Autor: Sebastian Sossalla

Disclaimer: Das sind Lösungsvorschläge, keine Musterlösungen.

Aufgabe 1: Allgemeines (8 Punkte)

1. Welche Aussagen sind wahr? (2 Punkte; Punktabzug bei falschen Antworten!)
<x> 32-Bit-Register können sowohl Signed- als auch Unsigned-Integer-Zahlen speichern.
<x> Mit dem gleichen Addierwerk können sowohl Signed- als auch Unsigned-Integer-Zahlen addiert werden.
< > Mit dem gleichen Addierwerk können sowohl Integer- als auch Fließkommazahlen addiert werden.
(bemerkung: bei Fließkomma-zahlen mit unterschiedlichen Exponenten muss erst geschiftet werden. Außerdem wird nur ein tteil der bytefolge addiert.)
<x> Alle Kontrollstrukturen können auf if-then-goto-Befehle abgebildet werden.
< > Computer mit Big-Endian-Format sind schneller als Rechner mit Little-Endian-Format.
< > Computer mit Big-Endian-Format brauchen mehr Speicher als Rechner mit Little-Endian-Format.
< > Beim Übertragen von Daten von einem Computer mit Big-Endian-Format auf einen Computer mit Little-Endian-Format sind alle Daten zu konvertieren.
(Byte-Variablen braucht man nicht zu konvertieren...
<x> Beim Übertragen von Daten von einem Computer mit Big-Endian-Format auf einen Computer mit Little-Endian-Format sind 16- und 32-Bit-Werte zu konvertieren.
(Ist richtig, hier macht die Byte-Order etwas aus...)

2. Für welche der nachfolgenden Konstrukte in einem Programm erzeugt ein Compiler ausführbaren Code? (2 Punkte; Punktabzug bei falschen Antworten!)
< > Typ-Definition
erzeugt keinen Assembler-Code.
< > Definition einer globalen Variablen
(Taucht zwar im Code auf, aber im Datensegment!)
<x> Definition mehrerer lokaler Variablen
(auf dem Stack-> also ja!)

< > Konstanten-Definition
<x> Initialisierung einer lokalen Variablen
< > Kommentar
<x> Variablen-Zuweisung
<x> for-Schleife

3. Erklären Sie den Begriff ,,effektive Adresse”! (2 Punkte)

Bei vielen Adressierungsarten wird zur Laufzeit eine sogenannte "effektive Adresse" dynamisch berechnet.

Beim Array-Zugriff zum Beispiel hat man eine feste Basisadresse der Datenstruktur, und einen variablen Index. Die effektive Adresse eines Feldeintrags wird dann berechnet durch Basisadresse + Index * Arrayeintrag (Scalefactor)

Ein anderes Beispiel wäre ein Zugriff auf den Stack verschoben um einen Eintrag: dann ist die effektive Adresse = Basisadresse + displacement.
Beispiel: movl 4(%ebp), ...

4. Warum haben Shift-Befehle auf kleinen Micro-Controllern so eine große Bedeutung? (2 Punkte)

Shift befehle können sehr einfach nachgebaut werden, sind extrem schnell und ausserdem relativ billig. Hinzukommt, dass durch Schift-arithmetik und addition auch noch Multiplikationen und Divisionen einfach nachgebildet werden können.

Aufgabe 2: Main-Bus (12 Punkte)

1. Füllen Sie folgende Tabelle nach folgendem Muster aus:
Tragen Sie in Felder der Tabelle jeweils

- -Eingang
- -Ausgang
- -Ein-/Ausgang

ein, je nachdem, ob die Leitung ein Eingang, ein Ausgang, oder ein Einund Ausgang der entsprechenden Komponente ist. Hat eine Leitung keine Verbindung zur Komponente, lassen Sie das Feld leer.
Drei Felder sind als Beispiel schon ausgefüllt.
(6 Punkte; Punktabzug für falsche Einträge)

	CPU	Speicher	IO-Gerät
MR	Ausgang	Eingang	
MW	Ausgang	Eingang	
IOR	Ausgang		Eingang
IOW	Ausgang		Eingang
Datenbus	Ein-/Ausgang	Ein-/Ausgang	Ein-/Ausgang
Adressbus	Ausgang	Eingang	Eingang
Interrupt	Eingang		

2. Wovon hängt es ab, wieviele Adressleitungen ein Main-Bus mindestens haben muss? (2 Punkte)

Das hängt von der Adressbusbreite ab. Man braucht für jedes Bit Adressbusbreite eben eine Leitung. 8 Adressbusbreite = 8 Bit.

3. Warum macht es keinen Sinn, z.B. 256 Adressleitungen in einen Rechner einzubauen? (2 Punkte)

Weil man damit eine maximal Adressraumgröße von 2^256 Byte hat, was völlig überdimensioniert ist, da man damit mehr Adressraumgröße, als das Universum Sterne hat...

4. Welche Vorteile hat man, wenn man die ALU, die CPU-internen Datenpfade und -Register von 8 Bit auf 32 Bit verbreitert? (2 Punkte)

Weil man dann statt 8 Bit 32 Bit mit einem Speicherzugriff holen kann und auch mit 32 Bit statt 8 Bit Werten rechnen kann -> Geschwindigkeit, schnelleres Lesen, schnelleres Cachen und schnellere Arithmetik.

Aufgabe 3: Speicher (8 Punkte)

Eine Firma speichert bisher ihre Datensätze der Form

```
struct data {
        struct data *next; /* Adresse des naechsten Datensatzes */
        char name[10]; /* Name des Mitarbeiters */
```

```
        int gehalt; /* Gehalt des Mitarbeiters */
    }
```

auf einem Rechner mit 8-Bit-Maschinenwörtern und 16-Bit-Adressen. Variablen vom Typ ,,char" werden in 8 Bit kodiert, ,,int" in 32 Bit. 32 KByte des Speichers werden vom Programm belegt.

1. Wieviele solche Datensätze passen maximal in den Rechner? Begründen Sie Ihre Antwort! (4 Punkte)

Größe der Datenstrucktur
2 Byte (für Pointer next) + 10 Byte (für name-Array) + 4 Byte (integer Gehalt) = 16 Byte.

(2^16 * 8 – 48 * 1024 * 8) / 16 * 8 = (2^16 – 48 * 1024) / 16
= 2^16 – 48 * 2^10) / 16 = 2^10 (2^6- 48) / 16 = 2^10 (2^6 – 48)/ 2^4
= 2^6 (2^6 – 48) = 2^12 – 2^6 * 48 = 2^12 – 2^6*2^4*3 = 2^12 – 2^10*3
= 2^10 (2^2 – 3) = 2^10 = 1024
Es passen 1024 Datensätze maximal in den Rechner!

Da der Speicher des alten Rechners vollständig belegt ist, soll der alte Rechner durch einen neuen mit 64-Bit-Adressen und 64-Bit-Maschinenwörtern ersetzt werden. Wieviel Platz nehmen die alten Daten jetzt ein (Hinweis: beachten Sie das neue Alignment) ? Begründen Sie Ihre Antwort! (4 Punkte)

```
struct data {
        struct data *next; /* Adresse des naechsten Datensatzes */
        char name[10]; /* Name des Mitarbeiters */
        int gehalt; /* Gehalt des Mitarbeiters */
        char dummy[2]
    }
```

2^6/2^3 =2^3 = 8 Byte

next	1 2 3 4 5 6 7 8	9 10 int						

data Größe = 8 + 10 + 4 + 2 = 24 Byte
alte Größe = 16 Byte. 32 * 1024 / 16 = 2^5 * 2^10 / 2^4 = 2^11 = 2048 Datensätze.
neue belegte Größe = 2^11 * 20 Byte = 2 * 1024 * 20 = 40 Kilobyte

Aufgabe 4: Ausdrücke (8 Punkte)

Was müsste alles programmiert werden, um auf einem einfachen 8-Bit-Microcontroller ohne Befehle für Integer-Multiplikation und -Division sowie ohne Gleitpunktoperationen die Funktion sinx berechnen zu können? Schreiben Sie keine Programme, sondern skizzieren Sie nur grob, welche Operationen Sie durch welche einfacheren Konstrukte nachbilden könnten!
Lösung:

- sin(x) kann mittels Taylor-Reihe in eine einfachere Funktion mit Gleitpunkt-operationen umgewandelt werden.
- Die Gleitpunkt-Division kann mittels Newton-Raphson auf Gleitpunktaddition und Gleitpunktmultiplikation zurückgeführt werden.
- Gleitpunktoperationen (+, -, *) können durch getrennte Behandlung von Vorzeichen, Mantisse und Exponent auf Ganzzahloperationen zurückgeführt werden.
- Integermultiplikation kann durch Shift und Add-Operationen ersetzt werden.
- Integer Operationen auf großen Integer-Zahlen können mittels mehrfacher Berechnung unter Einbeziehung des Carry-Bits ausgeführt werden.

Aufgabe 5: Unterprogramme (12 Punkte)

1. Schreiben Sie eine Assembler-Prozedur memcpy, die von einem Speicherbereich
Bytes in einen anderen Speicherbereich kopiert. (7 Punkte)
Der Prozedur werden

- die Zieladresse als 4-Byte-Wert
- die Quelladresse als 4-Byte-Wert
- die Anzahl der zu kopierenden Bytes als 4-Byte-Unsigned-Integer-Wert

in dieser Reihenfolge auf dem Stack übergeben.
Eine Optimierung ist nicht gefordert!
memcpy:

Lösung:
c-signatur wäre: memcpy(int ziel, int dest, unsigned int size)

```
memcpy:
        pushl %ebp
        movl %esp, %ebp          oder: enter $4
        subl 4, %esp             /* tmp = 0*/

        movl 8(%ebp), %ebx       /* size = ebx*/
        movl 12(%ebp), %edx      /* dest = edx*/
        movl 16(%ebp), %ecx      /* ziel = ecx*/

        movl $0, %esp            /* tmp = 0*/

_test: cmp %ebx, tmp
        jle _loop                /*while (tmp <= size)... tue*/
        goto _end

_loop:
        movb (%ecx, tmp, 1),  %eax
        movb %eax, (%edx, tmp, 1)   /* Byte verschieben */
        goto _test

_end  addl $4, %esp
        movl %ebp, %esp          oder einfach nur: leave
        popl %ebp
        ret
```

Aufgabe 5: Unterprogramme (Fortsetzung)

2. Angenommen, die Anzahl der zu kopierenden Bytes sei bei den meisten memcpy-Aufrufen relativ groß. Die Laufzeit der memcpy-Prozedur könnte dann auf einem 32-Bit-Rechner u.U. durch Long- statt Byte-Zugriffe verbessert werden.

• Warum? (1 Punkt)
Bei einem Long-Zugriff werden 4 Bytes auf einmal gelesen, anstatt 4 einzelne Byte-Zugriffe. Das ist nur ein Lesevorgang anstatt vier und spart dadurch beträchtlich Zeit.

Wann ist diese Optimierung u.U. nicht nutzbar? (1 Punkt)

??? noch keine Ahnung...

Welche Eigenschaften müssen die Ziel- und Quelladressen besitzen, damit der Kopiervorgang auf jeden Fall besonders schnell ablaufen kann? (3 Punkte)

??? noch keine Ahnung...

Aufgabe 6: Kontrollstrukturen/Ausdrücke (8 Punkte)

Gegeben Sei folgende Funktion:

```
int fac(int x) {
  int res;
        if (x <= 1) {
                res = 1;
        } else {
                res = fac(x - 1) * x;
        }
  return res;
}
```

Konvertieren Sie die Hochsprachen-Funktion in semantisch äquivalenten Hochsprachen-Code, so dass sich dieser möglichst leicht in Assembler-Code für eine Zwei-Adress-Maschine umwandeln lässt!

Hinweis: Zwei-Adress-Maschine bedeutet, dass bei Berechnungen nur Ausdrücke der Form var = var <op> operand; erlaubt sind (op ist Element von {+,−, , /,< ,>,<=,>=}, var ist eine Variable, operand kann eine Variable oder eine Konstante sein). Hinweis: if-then-else-Blöcke sollen in goto- bzw. if-then-goto-Anweisungen umgewandelt
werden.

```
int fac(int x) {
  int res;
        if (x <= 1) {
                res = 1;
        } else {
                res = fac(x - 1) * x;
        }
  return res;
}

fac:
  int res;
        if (x <= 1)  goto if;
        goto _else;
_if:
        res = 1;
        goto _end;
_else;
        int tmp = x;
        tmp = tmp -1;
        res = fac(tmp);
        res = res * x;
_end;
  return res;
}
```

Aufgabe 7: Unterprogramme (6 Punkte)

1. Welche Möglichkeiten gibt es in Assembler-Code, Parameter an Unterprogramme zu übergeben? Nennen Sie die Methoden sowie ihre Vor- und Nachteile!
(3 Punkte)

- Paramter per Register zu übergeben.
  Vorteil: sehr schnell
  Nachteil: Nur begrenzt viele Register vorhanden. Also können nur begrenzt viele Parameter übergeben werden. Register müssen evtl. vorher gesichert werden. Rekursionen sind nur begrenzt möglich umzusetzen. Bei vielen

unterprogramm-verschachtelungen wird man mit dieser Methode auf Grenzen stoßen, wegen begrenzter Register-Anzahl

- Parameter per Stack übergeben:
  Vorteil: es kann theoretisch eine beliebige Anzahl Parameter übergeben werden. Unterprogramm-verschachtelungstiefe theoretisch unbegrenzt.
  Nachteil: wesentlich langsamer als auf Art-und Weise wie oben, Der Stack muss nach dem Unterprogramm vom Programm selbst oder vom Aufrufer wieder geleert werden.
- Mischung aus Register und Stack

2. Welche Möglichkeiten gibt es in Assembler-Funktionen, Werte an das aufrufende Programm zurückzugeben? Nennen Sie die Methoden sowie ihre Vor- und Nachteile! (3 Punkte)

- Per Register:
  Vortei: sehr schnell
  Nachteil: Register können evtl überschrieben werden bei falschen Standards
  Nur eine begrenzte Anzahl an Register vorhanden.
  Register müssen evtl gesichert werden, vor Programmaufruf
- Per Stack:
  Vorteil: theoretisch unendlich viele Parameter können zurückgegeben werden.
  Nachteil: wesentlich langsamer, der Stack muss wieder geleert werden, Die Daten können nur anhand der Reihenfolge, wie sie auf dem Stack liegen, identifiziert werden.

Aufgabe 8: Protection (8 Punkte)

Wenn ein Rechner Multi-Prozess- und Multi-User-fähig sein soll, muss sichergestellt sein, dass parallel ablaufende Prozesse sich gegenseitig nicht gefährden/stören können.

1. Was bedeutet diese Anforderung für I/O-Befehle? (2 Punkte)

Ein Prozess im User-Modus darf nicht direkt auf I/O-Geräte zugreifen. Das bedeuted, dass die Nutzung der I/O-Befehle im User-Modus nicht erlaubt ist und zu Exceptions führt. Ein und Ausgabe nur über System-calls möglich. Durch eines System-calls wird vom User Modus in den Privilegierten Modus gewechselt und die I/O-Zugriffe durchgeführt. I/O-Befehle arbeiten auf einem anderen Adress-

bereich als normale Befehle wie mov, add etc. Ausserdem muss eine Sceduling Strategie angewandt werden, damit die Leistung auf alle User gleich verteilt wird.

2. Was bedeutet diese Anforderung für Speicherzugriffe? (2 Punkte)

Lese/Schreib-Zugriffe auf fremde Speicherbereiche darf im User-Modus nicht erlaubt sein. Ein Prozess im privilegierten Modus kann alle speicherzellen beschreiben/lesen. Das ist notwendig bei Prozess-wechsel.

3. Was bedeutet diese Anforderung für die Interrupt-, Exception- und System-Call-Behandlung? (2 Punkte)

Durch die Segmentierung muss während der Umschaltphased vom User-Modus -> Privilegirter Modus:

- Abspeichern der Instrucion-Pointer, Stack-Pointer, CC-Register, Segment-Register, und des Current-Privilege-Levels zunächst in temporären Registern.
- Laden der neuen Werte für IP, SP-, CC-, Segment- und CPL-Register
- Abspeichern der temporären Register auf dem neuen Stack.
durchgeführt werden.

4. Was muss die Hardware leisten? Was leistet das Betriebssystem? (2 Punkte)
Beschränkung auf Hardware-Ebene durch Basis-Register und Längen des Prozess-Segments. Weitere Einschränkung durch verschiedene logische Segmente pro Prozess: (Code-, Daten-, Stack-Segment). Weitere Verbesserung: Start-Adresse aus Sicht des Prozesses im User-Modus ist immer 0. Und das Betriebssystem bestimmt die Lage im Speicher.

Aufgabe 9: MMU (10 Punkte)

Da hab' ich ein Beispiel eingescannt - is 'ne Handskizze und online gestellt!

Aufgabe 10: Hardware (ALU) (10 Punkte)

Es soll ein Dekrementierer 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 Unterlauf-Ereignis über eine fünfte Leitung zu signalisieren. Für die Implementierung des Schaltnetzes dürfen

AND-, OR-, XOR- sowie NOTGatter verwendet werden. (10 Punkte; Punktabzug für unnötig aufwändige Schaltungen!)

Benutze Volladdierer.
Volladdierer-zelle hinmalen spar' ich mir jetzt. Allerdings net in der Klausur ;-)