Klausur Informatik 2
3. März 2003

Jähnichen / Flik
Lechtchinsky / Neunast

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 120 Minuten

- Als Hilfsmittel sind nur ein nicht programmierbarer Taschenrechner und ein doppelseitig, **in eigener Handschrift** beschriebenes DIN-A4-Blatt zugelassen.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	12	
2	11	
3	10	
4	6	
5	5	
6	6	

1. Aufgabe (12 Punkte): Algorithmen und Datenstrukturen

In dieser Aufgabe sollen doppelt verkettete Listen von ganzen Zahlen als eine Java-Klasse realisiert werden. Dabei soll die folgende Definition wie unten angegeben erweitert werden:

```
class IntList {
  class Elem {
    int value;
    Elem prev;
    Elem next;
  }

  Elem first;
  Elem last;
}
```

1.1. Aufbau der Klasse Elem (1 Punkt) Bei welchen Instanzvariablen der Klasse Elem handelt es sich um Referenzen? Begründen Sie Ihre Antwort.

1.2. Leere Listen (1 Punkt) Wie läßt sich eine leere Liste als eine Instanz der Klasse IntList darstellen? Implementieren Sie eine Methode clear der Klasse IntList, die alle Elemente aus einer Liste entfernt.

1.3. Operationen auf Listen (4 Punkte) Implementieren Sie die folgenden Operationen als Methoden der Klasse IntList:

- append fügt eine gegebene ganze Zahl am Ende der Liste ein;
- removeLast entfernt das letzte Element aus der Liste;
- length liefert die Länge der Liste.

1.4. Aufwand von Listenoperationen (1 Punkt) Welchen Aufwand haben die in der vorigen Unteraufgabe implementierten Methoden append, removeLast und length? Begründen Sie Ihre Antwort.

Hinweis: Hier genügt eine verbale Begründung, ein vollständiger Beweis ist nicht notwendig. Diese Aufgabe kann auch gelöst werden, wenn Sie für die Methoden keine Implementierung angegeben haben.

1.5. Entfernen von Elementen (5 Punkte) Implementieren Sie eine Methode remove, die alle Vorkommen einer gegebenen ganzen Zahl aus einer Liste entfernt.

2. Aufgabe (11 Punkte): Aufwandsabschätzung

Hinweis: Beachten Sie bei der Lösung dieser Aufgabe die Hinweise zur Aufwandsanalyse auf Seite 6.

2.1. $\mathcal{O}$-Kalkül (2 Punkte) Beweisen oder widerlegen Sie die folgende Behauptung (durch Argumentation, Beweis oder Gegenbeispiel): wenn f und g zwei Funktionen sind, dann gilt mindestens eine der folgenden Aussagen: $f \in \mathcal{O}(g)$ oder $g \in \mathcal{O}(f)$.

Hinweis: Beachten Sie, daß nicht alle Funktionen monoton sind.

2.2. Die Klassen $\mathcal{O}$, Ω und Θ (3 Punkte) Bestimmen Sie für die im folgenden angegebenen Funktionen, ob f Element der Klassen $\mathcal{O}(g)$, $\Omega(g)$ und $\Theta(g)$ ist. Geben Sie für *jede* dieser Klassen eine Antwort mit Begründung.

$$\begin{aligned} f(n) &= 3n^{\frac{10}{3}} + 5n - 3 \\ g(n) &= n^3 \end{aligned}$$

2.3. Aufwand von Methoden (5 Punkte) Die im folgenden definierten Methoden berechnen für ein gegebenes n den Wert 2^n einmal iterativ (pow2It) und einmal rekursiv (pow2Rec). Leiten Sie die Aufwandsfunktionen für die beiden Methoden her, bestimmen sie die kleinsten Aufwandsklassen, zu denen die Funktionen gehören, und vergleichen Sie diese. Die Herleitungen müssen nachvollziehbar sein (Zeilenbezug, verwendete Regeln usw.). Falls Sie Rekurrenzgleichungen verwenden, geben Sie die Nummer der Gleichung sowie die Werte der Konstanten c, b und k an (vgl. Tabelle auf Seite 6).

```
1  public static int pow2It( int n ) {
2    int x = 1;
3    for( int i = 0; i < n; i++ )
4      x = x + x;
5    return x;
6  }
7
8  public static int pow2Rec( int n ) {
9    if( n == 0 )
10     return 1;
11   else
12     return pow2Rec(n - 1) + pow2Rec(n - 1);
13 }
```

2.4. Aufwandsreduzierung (1 Punkt) Verändern Sie die Methode pow2Rec aus der vorigen Unteraufgabe so, daß sie einen Aufwand hat, der unter $\mathcal{O}(n^2)$ liegt. Die Methode soll dabei die Berechnung weiterhin rekursiv durchführen. Welchen Aufwand hat Ihre Version der Methode?

Hinweis: Hier genügt eine verbale Begründung, die Aufwandsfunktion muß nicht hergeleitet werden.

Hinweise zur Aufwandsanalyse

Bestimmung der Aufwandsfunktionen

Bei der Bestimmung der Aufwandsfunktion einer Methode werden folgende Operationen als jeweils ein Schritt gezählt:

- arithmetische, logische und Vergleichsoperatoren,
- Zuweisungen,
- Funktionsaufrufe,
- return-Anweisungen.

Die übrigen Anweisungen wie if und for werden nicht gezählt.

Rekurrenzrelationen

Die Aufwandsklasse einer Methode, deren Aufwand durch eine rekursive Funktion A gegeben ist, läßt sich aus der folgenden Tabelle ablesen:

	Rekurrenzrelation		$\mathcal{O}(A)$
(1)	$A(n) = A(n-1) + bn^k$		$\mathcal{O}(n^{k+1})$
(2)	$A(n) = cA(n-1) + bn^k$	mit $c > 1$	$\mathcal{O}(c^n)$
(3)	$A(n) = cA(n/d) + bn^k$	mit $c > d^k$	$\mathcal{O}(n^{\log_d c})$
(4)	$A(n) = cA(n/d) + bn^k$	mit $c < d^k$	$\mathcal{O}(n^k)$
(5)	$A(n) = cA(n/d) + bn^k$	mit $c = d^k$	$\mathcal{O}(n^k \log n)$

3. Aufgabe (10 Punkte): Programmverifikation

3.1. Zuweisungen (3 Punkte) Betrachten Sie das folgende Programmfragment (die Variable i ist vom Typ int):

```
1 i = 0;
2 i = i + 1;
```

Geben Sie die allgemeinste Vorbedingung an, bei der die Variable i nach der Ausführung dieses Fragments den Wert 1 erhält. Beweisen Sie ausgehend von der von Ihnen gewählten Vorbedingung, daß das Programmfragment die Nachbedingung i = 1 erfüllt. Geben Sie an, welche Regeln des Hoare-Kalküls Sie in dem Beweis verwendet haben.

3.2. Totale Korrektheit (1 Punkt) Was versteht man unter totaler Korrektheit eines Programms? Wie wird diese nachgewiesen?

3.3. Schleifen (6 Punkte) Betrachten Sie das folgende Programmfragment (die Variablen i, n und x sind vom Typ int):

```
1 i = 0;
2 while ( i != n ) {
3   x = x + i;
4   i = i + 1;
5 }
```

Nach der Ausführung des Fragments soll folgendes gelten: $x = \sum_{j=0}^{n-1} j$. Unter welcher Vorbedingung gilt dies? Beweisen Sie die totale Korrektheit des Programmfragments unter Annahme dieser Vorbedingung. Wie lautet bei Ihrem Beweis die Schleifeninvariante?

Hinweis: Die paritelle Korrektheit einer Schleife kann mit der folgenden Regel nachgewiesen werden:

$$\frac{\{I \wedge B\}\ \ P\ \ \{I\}}{\{I\}\ \ \mathsf{while}\,(B)\ \ P;\ \ \{I \wedge \neg B\}}$$

4. Aufgabe (6 Punkte): Assemblerprogrammierung

Hinweis: Beachten Sie bei der Bearbeitung dieser Aufgabe die VIP-Befehlstabelle auf Seite 11.

4.1. (3 Punkte) Gegeben ist die folgende Java-Methode, die feststellt, wie häufig die Zahl n im Array a vorkommt.

```
int count (int[] a , int n) {
  int occur = 0;
  int i = 0;
  while (i < a.length) {
    if (a[i] == n) occur++;
    i++;
  }
  return occur;
}
```

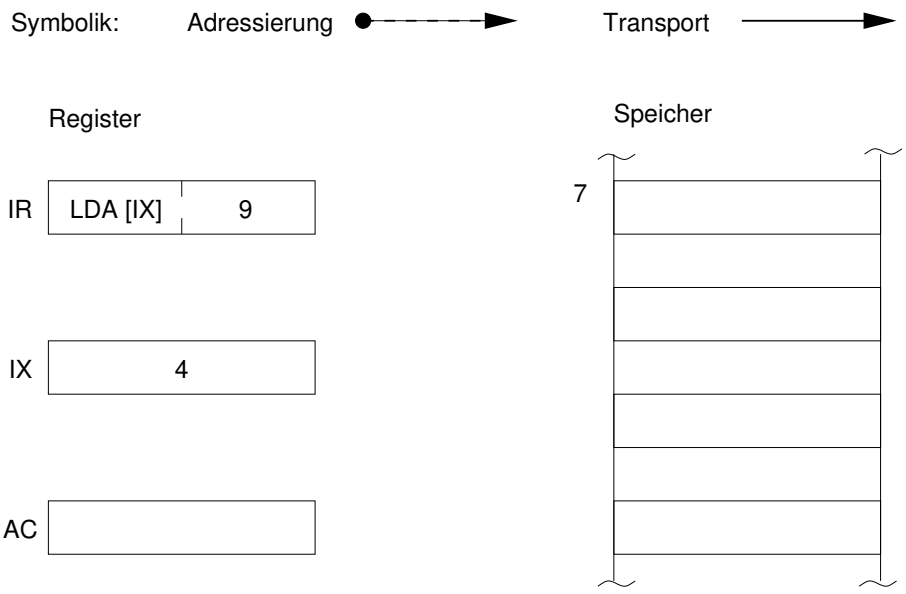
Schreiben Sie ein entsprechendes VIP-Assemblerprogramm, indem Sie die folgende Vorgabe vervollständigen. Kommentieren Sie das Programm.

```
            ORG 0
a           DAT 3, 4, 2, 3
n           DAT 3
occur       DAT 0
i           DAT 0
length      DAT 4
start       LDX i           ; i in IX laden
```

4.2. Assembleranweisungen (1 Punkt) Gegeben ist der folgende VIP-Assemblerprogrammausschnitt. Geben Sie die Werte im AC- und IX-Register nach Ausführung der Lade-Befehle an.

```
        ORG  1
A       RES  3
B       DAT  0
C       EQU  2
        LDA  #C
        LDX  #B
        :
```

4.3. Adressierungsarten (2 Punkte) Illustrieren Sie die Ausführung des im Instruktionsregister enthaltenen Befehls durch Zeichnen von Adressierungs- und Transportpfeilen im folgenden Bild:



VIP-Befehlstabelle

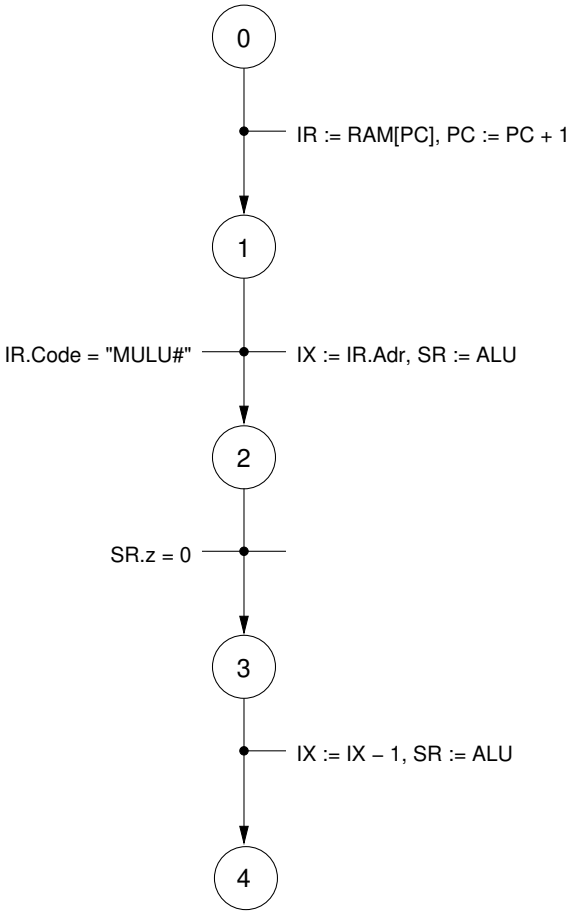
Befehl	Code	SR-Inhalt				Adressierungsarten				Wirkung
	(hex.)	z	n	v	c		#	@	[IX]	
HLT	00	-	-	-	-	-	-	-	-	Anhalten
NOP	01	-	-	-	-	-	-	-	-	Nichts
NOT	02	x	x	0	0	-	-	-	-	AC := ¬ AC (1-Komplement)
NEG	03	x	x	x	x	-	-	-	-	AC := 0 - AC (2-Komplement)
ASL	04	x	x	x	AC_{15}	-	-	-	-	AC := AC × 2
ASR	05	x	x	0	AC_0	-	-	-	-	AC := AC × 2^{-1}, AC_{15} := AC_{15}
LSL	06	x	x	0	AC_{15}	-	-	-	-	AC := AC × 2
LSR	07	x	x	0	AC_0	-	-	-	-	AC := AC × 2^{-1}, AC_{15} := 0
JMP	10	-	-	-	-	✓	-	-	-	PC := Op
BPL	11	-	-	-	-	✓	-	-	-	PC := Op, falls ≥ 0
BMI	12	-	-	-	-	✓	-	-	-	PC := Op, falls < 0
BCC	13	-	-	-	-	✓	-	-	-	PC := Op, falls kein Übertrag
BCS	14	-	-	-	-	✓	-	-	-	PC := Op, falls Übertrag
BVC	15	-	-	-	-	✓	-	-	-	PC := Op, falls kein Überlauf
BVS	16	-	-	-	-	✓	-	-	-	PC := Op, falls Überlauf
BEQ	17	-	-	-	-	✓	-	-	-	PC := Op, falls = (Null)
BNE	18	-	-	-	-	✓	-	-	-	PC := Op, falls ≠ (Null)
BGT	19	-	-	-	-	✓	-	-	-	PC := Op, falls >
BGE	1A	-	-	-	-	✓	-	-	-	PC := Op, falls ≥
BLE	1B	-	-	-	-	✓	-	-	-	PC := Op, falls ≤
BLT	1C	-	-	-	-	✓	-	-	-	PC := Op, falls <
BGTU	1D	-	-	-	-	✓	-	-	-	PC := Op, falls > (vorzeichenlos)
BGEU	13	-	-	-	-	✓	-	-	-	PC := Op, falls ≥ (vorzeichenlos)
BLEU	1E	-	-	-	-	✓	-	-	-	PC := Op, falls ≤ (vorzeichenlos)
BLTU	14	-	-	-	-	✓	-	-	-	PC := Op, falls < (vorzeichenlos)
LDA	20..23	x	x	0	0	✓	✓	✓	✓	AC := Op
LDX	28..2B	x	x	0	0	✓	✓	✓	✓	IX := Op
STA	30..33	-	-	-	-	✓	-	✓	✓	Op := AC
STX	38..3B	-	-	-	-	✓	-	✓	✓	Op := IX
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)

5. Aufgabe (5 Punkte): VIP

Gegeben ist das nachstehende, aus der Vorlesung bekannte VIP-Strukturbild.

5.1. Zustandsgraph (3 Punkte) Der VIP soll um einen Befehl "`MULU#`" mit der Codierung `0x1F` erweitert werden, der den Inhalt des Akkumulators mit einem Direktoperanden multipliziert. Beide Operanden werden dabei als vorzeichenlos interpretiert. (Es kann davon ausgegangen werden, daß der Direktoperand auf dem Weg zur ALU durch führende Nullen erweitert wird.) Die Multiplikation wird durch wiederholte Addition nachgebildet; das IX-Register wird dazu als Schleifenzähler verwendet, d.h. der vor der Befehlsausführung enthaltene Wert wird überschrieben.

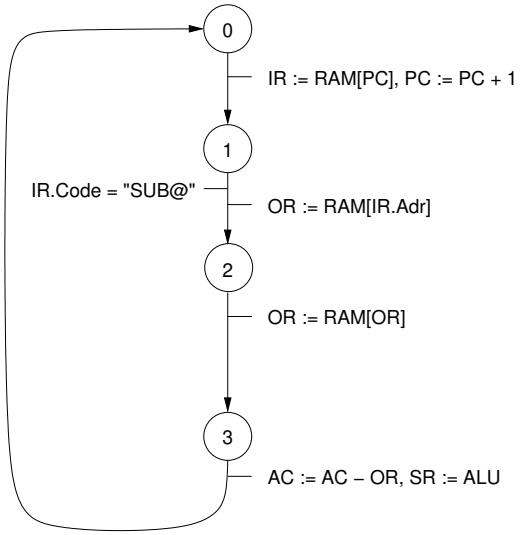
Vervollständigen Sie den Zustandsgraphen für den "`MULU#`"-Befehl durch Hinzufügen fehlender Zustandsübergänge und Operationen.



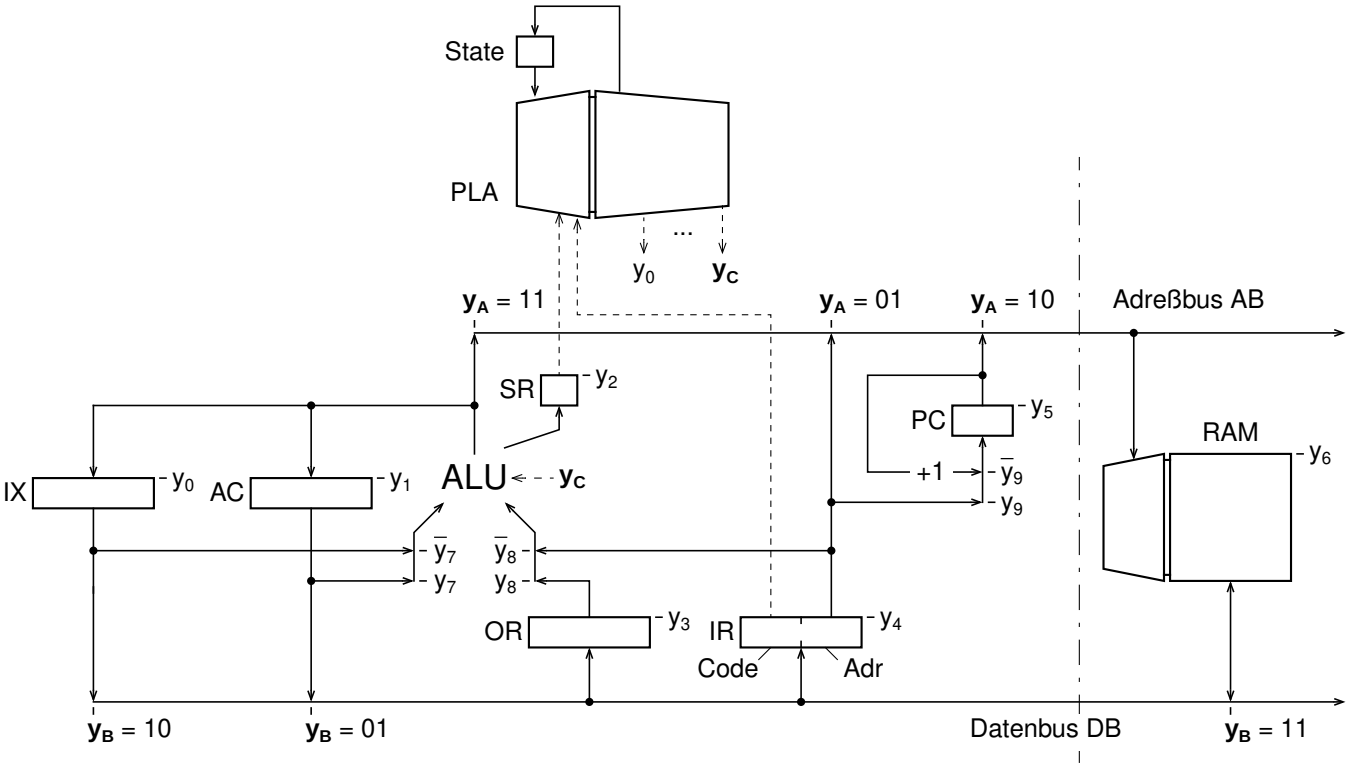
5.2. PLA-Tabelle (2 Punkte) Füllen Sie die nachstehende PLA-Tabelle gemäß dem gegebenen Zustandsgraphen des Befehls "**S**UB @" mit der Codierung 0x52 aus. Irrelevante Tabellenplätze dürfen freigelassen werden.

Hinweis: Ausschnitt aus der ALU-Steuertabelle (X: linker Operand, Y: rechter Operand, Z: Resultat)

$s_{0\ldots4}$	ALU-Operation
0 0 0 0 1	$Z = 0$
0 0 1 1 0	$Z = X + u_0$
1 0 0 1 0	$Z = X + Y + u_0$
1 0 1 0 1	$Z = Y$
1 1 0 0 0	$Z = X - 1 + u_0$
1 1 1 1 0	$Z = X \cdot 2 + u_0$
0 1 1 0 0	$Z = X - Y - 1 + u_0$

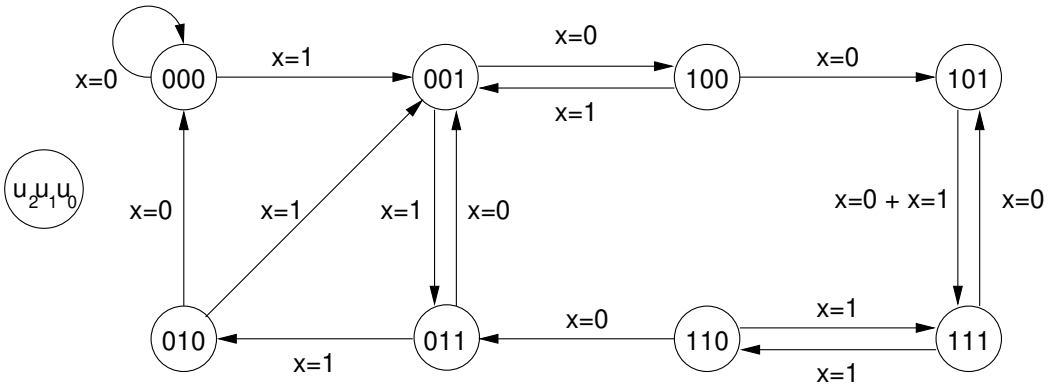


	SR				IR		Register/Speicher							Multiplexer					ALU
Z_i	z	n	c	v	Code	Z_j	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_A	y_B	$y_C = s_{0\ldots4}, u_0$



6. Aufgabe (6 Punkte): Schaltwerksentwurf

Gegeben ist der abgebildete Zustandsgraph eines Schaltwerks mit den drei Zustandsvariablen u_2, u_1, u_0 und dem Eingangssignal x,



6.1. (1 Punkt) Ergänzen Sie die nachstehende Tabelle um die Funktionswerte für die Übergangsfunktion u_2.

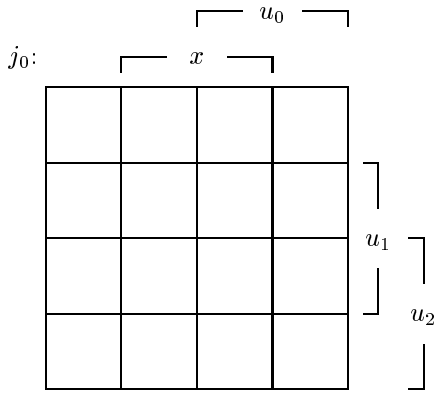
6.2. (2 Punkte) Ergänzen Sie die nachstehende Tabelle um die Ansteuerfunktionen s_1 und r_1 eines SR-Flipflops für u_1.

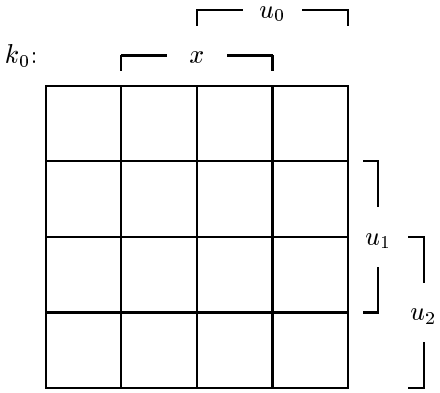
6.3. (2 Punkte) Ermitteln mit Hilfe der nachstehenden KV-Tafeln minimale disjunktive Normalformen für die Ansteuerfunktionen j_0 und k_0 eines JK-Flipflops für u_0.

Hinweis: Flipflop-Übergangstabelle

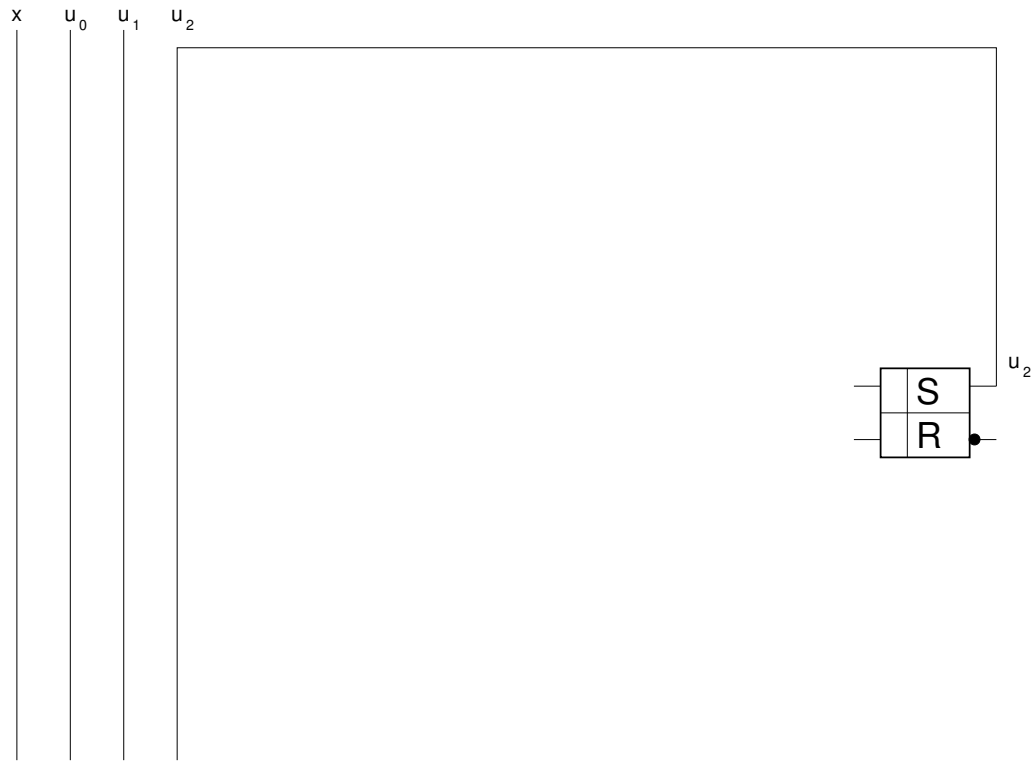
$v \to v^+$	d	s	r	j	k
$0 \to 0$	0	0	*	0	*
$0 \to 1$	1	1	0	1	*
$1 \to 0$	0	0	1	*	1
$1 \to 1$	1	*	0	*	0

u_2	u_1	u_0	x	u_2^+	u_1^+	u_0^+	s_1	r_1	j_0	k_0
0	0	0	0		0	0			0	*
0	0	0	1		0	1			1	*
0	0	1	0		0	0			*	1
0	0	1	1		1	1			*	0
0	1	0	0		0	0			0	*
0	1	0	1		0	1			1	*
0	1	1	0		0	1			*	0
0	1	1	1		1	0			*	1
1	0	0	0		0	1			1	*
1	0	0	1		0	1			1	*
1	0	1	0		1	1			*	0
1	0	1	1		1	1			*	0
1	1	0	0		1	1			1	*
1	1	0	1		1	1			1	*
1	1	1	0		0	1			*	0
1	1	1	1		1	0			*	1





6.4. (1 Punkt) Ergänzen Sie im folgenden Blockschaltbild die Eingangsbeschaltung für das Flipflop u_2 entsprechend den gegebenen Gleichungen:

$$\begin{aligned} s_2 &= u_0 \cdot \overline{u_1} \cdot \overline{x} + u_0 \cdot u_1 \cdot x + \overline{u_1} \cdot u_2 \cdot \overline{x} \\ r_2 &= \overline{u_0} \cdot u_1 \cdot \overline{x} + \overline{u_0} \cdot \overline{u_1} \cdot x \end{aligned}$$

Hinweis: Sie können Gatter mit bis zu drei Eingängen verwenden. Die Eingänge können direkt am Gatter negiert werden.