berlin

Fakultät IV

Klausur Informatik 2
02. März 2004

Jähnichen / Flik
Seifert / Neunast

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 120 Minuten

- Als Hilfsmittel sind nur ein nicht programmierbarer Taschenrechner und ein doppelseitig und **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! Streichen Sie gegebenenfalls eine Lösung durch!
- 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.
- Sollten Sie im Softwareteil eine Teilaufgabe nicht lösen können, so dürfen Sie die dort geforderte Funktion in anderen Teilaufgaben verwenden.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	5	
2	9	
3	11	
4	8	
5	6	
6	5	
7	6	

1. Aufgabe (5 Punkte): Java-Programmierung

Gegeben ist folgendes Java-Programm:

```
class B {
   public static final int MAX = 1024;
   public int f () { return 3; }
}

class A extends B {
   static public int n = 0;
   public int i = 0;
   public static void main( String args [] ) {
      // ... siehe unten
   }
}
```

Nehmen Sie an, dass die Methode main der Klasse A jeweils ausschließlich aus dem Code besteht, der in jeder der folgenden Teilaufgaben steht. Kreuzen Sie die jeweils richtige Antwort an!

1. Sind die Zuweisungen an a.i und A.n korrekt (d.h. können kompiliert und ausgeführt werden)? ☐ beide korrekt ☐ nur a.i korrekt ☐ nur A.n korrekt

```
A a = new A();
a.i = A.n;
A.n = A.n + a.i;
```

2. Welchen Wert hat r nach der Zuweisung? ☐ 2 ☐ 3 ☐ 4

```
double r;
A a1 = new A(), a2 = new A();
a1.i = ++A.n;
a2.i = A.n++;
r    = A.n + a1.i + a2.i;
```

3. Ist die Zuweisung an b korrekt? ☐ korrekt ☐ Compilefehler ☐ Laufzeitfehler

```
A a = new A();
B b = a;
```

4. Welcher der drei Aufrufe von f ist nicht korrekt? ☐ der erste ☐ der zweite ☐ der dritte

```
int j;
B b = new B();
j   = f ();
j   = b.f ();
j   = B.f ();
```

5. In welcher Zeile tritt der Fehler auf? ☐ kein Fehler ☐ zweite Zeile ☐ dritte Zeile

```
1 int j;
2 j = MAX;
3 MAX = MAX + 1;
```

Hinweis: Jedes richtige Kreuz gibt einen Punkt, für jedes falsche Kreuz wird ein halber Punkt abgezogen. Weniger als insgesamt 0 Punkte gibt es in dieser Aufgabe nicht.

2. Aufgabe (9 Punkte): Algorithmen und Datenstrukturen

In dieser Aufgabe geht es um die Implementierung einer Java-Klasse SparseVec für eine platzsparende Repräsentation von mathematischen Vektoren. Eine solche Repräsentation bietet sich an, wenn mit sehr langen Vektoren gearbeitet werden soll, bei denen die meisten Elemente jedoch den Wert 0 haben.

In einer Instanz der Klasse SparseVec werden nur die von 0 verschiedenen Werte gespeichert, und zwar in Tupeln der Form (Zeile, Wert), repräsentiert durch Instanzen der Java-Klasse Element.

Folgende Definition von SparseVec und Element sind gegeben:

```
public class SparseVec {

    private class Element {
        int     row;
        double  value;
        Element next;
    }

    private Element first ;
    private int     dim;

    // 'd' is the dimension of the vector
    public SparseVec( int d ) {
        dim   = d;
        first = null ;
    }
}
```

2.1. Implementierung von setValue (4 Punkte) Deklarieren und definieren Sie eine Methode setValue, die als Argumente die Zeile und den Wert erhält, der in der entsprechenden Zeile des Vektors zu speichern ist.

Existiert bereits eine Instanz von Element, die einen Wert für diese Zeile speichert, so soll dieses Objekt wiederverwendet und nicht neu angelegt werden. Außerdem ist beim Einfügen von Elementen in die Liste darauf zu achten, dass die Liste hinsichtlich der Zeilennummer aufsteigend sortiert sein soll. (Dies ist wichtig für die Implementation der Methode sproduct .)

2.2. Implementierung von getValue (2 Punkte) Deklarieren und definieren Sie eine Methode getValue, die als Argument die Zeile erhält, deren Wert als Resultat geliefert werden soll.

2.3. Skalarprodukt (3 Punkte) Deklarieren und definieren Sie eine Klassenmethode sproduct, die als Argumente zwei Vektoren erhält und als Ergebnis das Skalarprodukt der beiden Vektoren liefert. Das Skalarprodukt von zwei Vektoren $\mathbf{x}$ und $\mathbf{y}$ ist definiert als $<\mathbf{x},\mathbf{y}>=\sum_{i=1}^{N} x_i y_i$, wobei N die Dimension des Vektors ist.

Machen Sie hier ausdrücklich davon Gebrauch, dass (i) in den Vektoren nur die von 0 verschiedenen Elemente abgespeichert werden und (ii) die Tupel in der Liste in aufsteigender Reihenfolge sortiert sind.

3. Aufgabe (11 Punkte): Aufwandsabschätzung

3.1. Die Klassen $\mathcal{O}(g)$, $\Omega(g)$, und $\Theta(g)$ (4 Punkte) Prüfen Sie mittels der Definition des $\mathcal{O}$-Kalküls oder einer Grenzwertbetrachtung für die gegebenen Funktionen f und g, ob f Element von $\mathcal{O}(g)$, $\Omega(g)$ und/oder $\Theta(g)$ ist.

1. $f(x) = 3 \ln x^4 + x^2$
   $g(x) = x^2 + 2x^3$

2. $f(x) = \frac{4x^5+5x^2}{3x^2}$
   $g(x) = \frac{7x^3+5x^2+6}{4x}$

3.2. Aufwandsklasse von max (5 Punkte) Gegeben ist die folgende Methode zum Suchen eines Maximums innerhalb eines zweidimensionalen Arrays. Das übergebene Array ist quadratisch, d.h. es besitzt n Zeilen und n Spalten.

Geben Sie die Aufwandsfunktion für den *Worst-Case*-Aufwand der Methode an, und bestimmen Sie die kleinste Aufwandsklasse, zu der diese gehört. Die Rechnung muss nachvollziehbar, und der Bezug auf Code-Zeilen und verwendete Regeln klar erkennbar sein.

```
static public int max(int [][] a) {
    // PRE: ∀i ∈ [0, a.length − 1] : a[i].length = a.length
    int max = Integer.MIN_VALUE;
    for ( int i = 0; i < a.length; i++) {
        for ( int j=0; j<a[i].length; j++) {
            if (a[i][j] > max) {
                max = a[i][j];
            }
        }
    }
    return max;
}
```

3.3. Array für den Worst Case (1 Punkt) Welche Eigenschaften muss das Array a erfüllen, damit der *Worst Case* für max eintritt?

3.4. Einschätzung von Algorithmen (1 Punkt) Nehmen Sie an, zur Lösung eines Problems existieren zwei Algorithmen. In Abhängigkeit der Aufgabengröße n sei die Aufwandsklasse des ersten Algorithmus $A_1(n) \in \mathcal{O}(n^2)$, und die des zweiten sei $A_2(n) \in \mathcal{O}(n \cdot \log n)$ (beispielsweise InsertionSort und HeapSort, wenn es sich um ein Sortierproblem handeln würde). Schildern Sie kurz mindestens einen möglichen Grund oder Umstand dafür, dass man sich für den anscheinend nachteilhaften Algorithmus mit quadratischem Aufwand entscheiden würde.

4. Aufgabe (8 Punkte): Programmverifikation

4.1. Partielle Korrektheit (6 Punkte) Beweisen Sie für das folgende Programmfragment die partielle Korrektheit bezüglich der gegebenen POST-Bedingung und geben Sie die dafür benötigte Vorbedingung an. Benutzen Sie den Vordruck auf der nächsten Seite und kennzeichnen Sie Ihre Invariante an allen auftretenden Stellen deutlich.

```
int sum (int a []) {
    // PRE: a.length ≡ n ∧ ...
    int sum, i;
    sum = 0;
    i = a.length−1;
    while (i >= 0) {
        sum = sum + a[i];
        i = i − 1;
    }
    // sum = Σ_{k=0}^{n−1} a_k ∧ i = −1
    // POST: sum = Σ_{k=0}^{n−1} a_k
    return sum;
}
```

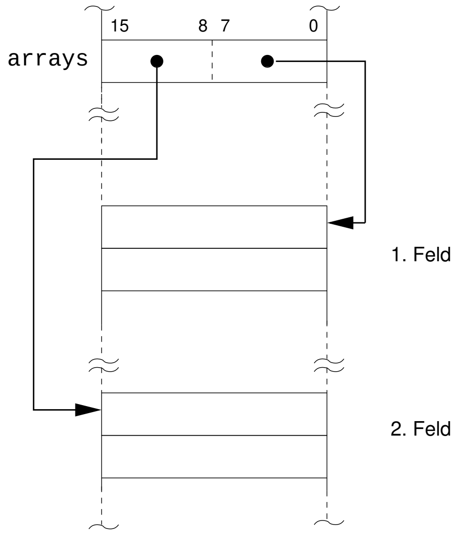
4.2. Korrektheit, Spezifikation, Testen (2 Punkte) Definieren Sie kurz die drei Begriffe (Korrektheit, Spezifikation, Testen einer Methode), wobei Sie insbesondere auf die Zusammenhänge der drei Begriffe untereinander eingehen.

```
int sum (int a []) {
    // PRE: a.length ≡ n∧

    int sum, i;

    sum = 0;

    i = a.length−1;

    while (i >= 0) {

        sum = sum + a[i];

        i = i − 1;

    }

    // sum = Σ_{k=0}^{n−1} a_k ∧ i = −1
    // POST: sum = Σ_{k=0}^{n−1} a_k

    return sum;
}
```

5. Aufgabe (6 Punkte): Assemblerprogrammierung

Hinweis: Die Lösung dieser Aufgabe erfolgt in drei Teilaufgaben, bei denen jeweils nur die zur Lösung der Aufgabe erforderlichen Zeilen Assemblercode anzugeben sind.

Es soll ein VIP-Assemblerprogramm geschrieben werden, das zwei Felder elementweise addiert. Die Adressen der beiden Felder sind im Hauptspeicher als ein 16-Bit-Speicherwort mit dem Label `arrays` abgelegt. Die Adresse des ersten Feldes belegt dabei die unteren 8 Bits, die Adresse des zweiten Feldes belegt die oberen 8 Bits, wie die folgende Darstellung zeigt:



Die beiden Felder haben die gleiche Länge, die in der Speicherzelle mit der symbolischen Adresse `length` abgelegt ist. Das Ergebnisfeld soll an der symbolischen Adresse `result` beginnen.

Verwenden Sie folgende Vorgaben:

```
            ORG 0
arrays      DAT 0xF2EE
length      DAT 4
ptr1        RES 1
ptr2        RES 1
result      RES 4
```

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

5.1. (2 Punkte) Geben Sie ein Programmfragment an, das die Adresse des ersten Feldes ausliest und diese an der Adresse mit dem Label `ptr1` speichert.

5.2. (2 Punkte) Geben Sie ein Programmfragment an, das die Adresse des zweiten Feldes ausliest und diese an der Adresse mit dem Label `ptr2` speichert.

5.3. (2 Punkte) Gegeben ist eine unvollständige Implementation einer Schleife, in der die beiden Felder elementweise addiert werden sollen. Vervollständigen Sie die Vorgabe. Zur Adressierung des Ergebnisfeldes `result` ist das Indexregister zu verwenden.

```
            LDX #0
loop




            LDA ptr1
            ADD #1
            STA ptr1
            LDA ptr2
            ADD #1
            STA ptr2
            ADDX #1
            CMPX length
            BNE loop
            HLT
            END Start
```

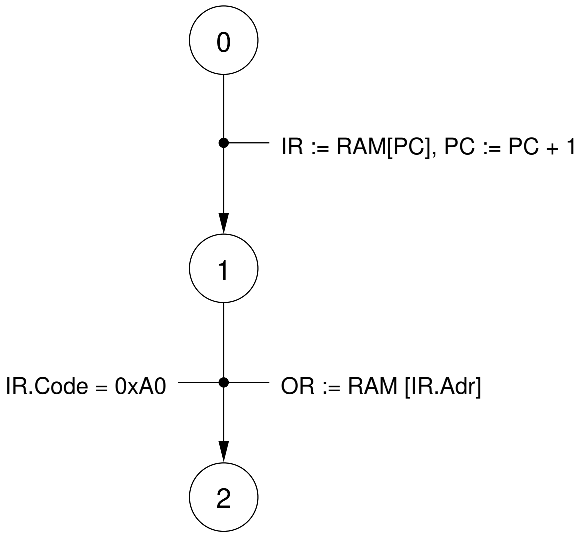
VIP-Befehlstabelle

Befehl	Code (hex.)	SR-Inhalt z	n	v	c	Adressierungsarten	#	@	[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	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)

6. Aufgabe (5 Punkte): VIP

Hinweis: Gegeben ist das aus der Vorlesung bekannte VIP-Strukturbild auf Seite 13.

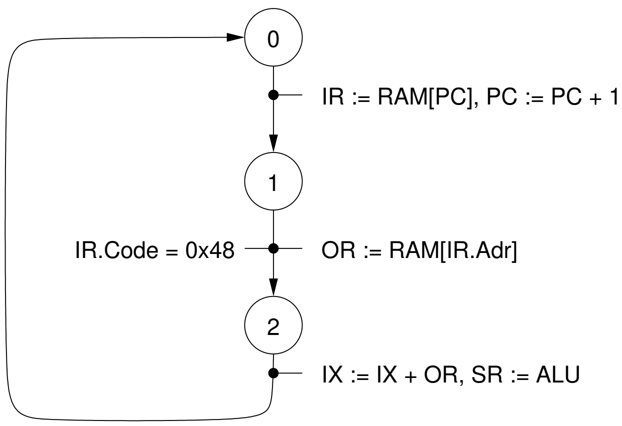
6.1. Zustandsgraph (3 Punkte) Der VIP soll um einen Befehl "`MODU`" mit der Codierung `0xA0` und der speicherdirekten Adressierung erweitert werden, der den ganzzahligen Rest einer Ganzzahl-Division des Akkumulatorinhaltes mit dem adressierten Operanden bildet. Beide Zahlen werden als vorzeichenlos interpretiert. Bei einer Division durch 0 wird eine 0 als Ergebnis geliefert und das z-Bit wird dementsprechend auf 1 gesetzt. Die Division ist mittels wiederholter Subtraktion nachzubilden.
Vervollständigen Sie den Zustandsgraphen für den "`MODU`"-Befehl durch Hinzufügen fehlender Zustände, Zustandsübergänge (ggf. mit Übergangsbedingungen) und Register-Transfer-Operationen.



6.2. PLA-Tabelle (2 Punkte) Füllen Sie die nachstehende PLA-Tabelle gemäß dem gegebenen Zustandsgraphen des Befehls “`ADDX`” mit der Codierung `0x48` aus. Irrelevante Tabellenplätze dürfen freigelassen werden.

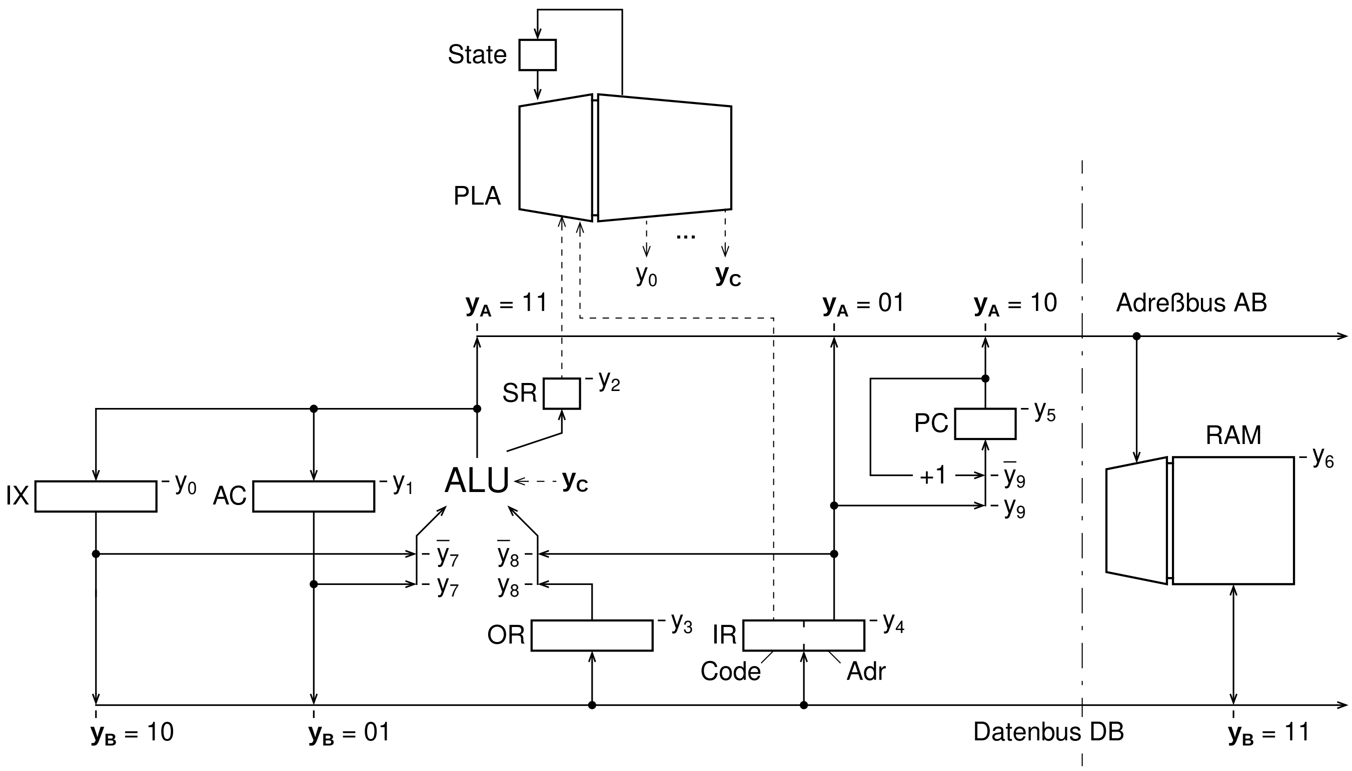
Hinweis: Ausschnitt aus der ALU-Steuertabelle (X: linker Operand, Y: rechter Operand, Z: Resultat)

$s_{0...4}$	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	$Z = X - Y - 1 + u_0$



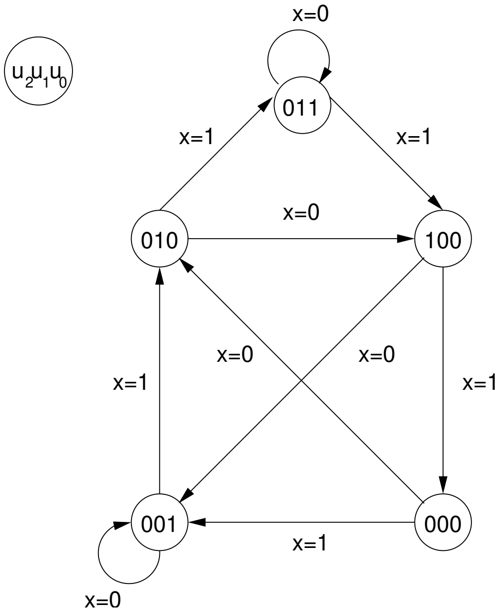
Z_i	SR				IR	Z_j	Register/Speicher							Multiplexer					ALU
	z	n	c	v	Code		y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_A	y_B	$y_C = s_{0...4}, u_0$

VIP-Strukturbild



7. 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. Das Schaltwerk soll mit zwei SR-Flipflops und einem JK-Flipflop realisiert werden.



7.1. (1 Punkt) Ergänzen Sie die nachstehende Tabelle um die Funktionswerte für die Übergangsfunktion u_2.

7.2. (2 Punkte) Ergänzen Sie die nachstehende Tabelle um die Ansteuerfunktionen s_1 und r_1 eines SR-Flipflops für u_1.

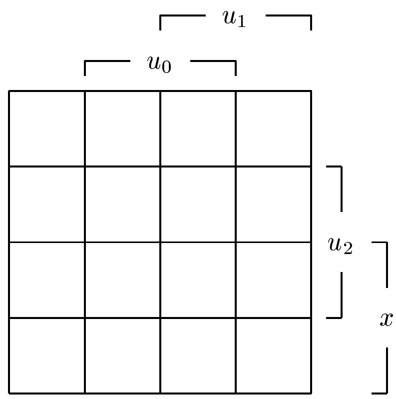
7.3. (2 Punkte) Ermitteln Sie mit Hilfe der nachstehenden KV-Tafeln je eine minimale DNF für die Ansteuerfunktionen j_0 und k_0 eines JK-Flipflops für u_0.

Hinweis: Flipflop-Übergangstabelle

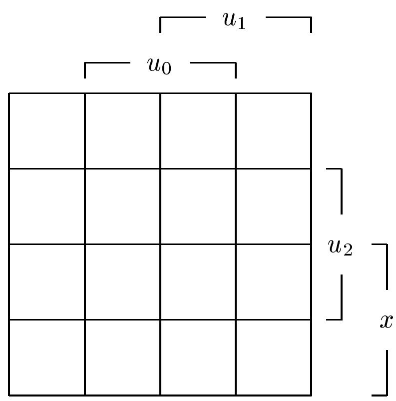
$v \rightarrow v^+$	d	s	r	j	k
$0 \rightarrow 0$	0	0	*	0	*
$0 \rightarrow 1$	1	1	0	1	*
$1 \rightarrow 0$	0	0	1	*	1
$1 \rightarrow 1$	1	*	0	*	0

u_2	u_1	u_0	x	u_2^+	u_1^+	u_0^+	s_2	r_2	s_1	r_1	j_0	k_0
0	0	0	0		1						0	*
0	0	0	1		0						1	*
0	0	1	0		0						*	0
0	0	1	1		1						*	1
0	1	0	0		0						0	*
0	1	0	1		1						1	*
0	1	1	0		1						*	0
0	1	1	1		0						*	1
1	0	0	0		0						1	*
1	0	0	1		0						0	*

j_0:



k_0:



7.4. (1 Punkt) Ergänzen Sie im folgenden Blockschaltbild die Eingangsbeschaltung für das Flipflop u_2 entsprechend den gegebenen Gleichungen:

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

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