Klausur (27.07.2007):
Technische Grundlagen der Informatik 2
Rechnerorganisation
SS 2007

Vorname	:	.....
Name	:	.....
Matrikelnummer	:	.....
Studiengang	:	.....

Aufgabe	1	2	3	4	5	6	7	8	9	$\sum$
max. Punkte	11	7	8	12	8	10	15	15	14	100
erreichte Punkte										
Korrektor										

Wichtige Hinweise:

- Deckblatt ausfüllen
- Kopf aller abgegebenen Seiten ausfüllen, d.h. mit Namen und Matrikelnummer versehen
- für die Lösungen sind die Aufgabenblätter zu verwenden
- für die Lösungen darf weder Bleistift noch Rotstift verwendet werden
- der Lösungsweg muss nachvollziehbar sein
- Taschenrechner, Vorlesungsskript und Übungsmitschriften sind **nicht** erlaubt

- Mobiltelefone sind auszuschalten
- Betrugsversuche werden mit einem Nichtbestehen der Klausur geahndet

Aufgabe 1 (11 Punkte)

(a) Nennen Sie drei Formen der Darstellung negativer Zahlen im Binärsystem und stellen Sie jeweils die Zahl –4 als 8 Bit breite Zahl dar.

(b) Beschreiben Sie das Vorgehen bei Zero-Extension (Nullerweiterung) und bei Sign-Extension (Vorzeichenerweiterung) und geben Sie die Wirkung beider Verfahren an.

(c) Grenzen Sie die Begriffe Assemblerprogramm, Maschinenprogramm, Mikroprogramm gegeneinander ab.

(d) Nennen Sie drei typische Register eines Interface-Adapters und deren Nutzen.

(e) Beschreiben Sie das Prinzip der Fließbandverarbeitung (pipelining) und deren Nutzen.

Aufgabe 2 (7 Punkte)

(a) Geben Sie zu den beiden Rechenbeispielen das jeweilige Bitmuster des Ergebniswerts sowie die jeweiligen Zustände der Condition-Code-Bits an.

			1	0	1	0	1	0					0	0	1	1	0	1		
	+		1	0	0	1	0	1			−		0	0	1	1	1	1		

(b) Wandeln Sie die 2-Komplementzahl 1010.0110 in eine Gleitpunktzahl mit folgendem Format um:
1 Vorzeichenbit s, 4 Exponentenbits e, 11 Fraction-Bits f, (Bias b = 7)

Aufgabe 3 (8 Punkte)

Es sind die Zahlen 100111001 (Multiplikand) und 11100111 (Multiplikator) mit Hilfe des Booth-Algorithmus miteinander zu multiplizieren.

(a) Führen Sie zunächst die notwendigen Umformungen (Komplementbildung, Booth-Recording) durch!

(b) Multiplizieren Sie nun die beiden Zahlen miteinander!

	1	0	0	1	1	1	0	0	1	×	1	1	1	0	0	1	1	1	

(c) Angenommen, Multiplikand und Multiplikator würden vertauscht. Inwiefern würde sich dann der Berechnungsaufwand verändern, wenn die Multiplikation wiederum mit dem Booth-Algorithmus durchgeführt werden würde? Notieren Sie die zur Ermittlung des quantitativen Unterschiedes notwendigen Zwischenschritte!

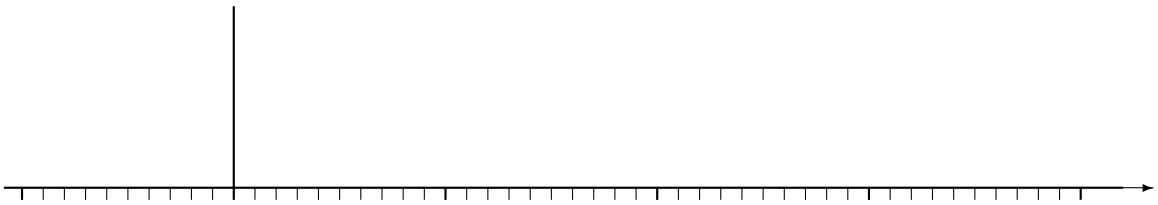
Aufgabe 4 (12 Punkte)

Führen Sie mit den Zahlen 0100100 (Dividend) und 0101 (Divisor) eine Division **ohne** Rückstellen des Zwischenrestes durch.

(a)

	0	1	0	0	1	0	0	/	0	1	0	1	=						

(b) Stellen Sie den Verlauf der Division, d.h. jeden Teilschritt, auf dem Zahlenstrahl dar. Beschriften Sie die wichtigen Punkte und die Sprünge!



(c) Nennen Sie für den obigen Dividenden den größten Divisor der zu einem Divisionsüberlauf führen würde!

Aufgabe 5 (8 Punkte)

Gegeben ist das folgende VIP-Assemblerprogramm:

```
1  ORG     3
2  U       RES     1
3  V       EQU     3
4  W       DAT     3,4,5
5  Start   LDX     #V
6          LDA     U[IX]
7          ADD     W
8          STA     @W
9          HLT
10         END     Start
```

Geben Sie für die relevanten Codezeilen die Adressierungsart und alle neugeschriebenen Werte mit den dazugehörigen Zieladressen (Register, Speicherzelle) an.

Nr.	Adressierungsart	Zieladresse	Wert

Aufgabe 6 (10 Punkte)

Gegeben ist das folgende VIP-Assemblerprogramm:

```
1   ORG     15
2   Cnt     RES     1
3   Len     EQU     4
4   Arr     DAT     1,2,3,4,5
5   Start   LDX     #0
6           STX     Cnt
7   L1      LDA     Arr[IX]
8           ADDX    #1
9           LSR
10          BCC     L2
11          LDA     Cnt
12          ADD     #1
13          STA     Cnt
14  L2      CMPX    #Len
15          BNE     L1
16          HLT
17          END     Start
```

(a) Assemblieren Sie das Programm bis einschließlich Zeile 10!
Notieren Sie die Werte hexadezimal.

Speicheradresse	Inhalt

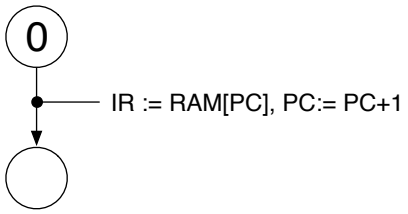
(b) Beschreiben Sie **kurz** die Funktionalität des Programms

Aufgabe 7 (15 Punkte)

Der Befehlssatz des VIP soll um einen Befehl `CMPAX` erweitert werden. Dieser vergleicht die Inhalte der Register AC und IX (AC–IX) und beeinflusst als Ergebnis ausschließlich die Bedingungsbits in SR. Alle Register- und Speicherinhalte, die möglicherweise nachfolgend noch benötigt werden, dürfen durch den Befehl nicht verändert werden.

Befehl	Code	z	n	c	v	#	@	[IX]	Wirkung
CMPAX	0xC0	x	x	x	x	$\surd$	-	-	-	SR neu setzen

(a) Entwickeln Sie den Zustandsgraphenausschnitt des VIP für den `CMPAX`-Befehl.

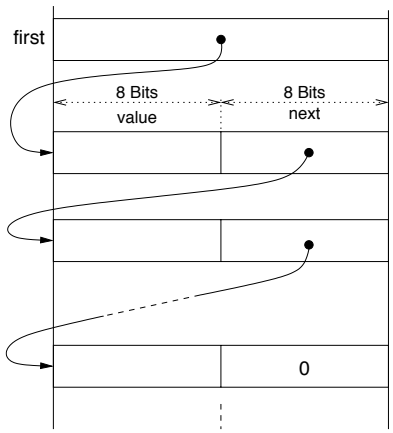


(b) Ergänzen Sie die PLA-Steuertabelle um die erforderlichen Zeilen.

	SR				IR		Register y_n								Mux y_n						ALU y_C
Z_i	z	n	c	v	Code	Z_j	0	1	2	3	4	5	6	D	7	8	9	A	B	E	$s_{0...4}u_0$
0	-	-	-	-	-		0	0	0	0	1	1	0	0	x	x	00	010	11	x	xxxxxx

Aufgabe 8 (15 Punkte)

Das nebenstehende Speicherabbild zeigt die Realisierung einer einfach verketteten Liste. Jedes Listenglied entspricht einem 16-Bit-Speicherwort und besteht aus einem Zahlenwert *value* und einem Zeiger *next* mit der Adresse des nächsten Listengliedes. Das letzte Glied der Liste wird durch *next = 0* gekennzeichnet. Die Speicherzelle mit der symbolischen Adresse *first* enthält den Zeiger auf das erste Listenglied. Mit dem Zeigerwert 0 wird eine leere Liste angezeigt.



Schreiben Sie ein VIP-Assemblerprogramm, das die Summe der Zahlenwerte ermittelt. Benutzen/ ergänzen Sie dazu die vorgegebenen Codefragmente.

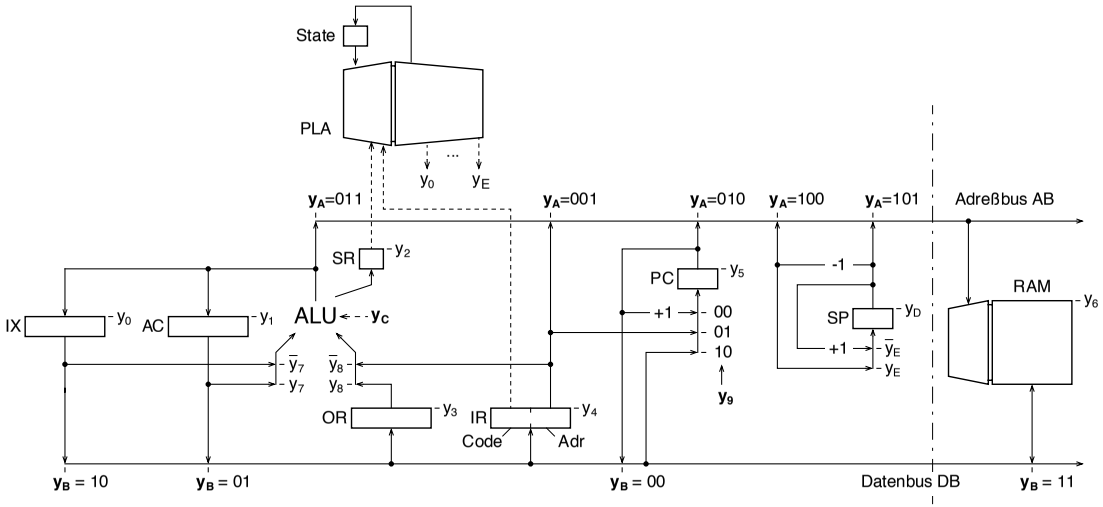
```
          ORG  0
first
liste     DAT  0x0105,0x0204,0x0300,0x0403,0x0502
sum


Start




          END  Start
```

Aufgabe 9 (14 Punkte)

Schreiben Sie ein Unterprogramm, das zwei mit Hilfe des Stacks übergebene Werte (call-by-value) addiert und das Ergebnis an eine ebenfalls übergebene Adresse schreibt (call-by-reference).

```
1  ORG     0
2  a       DAT     5
3  b       DAT     3
4  y       RES     1
5  Start   LDA     a
6          PUSH
7          LDA     b
8          PUSH
9          LDA #y
10         PUSH
11         JSR     Add
12         HLT
```

VIP-Strukturbild



ALU-Operationen

				arithmetische Operationen	logische Operationen
s_0	s_1	s_2	s_3	$s_4 = 0$	$s_4 = 1$
0	0	0	0	$Z = -1 + u_0$	$Z = 0$
0	0	0	1	$Z = X \vee Y + u_0$	$Z = \overline{X \vee Y}$
0	0	1	0	$Z = X \vee \overline{Y} + u_0$	$Z = \overline{X} \wedge Y$
0	0	1	1	$Z = X + u_0$	$Z = \overline{X}$
0	1	0	0	$Z = X \wedge \overline{Y} - 1 + u_0$	$Z = X \wedge \overline{Y}$
0	1	0	1	$Z = (X \vee Y) + (X \wedge \overline{Y}) + u_0$	$Z = \overline{Y}$
0	1	1	0	$Z = X - Y - 1 + u_0$	$Z = X \oplus Y$
0	1	1	1	$Z = (X \wedge \overline{Y}) + X + u_0$	$Z = \overline{X \wedge Y}$
1	0	0	0	$Z = (X \wedge Y) - 1 + u_0$	$Z = X \wedge Y$
1	0	0	1	$Z = X + Y + u_0$	$Z = X \equiv Y$
1	0	1	0	$Z = (X \vee \overline{Y}) + (X \wedge \overline{Y}) + u_0$	$Z = Y$
1	0	1	1	$Z = (X \wedge Y) + X + u_0$	$Z = \overline{X} \vee Y$
1	1	0	0	$Z = X - 1 + u_0$	$Z = X$
1	1	0	1	$Z = X \cdot 2^{-1}$	$Z = X \vee \overline{Y}$
1	1	1	0	$Z = X \triangleright 1$	$Z = X \vee Y$
1	1	1	1	$Z = X \cdot 2 + u_0$	$Z = 1$

$s_{0..4}$: ALU-Steuervektor ($y_C = s_{0..4}u_0$)
X/Y: linker/rechter ALU-Eingang (2-Komplement-Wert oder Bitvektor)
Z: ALU-Ausgang
$\triangleright$: Vorzeichenbehafteter Rechts-Shift

VIP-Befehlssatz

Befehl	Code (hex.)	SR-Inhalt z	n	c	v	Adressierung #	@	[IX]	Wirkung
HLT	00	-	-	-	-	-	-	-	Anhalten
NOP	01	-	-	-	-	-	-	-	Nichts
NOT	02	x	x	0	0	-	-	-	AC := ¬ AC, 1-Komplement
NEG	03	x	x	x	x	-	-	-	AC := 0 – AC, 2-Komplement
ASL	04	x	x	AC_{15}	x	-	-	-	AC := AC × 2
ASR	05	x	x	AC_0	0	-	-	-	AC := AC × 2^{-1}, AC_{15} := AC_{15}
LSL	06	x	x	AC_{15}	0	-	-	-	AC := AC × 2
LSR	07	x	x	AC_0	0	-	-	-	AC := AC × 2^{-1}, AC_{15} := 0
PUSH	08	-	-		-	-	-	-	RAM[SP–1]:= AC, SP := SP–1
POP	09	x	x	0	0	-	-	-	AC := RAM[SP], SP := SP+1
JMP	10	-	-	-	-	√	-	-	PC := Op
BPL	11	-	-	-	-	√	-	-	PC := Opif SR = ≥ 0
BMI	12	-	-	-	-	√	-	-	PC := Op if SR = < 0
BCC	13	-	-	-	-	√	-	-	PC := Op if SR = kein Übertrag
BCS	14	-	-	-	-	√	-	-	PC := Op if SR = Übertrag
BVC	15	-	-	-	-	√	-	-	PC := Op if SR = kein Überlauf
BVS	16	-	-	-	-	√	-	-	PC := Op if SR = Überlauf
BEQ	17	-	-	-	-	√	-	-	PC := Op if SR = = (Null)
BNE	18	-	-	-	-	√	-	-	PC := Op if SR = ≠ (Null)
BGT	19	-	-	-	-	√	-	-	PC := Op if SR = >
BGE	1A	-	-	-	-	√	-	-	PC := Op if SR = ≥
BLE	1B	-	-	-	-	√	-	-	PC := Op if SR = ≤
BLT	1C	-	-	-	-	√	-	-	PC := Op if SR = <
BGTU	1D	-	-	-	-	√	-	-	PC := Op if SR = > (vorzeichenlos)
BGEU	13	-	-	-	-	√	-	-	PC := Op if SR = ≥ (vorzeichenlos)
BLEU	1E	-	-	-	-	√	-	-	PC := Op if SR = ≤ (vorzeichenlos)
BLTU	14	-	-	-	-	√	-	-	PC := Op if SR = < (vorzeichenlos)
JSR	1F	-	-	-	-	√	-	-	PC := Op, RAM[SP–1] := PC, SP := SP–1
LDA	20..23	x	x	0	0	√	√	√	AC := Op
LDX	28..2B	x	x	0	0	√	√	√	IX := Op
RDS	24..27	x	x	0	0	-	√	-	AC := RAM[SP+Op]
STA	30..33	-	-	-	-	√	-	√	Op := AC
STX	38..3B	-	-	-	-	√	√	√	Op := IX
WRS	34..37	-	-	-	-	-	√	-	RAM[SP+Op] := AC
ADD	40..43	x	x	x	x	√	√	√	AC := AC + Op
ADDX	48..4B	x	x	x	x	√	√	√	IX := IX + Op
SUB	50..53	x	x	x	x	√	√	√	AC := AC – Op
SUBX	58..5B	x	x	x	x	√	√	√	IX := IX – Op
CMP	60..63	x	x	x	x	√	-	-	AC – Op
CMPX	68..6B	x	x	x	x	√	√	√	IX – Op
AND	70..73	x	x	0	0	√	√	√	AC := AC ∧ Op (bitpaarweise)
OR	80..83	x	x	0	0	√	√	√	AC := AC ∨ Op (bitpaarweise)
XOR	90..93	x	x	0	0	√	√	√	AC := AC ⊕ Op (bitpaarweise)

$x/-$: Statusbit wird verändert/ wird nicht verändert
√/−: Adressierungsart verfügbar/ nicht verfügbar
Op: Operand gemäß Adressierungsart