Autor: Sebastian Sossalla

Disclaimer: Das sind Lösungsvorschläge, keine Musterlösungen.

Aufgabe 1: Allgemeines (13 Punkte)

1. Welche Aussagen sind wahr, welche sind falsch? (4 Punkte; Punktabzug bei falschen Antworten!)

wahr	falsch	
<x>	< >	Bei Little-Endian steht das niederwertige Byte an der niedrigsten Adresse.
< >	<x>	Für sinnvolle Programme sind mindestens 8 Adressierungsarten notwendig.
< >	<x>	Überläufe bei Berechnungen des Instruction-Pointers werden im Carry-Flag gespeichert.
< >	<x>	Das Carry-Flag ist Teil des Program-Counters.
<x>	< >	Mit Hilfe des Carry-Flags können beliebig große Integer-Zahlen miteinander addiert werden.
<x>	< >	In Interrupt-Handlern muss mit bestimmten ret-Befehlen zum Hauptprogramm zurückgekehrt werden.
<x>	< >	Exceptions und System-Calls haben große Ähnlichkeiten.
<x>	< >	Shift-Operationen sind leicht in Hardware zu bauen.

Welche der nachfolgenden Informationen werden i.a. beim Speicherzugriff in der Paging-Einheit zur Erlaubnisprüfung genutzt? (4 Punkte; Punktabzug bei falschen Antworten!)

genutzt	nicht genutzt	
<x>	< >	aktueller Privilege-Level
< >	<x>	virtuelle Adresse
< >	<x>	physikalische Adresse
<x>	< >	Read- oder Write-Zugriff
< >	<x>	Segment-Nummer
<x>	< >	Read-only-Bit in Page-Tabelle
< >	<x>	Zero-Bit
< >	<x>	Adressierungsart

3. Was sind die Unterschiede von mov-Befehlen und in- bzw. out-Befehlen? (3 Punkte)

Der gravierende Unterschied von mov-Befehlen zu in/out-Befehlen ist der, dass bei mov-Befehlen der normale Adressraum be nutzt wird, und bei in/out-Befehlen Teile des speziellen I/O-Bereichs angesprochen werden. in/out-Befehle sind System-calls, mov-befehle hingegen normale Schiebe-befehle!

4. Warum muss jede CPU vor Abarbeitung eines Interrupt-Handlers die Interrupt-Erkennung abschalten? (2 Punkte)

Da ansonsten der Interrupt-Handler beim abarbeiten erneut durch einen Interrupt unterbrochen werden würde. Es handelt sich also beim Interrupt-Handler um einen kritischen Abschnitt, bei dem keine Nebenläufigkeit entstehen darf!

Aufgabe 2: Hardware (12 Punkte)

1. Fülen 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 Ein-und Ausgang der entsprechenden Komponente ist. Hat eine Leitung keine Verbindung zur Komponente, lassen Sie das Feld leer.
Drei Felder sind als Beispiel schon ausgefüllt.

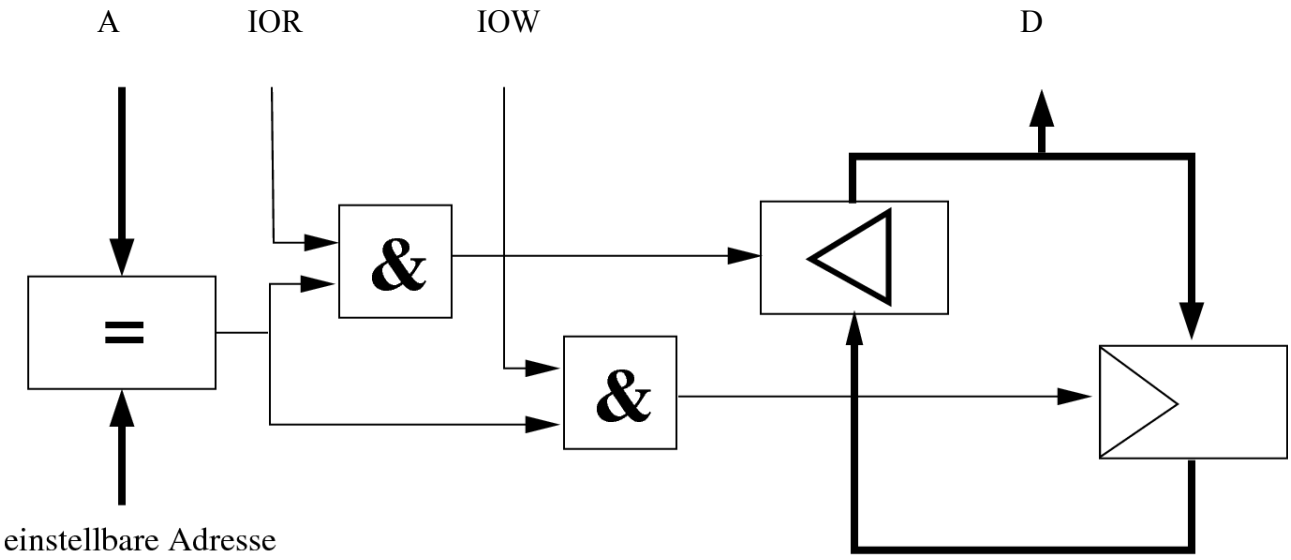
(6 Punkte; Punktabzug für falsche Einträge)

	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		

Aufgabe 2: Hardware (Fortsetzung)

2. Zeichnen Sie den Schaltplan eines möglichen Aufbaus eines I/O-Gerätes,

das einen Sensor auslesen und einen Aktor steuern kann! (6 Punkte)



Aufgabe 3: Speicher (12 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 */
}
```

next	12	34	56	78	910	gehalt					

Alignment passt!

auf einem Rechner mit 8-Bit-Maschinenwörtern und 16-Bit-Adressen. Variablen vom Typ ,,char" werden in 8 Bit kodiert, ,,int" in 32 Bit. 48 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!

2. Da der Speicher des alten Rechners vollständig belegt ist, soll der alte Rechner durch einen neuen mit 32-Bit-Adressen und 32-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 */
        char dummy[2];
        int gehalt; /* Gehalt des Mitarbeiters */
}
```

next	1234	5678	910	gehalt					

Alignment passt nicht, also Dummy einfügen.

Anzahl der alten Datensätze:
2 Byte (für Pointer next) + 10 Byte (für name-Array) + 4 Byte (integer Gehalt) = 16 Byte. 48 * 1024 Byte / 16 Byte = 3 * 1024 Datensätze = 3072 Datensätze.

Anzahl der neuen Datensätze:
4 Byte + 10 Byte + 4 Byte+ 2 Byte = 20 Byte.
20 Byte * 3 * 1024 (3072) Datensätze = 60 Kilobyte.

Aufgabe 3: Speicher (Fortsetzung)

3. Schreiben Sie für eine Little-Endian-CPU ein kleines Assembler-Unterprogramm, das eine auf dem Stack übergebene 4-Byte-Integerzahl in den Speicher im Big-Endian-Format schreibt! Die Zieladresse wird ebenfalls als 4-Byte-Wert auf dem Stack übergeben. (4 Punkte)

```
littletobig:
        pushl %ebp, %esp
```

```
movl %esp, %ebp                    enter $4
subl $4, %esp

movl 8(%ebp), %eax   /* Adresse aus dem Stack holen...*/
movb 12(%ebp). %ah
movb %ah, 3(%esp)
movb 13(%ebp), %ah
movb %ah, 2(%esp)
movb 14(%ebp), %ah
movb %ah, 1(%esp)
movb 15(%ebp), %ah
movb %ah, 0(%esp)
movl (%esp), (%eax) /*big-Endian-Zahl an Speicher-Adresse schreiben!!!*/

addl $4, %esp                      leave
movl %ebp, %esp
popl %ebp
ret
```

Aufgabe 4: Ausdrücke (18 Punkte)

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

- tan(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 4: Ausdrücke (Fortsetzung)

2. Schreiben Sie eine Hochsprachen- oder Assemblerfunktion, die zwei übergebene 16-Bit-Unsigned-Integer-Zahlen multipliziert und das Ergebnis als 32-Bit-Unsigned-Integer-Zahl zurückliefert! Es darf in der Funktion keine

Multiplikation verwendet werden. Shift-Operationen sind erlaubt. (10 Punkte)

schau hi schau her: eine 1:1 Übungsaufgabe:

```
unsigned long mult( unsigned short x, unsigned short y) {
        unsigned long d = (unsigned long) x;
        unsigned long res = 0;

        for (int i = 0; i < 16; i++) {
                d = d  << 1;
                res = res << 1;
                if (0x10000 <= d) {
                        d -= 0x10000;
                        res += y
                }
        }
}
```

Aufgabe 5: Unterprogramme (16 Punkte)

Gegeben Sei folgende Funktion:

```
unsigned int fibo(unsigned int x) {
   unsigned int res;
        if (x < 2) {
                res = 1;
        } else {
                res = fibo(x - 1) + fibo(x - 2);
        }
   return res;
}
```

1. 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. (8 Punkte)

Schritt 1:

```
unsigned int fibo(  unsigned int x) {
```

```
        usigned int res;
        if (x < 2) goto _if
_elseif:
        tmp1 = x-1;
        tmp2 = x-2;
        res = fibo(tmp1);
        res = res + fibo(tmp2);
        goto _end;
_if:    res = 1;
_end: return res;
}
```

Aufgabe 5: Unterprogramme (Fortsetzung)

2. Geben Sie Assembler-Code einer Zwei-Adress-Maschine an, der semantisch äquivalent zur gegebenen fibo-Funktion ist! (8 Punkte)

Schritt 1:

```
unsigned int fibo(  unsigned int x) {
        unsigned int res;
        unsigned int tmp1;
        unsigned int tmp2;
        if (x < 2) goto _if;
_elseif:
        tmp1 = x-1;
        tmp2 = x-2;
        res = fibo(tmp1);
        res = res + fibo(tmp2);
        goto _end;
_if:    res = 1;
_end: return res;
}
```

```
fibo:   pushl %ebp
        movl %esp, %ebp          enter $12
```

```
        subl $12, %esp

        movl 8(%ebp), %ebx    /* parameter x holen*/
        cmp $2, %ebx          /* if (x < 2)...*/
        jl _if
_elseif:
        movl %ebx, 4(%esp)  /* tmp1 = x*/
        movl %ebx, (%esp)   /* tmp2 = x*/
        subl $1, 4(%esp)      /* tmp1 -= 1*/
        subl $2, (%esp)      /* tmp2 -= 2*/
        pushl 4(%esp)
        call fibo
        movl %eax, 8(%esp)  /*res = fibo(tmp1)*/
        pushl (%esp)
        call fibo
        addl %eax, 8(%esp)
        jmp _end
_if:    movl $1, 8(%esp)       /* res = 1*/
_end:   movl 8(%esp), %eax
        addl $12, %esp

        movl %ebp, %esp          leave
        popl %ebp
        ret
```

Aufgabe 6: Protection (8 Punkte)

Eine CPU biete eine Memory-Management-Unit mit folgenden Eigenschaften:

- zweistufige Adresstabellen
- je 1024 Einträge zu je 4 Byte in den Tabellen
  - Bit 31-12: höherwertige Bits der physikalischen Adresse
  - Bit 11-2: unbenutzt
  - Bit 1: Write-enable-Bit
  - Bit 0: Present-Bit
- Pages zu je 4 kByte Größe

Folgendes soll für ein Programm erfüllt sein:

- im virtuellen Adressbereich 0x00000000 bis 0x00001fff soll auf einen ROM-Baustein zugegriffen werden (Code),
- im virtuellen Adressbereich 0x00010000 bis 0x00010fff soll auf Hauptspeicher zugegriffen werden (Daten),
- im virtuellen Adressbereich 0xfffff000 bis 0xffffffff soll auf Hauptspeicher

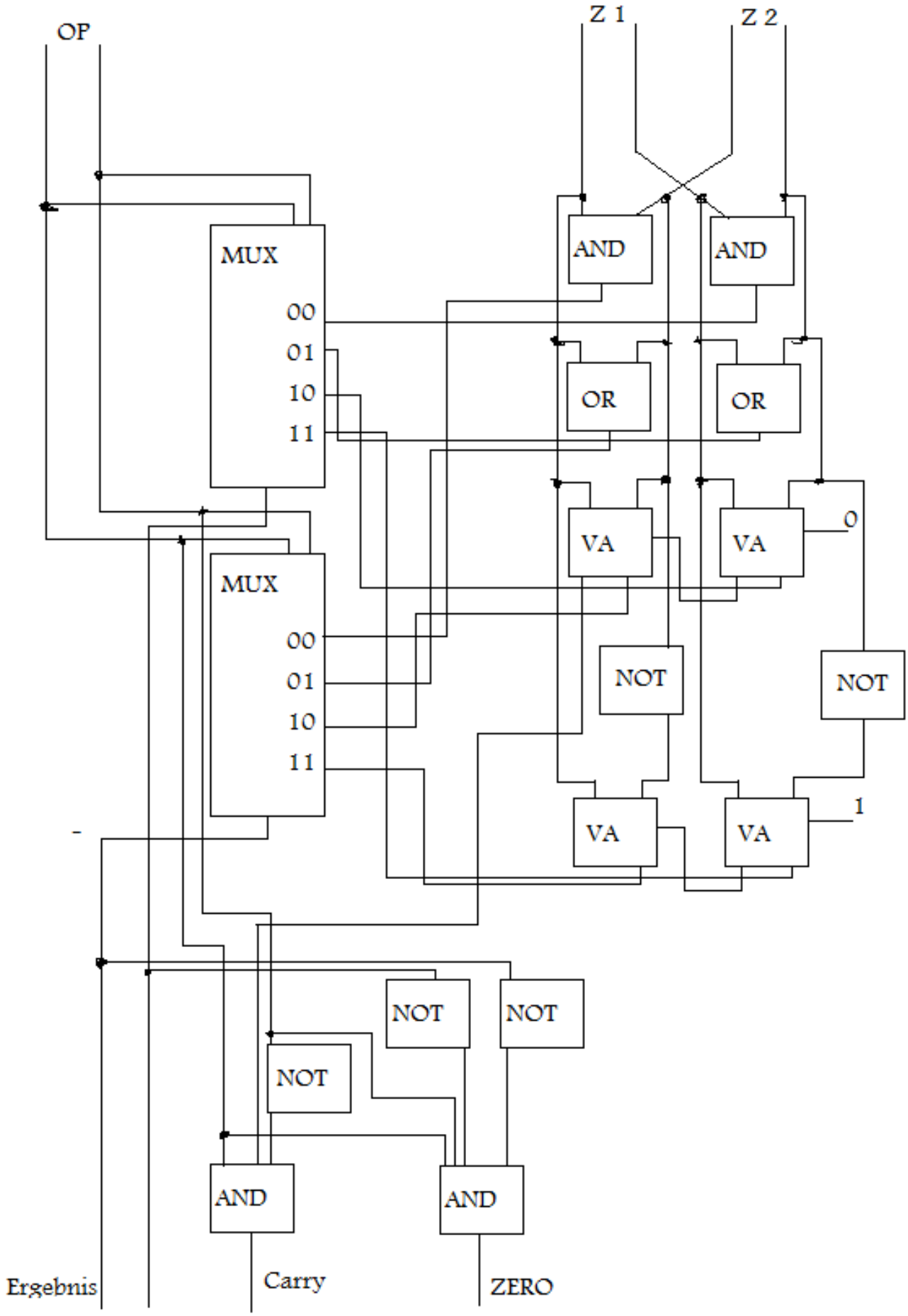
zugegriffen werden (Stack),

- alle anderen Bereiche sollen Zugriffsfehler auslösen.

  Physikalisch sind der ROM-Baustein an Adresse 0xffffe000 und der Hauptspeicher von Adresse 0x00000000 bis 0x80000000 am Bus. Beschreiben Sie eine m¨ogliche Page-Tabelle, die obige Bedingungen erfüllt! (8 Punkte)

Aufgabe 7: Hardware (ALU) (11 Punkte)

Zeichnen Sie das Schaltbild einer kleinen ALU.
Als Eingabe dienen zwei Operanden mit je 2 Bit sowie zwei Leitungen zum Signalisieren der Operation (00: And, 01: Or, 10: Add, 11: Sub).
Als Ausgabe sind 2 Bit für das eigentliche Rechenergebnis sowie ein Carry- und ein Zero-Bit vorgesehen.
Für das Schaltbild dürfen Volladdierer, Multiplexer und AND-, OR- und NOT Gatter (jeweils beliebiger Breite) verwendet werden.


OP
Z 1
Z 2
MUX
00
01
10
11
AND
AND
OR
OR
VA
VA
0
MUX
00
01
10
11
NOT
NOT
VA
VA
1
-
NOT
NOT
NOT
AND
AND
Ergebnis
Carry
ZERO