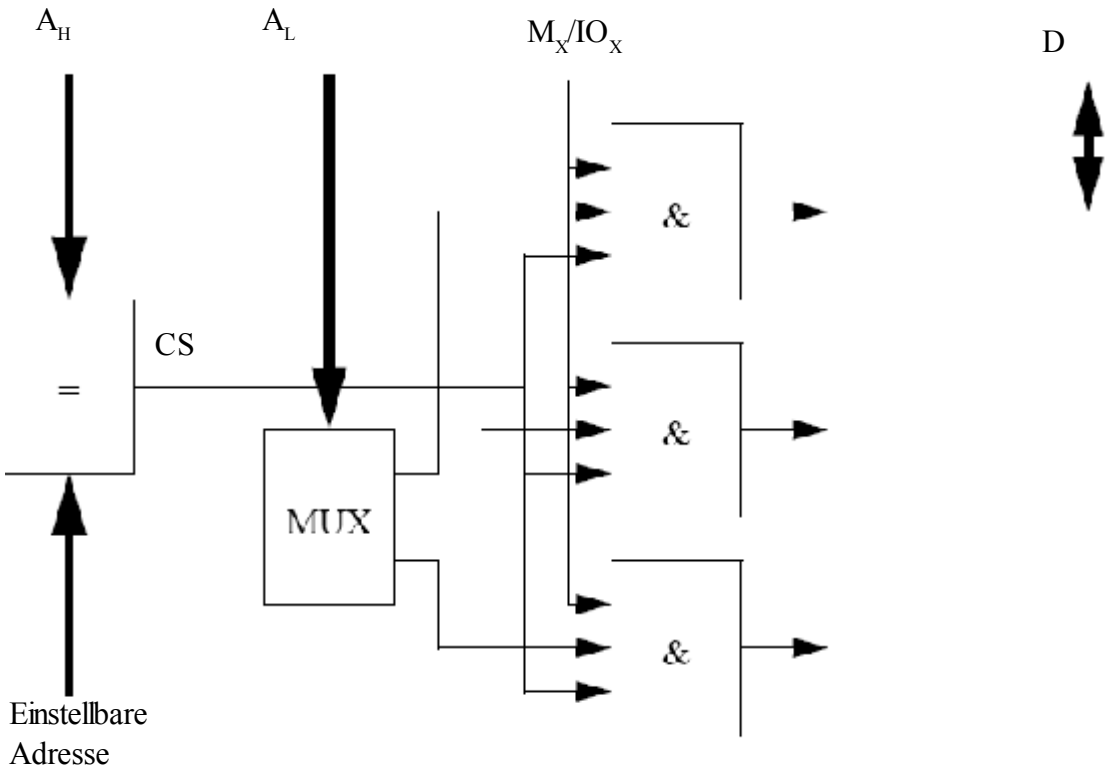
4b)Compilieren Sie die Funktion „getnum“! Generieren Sie Assembler-Code für einen bekannten Prozessor! (14 Punkte)

```
/* c = (%esp)*/
/* num = 4(%esp)*/
/* sign = 8(%esp)*/


_getnum:
        push %ebp
        movl %esp, %ebp
        subl $12, %esp

        pushl 8(%ebp)
        call fgetc
        movl %eax, (%esp)

        cmpl $'-', c
        jne _else
        movl $1, 8(%esp)
        pushl 8(%esp)
        call fgetc
        movl %eax, (%esp)
_else:

        movl $1,8(%esp)
        movl $0, 4(%esp)
        jmp _test
```

```
_loop:  movl 4(%esp), %eax
        addl $10, %eax
        addl (%esp), %eax
        subl $'0', %eax
        movl %eax, 4(%esp)
        pushl 8(%esp)
        call fgetc
        movl %eax, (%esp)
_test:  movl (%esp), %eax
        cmpl $'0', %eax
        ja 'next
        cmpl $'9', %eax
        jbe _loop
_next:
        pushl 8(%ebp)
        pushl (%esp)
        call ungetc
        movl 8(%esp), %eax
        addl 4(%esp), %eax
        leave
        ret
```

Aufgabe 5: I/O(16 Punkte)

Das folgende Bild zeigt einen Teil eines typischen Aufbaus von I/O-Karten für Rechner:



5a) Beschriften Sie die Signale im Diagramm!(3 Punkte)

5b) Beschreiben Sie die Funktion der einzelnen Signale und die Aufgaben der einzelnen Logik-

Bauteile! (13 Punkte)

Mit dem Chip-Select-Signal wird die RAM-Speicherzelle aktiviert, mit der gearbeitet werden soll. Dadurch wird durch mehrere Speicherbausteine ein größerer Gesamt-Speicher konstruiert.

Beim Rest weiß ich noch nicht, was das soll: Super viel Erklärt und die Zeichnung im Script ist für'n Arsch!

Aufgabe 6: Interrupts(10 Punkte)

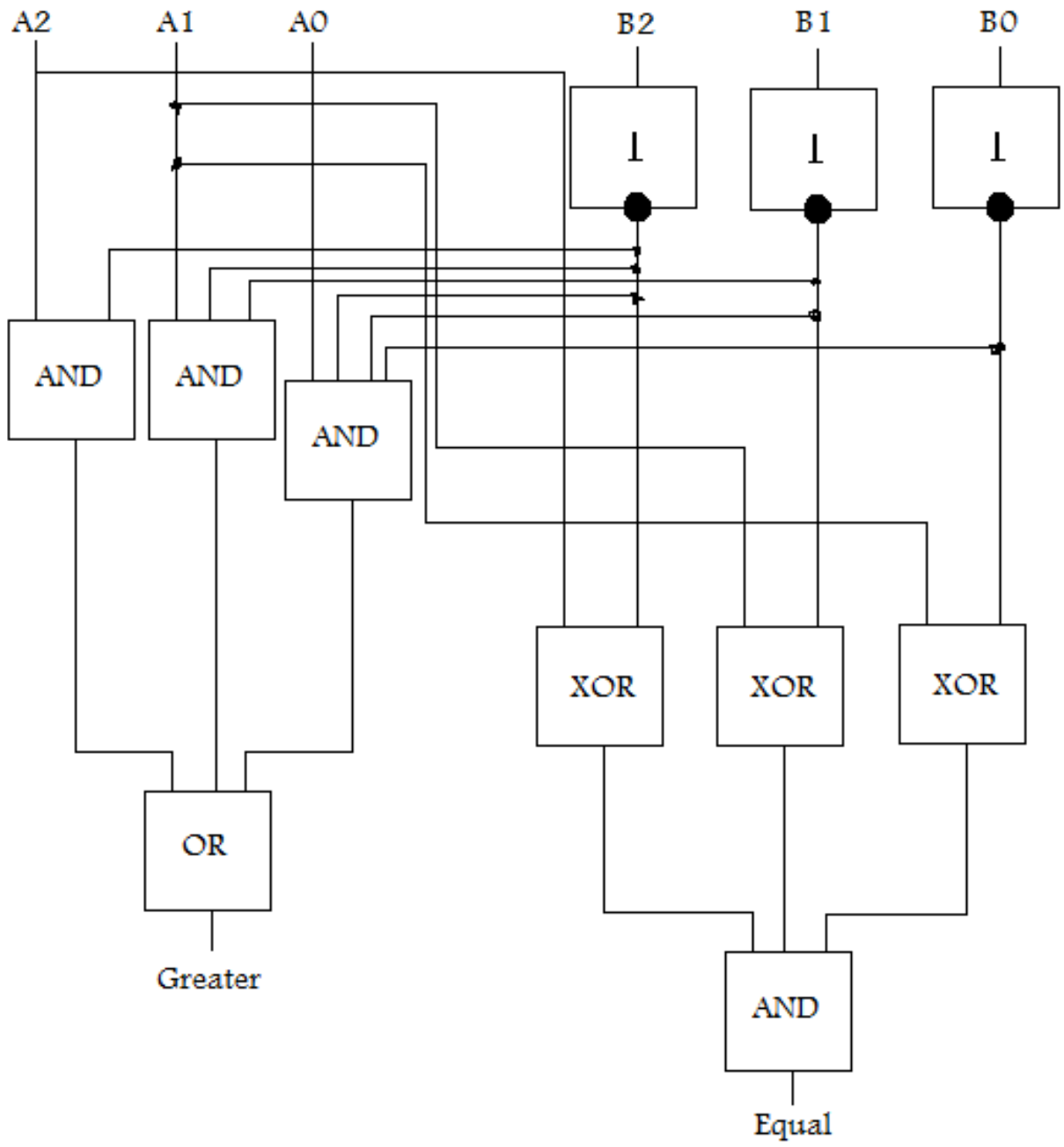
Beschreiben Sie, welche Schritte beim Auftreten eines Interrupts im User-Mode in einer CPU ablaufen, die zwischen User- und Supervisor-Mode unterscheidet! Achten Sie gegebenenfalls auf die Reihenfolge der Schritte! Welche Aktionen können gleichzeitig geschehen, welche Aktionen müssen nacheinander ausgeführt werden?
Lösung: (Hier muss noch geklärt werden, wie der Wechsel in Supervisermodus abläuft!!!)

1. Interuptquelle sendet Interruptrequest
2. CPU überprüft, ob Interrupt zulässig ist(nicht maskiert ist) und nimmt ihn ggf. an.
(Interrupt-Anforderungen bleiben so lange bstehen, bis sie akzeptiert werden!)
3. CPU schaut nach, ob Interrupt-Enable-Bit auf 0 ist. Falls "JA" => CPU quittiert den IRQ mit Interrupt-Acknowledge.
4. Abspeichern der Instruction-Pointers, einiger Register wie CC-Register, Abspeichern des Current-Privilege-Levels, Stackpointer für den Reibungslosen Rücksprung auf den Stack
5. Wechseln zu Supervisormodus.)
6. Sprung in die Interrupt-Service-Routine
7. Service-Routine sichert alle benutzen Register, damit Unterbrochene Programme nicht gestört werden und setzt das Interupt-Enable-Bit auf 0
8. Nach Ausführung der Routine: Laden einiger Register (z.B. neuer Wert für das Interrupt-Enable-Bit, Laden des neuen Current-Privilege-Levels)
9. Laden des Instruction-Pointers
10. Wiederaufnahme des Unterbrochenen Programms

Aufgabe 7: Hardware(12 Punkte)

Es soll ein Vergleicher aufgebaut werden. Zu vergleichen sind zwei vorzeichenlose je 3 Bit breite Operanden (A und B). Es sind zwei Signale auszugeben:- ob die beiden Operanden gleich groß sind (Equal) (4 Punkte)- ob A größer ist als B (Greater) (8 Punkte).
Für die Implementierung des Schaltnetzes dürfen AND-, OR-, XOR- sowie NOT-Gatter verwendet werden.

Manu's Lösung spricht für sich:


A2
A1
A0
B2
B1
B0
1
1
1
AND
AND
AND
XOR
XOR
XOR
OR
Greater
AND
Equal