Fakultät IV

Klausur Informatik 2
25. Juli 2002

Gorlatch / Flik
Jucknath / Neunast

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 120 Minuten

- 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!
- Die Java-Implementierungen müssen nicht dokumentiert/kommentiert 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	9	
3	12	
4	5	
5	4	
6	8	

1. Aufgabe (12 Punkte): Algorithmen und Datenstrukturen

1.1. Klassen und Instanzen (1 Punkte) Geben Sie im vorliegende Code-Fragment für jeden Bezeichner an, wo es sich jeweils um eine Klasse und wo es sich um die Instanz einer Klasse handelt.

```
public class Test
{
  public int value = 1;

  public bool testValue(Test t)
  {
     if(value == t.value)   return true;
     else                   return false;
  }
}
```

1.2. Call by Value und Call by Reference (1 Punkte) Geben Sie im vorliegende Code-Fragment für jeden Variablen-Bezeichner an, ob es sich um einen Referenz-Datentypen oder um einen primitiven Datentypen handelt.

```
public class Test
{
  public int value = 1;

  String result1 = "equal";
  String result2 = "not ";

  public bool testValue(Test t)
  {
     if(value == t.value)
     {
   System.out.println(result1);

   return true;
     }else{
   System.out.println(result2 + result 1);

   return false;
     }
  }
}
```

1.3. Telefonische Bestellung bei einem Pizza-Service (3 Punkte) Implementieren Sie eine Klasse `Bestellungen` , die alle Kundenanrufe derart verwaltet, daß der Kunde, der als erster anruft, auch als erster bedient wird. Benutzen Sie dazu die Klasse `Kundenanruf` und betrachten Sie die Klasse `Pizza` als gegeben.

```
public class Kundenanruf
{
   String Kundenname;
   int Telefonnummer;
   Pizza[] Kundenwunsch;
}
```

Implementieren Sie zu der Klasse `Bestellungen` zwei Methoden: `insert` (nimmt einen neuen Kundenanruf in die Bestellung mit auf und gibt die neue Bestellung zurück) und `delete` (nimmt den als nächsten zu bearbeitenden Kundenanruf aus der Bestellung heraus und gibt diesen Kundenanruf zurück).

1.4. VIP-Kunden (1 Punkte) Erweitern Sie die Klasse `Bestellung` derart, daß auch VIP-Kunden verwaltet werden können. Wenn ein VIP-Kunde anruft, soll er immer vor allen anderen Kunden behandelt werden, selbst wenn diese auch VIP-Kunden sind.

1.5. Vorbestellungen (3 Punkte) Implementieren Sie eine neue Klasse `Vorbestellung` , die alle Funktionalitäten von `Bestellung` hat, aber zusätzlich noch über einen Integerwert `Uhrzeit` verfügt, der angibt, um wieviel Minuten nach Geschäftsbeginn eine Bestellung ausgeführt werde soll. Sie müssen nicht alle Methoden neu schreiben, sondern können diese aus der Klasse `Bestellung` mit übernehmen, wenn Sie die Klasse `Vorbestellung` mit dem Schlüsselwort `extends` von der Klasse `Bestellung` ableiten.
Implementieren Sie für die Klasse `Vorbestellung` eine Methode `gesamtBestellung` , die eine `Vorbestellung` a und eine `Vorbestellung` b übergeben bekommt und `Vorbestellung` zurückgibt, die alle Kundenanrufe aus a und enthält, in Reihenfolge der Uhrzeit.

1.6. Suche innerhalb der Vorbestellung (3 Punkte) Implementieren Sie eine Methode `Kundensuche` , die einen Integerwert `Telefon` übergeben bekommt und diesen innerhalb einer `Vorbestellung` sucht. Zurückgegeben werden soll die Position des gefunden Kunden, gerechnet vom Kunden, der als nächstes bedient werden soll. Wenn die Telefonnummer nicht in der gesamten Vorbestellung enthalten sein sollte, oder die Vorbestellung leer ist, soll der Wert größer als 1024 zurückgegeben werden.

1.7. Pizza-Planer (3 Punkte) Implementieren Sie eine Methode `pizzaListe` , die als Paramter einen Interwert n übergeben bekommt und ein Array von `Pizza` zurückgibt, welches die nächsten n `Pizza` aus der `Bestellung` der am nächsten zu bearbeitenden `Kundenanrufe` zurückgibt. Falls die `Bestellung` weniger als n `Pizza` insgesamt enthält, soll ein leeres Array zurückgegeben werden.
Implementieren Sie eine Methode `kundenListe` , welche als Paramter ein Array von n `Pizza` bekommt zurückgibt

2. Aufgabe (9 Punkte): Aufwandsabschätzung

2.1. Die Klassen $\mathcal{O}, \Omega, \Theta$ (4 Punkte) Bestimmen Sie für die im folgenden angegebenen Funktionen, ob f jeweils Element der Klassen $\mathcal{O}(g)$, $\Omega(g)$, und $\Theta(g)$ ist:

- $f(n) = 5n^{\frac{3}{2}}$
  $g(n) = n\sqrt{n}$
- $f(n) = 2n^{\frac{1}{2}} - 4$
  $g(n) = \ln n$

2.2. Aufwand von foobar (5 Punkte) Bestimmen Sie die Aufwandsfunktion der Methode foobar für den „Worst Case" und leiten Sie daraus die Aufwandsklasse der Methode ab. Vergessen Sie nicht, die Herleitung nachvollziehbar darzustellen (Zeilenbezug, verwendete Regeln, …).

```
public class fooboobar
{
Z 01:     public static int foobar(int[] values, int foo,
Z 02:       int AreaStart, int AreaEnd)
Z 03:     {
Z 04:        if(AreaStart == AreaEnd &&
Z 05:           values[AreaStart] != foo) return -1;
Z 06:
Z 07:        int middleOfArea = (AreaStart + AreaEnd) / 2;
Z 08:
Z 09:        if(values[middleOfArea] < foo)
Z 10:            return foobar(values, foo, middleOfArea + 1, AreaEnd);
Z 11:        if(values[middleOfArea] > foo)
Z 12:            return foobar(values, foo, AreaStart, middleOfArea);
Z 13:
Z 14:        return middleOfSearchArea;
Z 15:    }

    public static int boobar(int[] values, int foo)
    {
        if(values == null || values.length == 0) return -1;
        return foobar(values, foo, 0, values.length-1);
    }
}
```

Die Aufwandsklasse einer Funktion, deren Aufwandsfunktion durch eine rekursive Definition gegeben ist, kann aus folgender Tabelle abgelesen werden:

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

Rechenregeln ($\mathcal{O}$-Kalkül):

(1)	$f \in \mathcal{O}(f)$	
(2)	$k * \mathcal{O}(f) = \mathcal{O}(f),$	k konstant
(3)	$\mathcal{O}(\mathcal{O}(f)) = \mathcal{O}(f)$	
(4)	$\mathcal{O}(f) * \mathcal{O}(g) = \mathcal{O}(f * g)$	
(5)	$\mathcal{O}(f * g) = f * \mathcal{O}(g)$	
(6)	$\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f + g)$	
(7)	$f + \mathcal{O}(g) \subseteq \mathcal{O}(f + g)$	
(8)	$\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f),$	**nur falls** $\mathcal{O}(g) \subseteq \mathcal{O}(f)$
(9)	$a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0 \in \mathcal{O}(n^k),$	$a_k > 0$

3. Aufgabe (12 Punkte): Programmverifikation

3.1. Formalisierung von Aussagen (4 Punkte) Formalisieren Sie die folgenden Aussagen (Hilfsprädikate sind mit zu definieren):

1. m ist die mittlere von drei ganzen Zahlen a, b, c.
2. Der char[] s ist in einem char[] str enthalten.

3.2. Partielle Korrektheit (8 Punkte) Im folgenden finden Sie eine Methode fibonacci, die einen Array der ersten n Fibonacci-Zahlen zurückgibt. Bestimmen Sie die für die POST-Condition notwendige PRE-Condition und beweisen Sie damit die Korrektheit der Methode (in der Vorgabe auf der folgenden Seite). Dabei können Sie die folgende Definition der Fibonacci-Zahlen benutzen:

$$\mathrm{fib}(n) = \begin{cases} 1; & n = 0 \\ 1; & n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2); & n \geq 2 \end{cases}$$

```
public static int [] fib (int n) {
    // PRE: ?
    int [] arr = new int[n];
    arr [0] = 1;
    arr [1] = 1;
    int i = 2;
    while (i < n) {
        arr[i] = arr[i-1] + arr[i-2];
        i = i + 1;
    }
    // POST: arr ∈ char^n ∧ arr_0 = 1 ∧ arr_1 = 1 ∧ ∀k ∈ {2,...n-1} : arr_k = arr_{k-1} + arr_{k-2}
    // POST: arr ∈ char^n ∧ ∀k ∈ {0,...n-1} : arr_k = fib(k)
    return arr;
}
```

```
public static int[] fib(int n) {
    // PRE:

    int[] arr = new int[n];

    arr[0] = 1;

    arr[1] = 1;

    int i = 2;

    while (i < n) {

        arr[i] = arr[i-1] + arr[i-2];

        i = i + 1;

    }

    // POST: arr ∈ char^n ∧ arr_0 = 1 ∧ arr_1 = 1 ∧ ∀k ∈ {2,...n−1} : arr_k = arr_{k−1} + arr_{k−2}
    // POST: arr ∈ char^n ∧ ∀k ∈ {0,...n−1} : arr_k = fib(k)
    return arr;
}
```

4. Aufgabe (5 Punkte): Assemblerprogrammierung

Hinweis: Gegeben ist die weiter hinten stehende VIP-Befehlstabelle.

4.1. (3 Punkte) Im Hauptspeicher befindet sich an den symbolischen Adressen `Zehner` und `Einer` je eine ASCII-codierte Dezimalziffer (jeweils ein 16-Bit-Wort).
Schreiben Sie ein vollständiges VIP-Assemblerprogramm, das die beiden Ziffern in gepackter Form, d.h. als zweistellige BCD-Zahl in der niederwertigen Worthälfte der Speicherzelle `BCD` ablegt. Beispiel:

	15 12	11 8	7 4	3 0
Zehner	0	0	3	4
Einer	0	0	3	7
BCD			4	7

```
        ORG   0
Zehner  DAT   '4'
Einer   DAT   '7'
BCD     RES   1
```

4.2. Adressierungsarten (2 Punkte) Gegeben ist das folgende VIP-Assemblerprogramm. Geben Sie die nach Ausführung des Programms in den Speicherzellen mit den Adressen 0 bis 3 befindlichen Inhalte an.

Hinweis: Beachten Sie, daß hier numerische Adressen verwendet sind und keine symbolischen!
Alle Zahlenangaben sind im Dezimalsystem.

```
ORG  4
LDX  #3
LDA  #2
STX  1
STA  @1
ADD  0[IX]
STA  0
SUBX 3
STX  1[IX]
HLT
END  4
```

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 := $\neg$ AC, 1-Komplement
NEG	03	x	x	x	x	-	-	-	-	AC := 0 - AC, 2-Komplement
ASL	04	x	x	AC_{15}	x	-	-	-	-	AC := AC $\times$ 2
ASR	05	x	x	AC_0	0	-	-	-	-	AC := AC $\times$ 2^{-1}, AC_{15} := AC_{15}
LSL	06	x	x	AC_{15}	0	-	-	-	-	AC := AC $\times$ 2
LSR	07	x	x	AC_0	0	-	-	-	-	AC := AC $\times$ 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 $\neq$ (Null)
BGT	19	-	-	-	-	✓	-	-	-	PC := Op, falls $>$
BGE	1A	-	-	-	-	✓	-	-	-	PC := Op, falls $\geq$
BLE	1B	-	-	-	-	✓	-	-	-	PC := Op, falls $\leq$
BLT	1C	-	-	-	-	✓	-	-	-	PC := Op, falls $<$
BGTU	1D	-	-	-	-	✓	-	-	-	PC := Op, falls $>$ (vorzeichenlos)
BGEU	13	-	-	-	-	✓	-	-	-	PC := Op, falls $\geq$ (vorzeichenlos)
BLEU	1E	-	-	-	-	✓	-	-	-	PC := Op, falls $\leq$ (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 $\wedge$ Op (bitpaarweise)
OR	80..83	x	x	0	0	✓	✓	✓	✓	AC := AC $\vee$ Op (bitpaarweise)
XOR	90..93	x	x	0	0	✓	✓	✓	✓	AC := AC $\oplus$ Op (bitpaarweise)

Tabelle 1: VIP-Befehlssatz-Referenz

5. Aufgabe (4 Punkte): Erweiterung des VIP

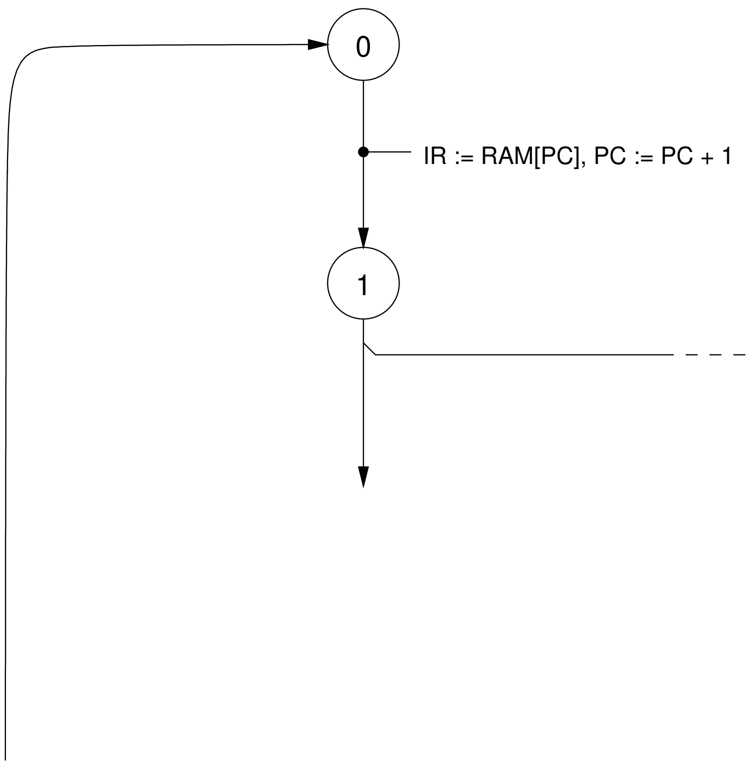
Der VIP soll um einen Befehl "**D**ecrement inde**X** and **B**ranch if **N**ot equal" `DBNX m` mit der Codierung `0x1F` erweitert werden. Er soll die gleiche Wirkung wie die nachstehende Befehlsfolge haben:

```
SUBX  #1
BNE   m
```

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

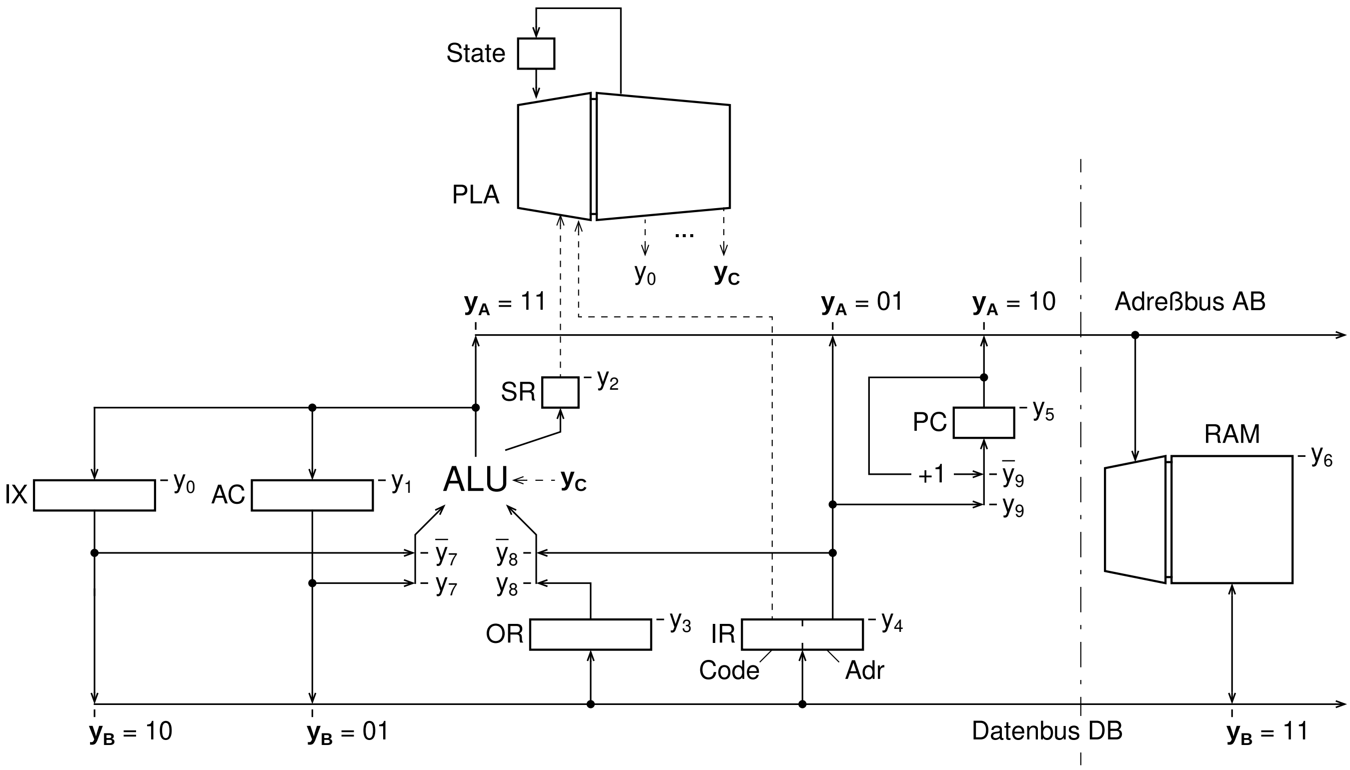
$s_{0\ldots4}$	ALU-Operation
0 0 1 1 0	$Z = X + u_0$
1 0 0 1 0	$Z = X + Y + u_0$
1 0 1 0 1	$Z = Y$
1 1 0 0 0	$Z = X - 1 + u_0$
1 1 1 1 0	$Z = X \cdot 2 + u_0$

5.1. (1 Punkte) Vervollständigen Sie den Zustandsgraphen für den "`DBNX`"-Befehl mit minimaler Anzahl von Zuständen (unter Ausnutzung parallel ausführbarer Operationen).



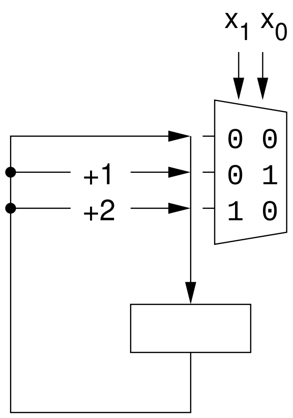
5.2. (3 Punkte) Füllen Sie die PLA-Tabelle gemäß des Zustandsgraphen der vorangehenden Teilaufgabe aus. Irrelevante Tabellenplätze dürfen freigelassen werden.

	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	$y_C = s_{0...4}, u_0$
0	-	-	-	-	-	1	0	0	0		1	1	0			0	10	11	

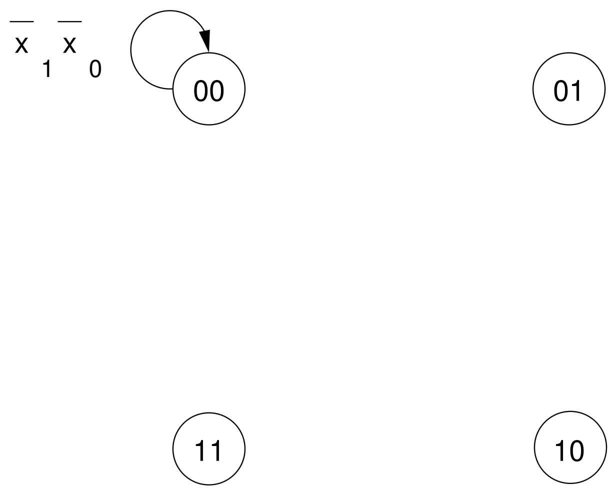


6. Aufgabe (8 Punkte): Schaltwerksentwurf

Es soll ein Modulo-4-Zähler mit zwei Flipflops u_1, u_0 entworfen werden, der in Abhängigkeit von zwei Eingangssignalen x_1, x_0 wie folgt zählt:



6.1. (2 Punkte) Vervollständigen Sie den Zustandsgraphen.

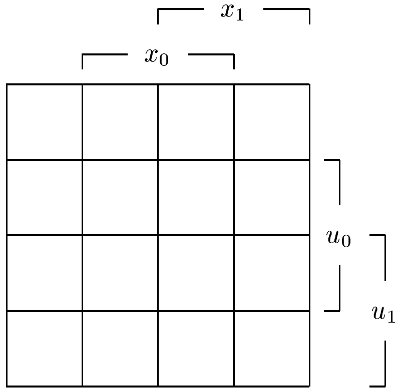


6.2. **(3 Punkte)** Der Zähler soll mit zwei unterschiedlichen Flipflop-Typen realisiert werden, einem JK-Flipflop u_1 mit den Eingängen j_1, k_1 und einem D-Flipflop u_0 mit dem Eingang d_0.
Ergänzen Sie die folgende Tabelle um die fehlenden Einträge.

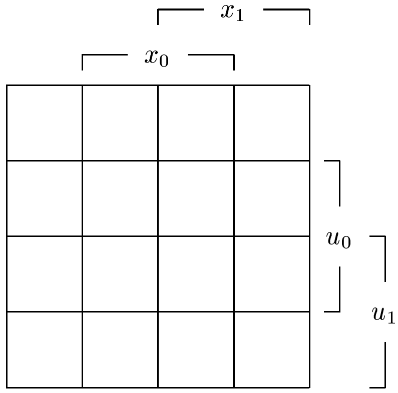
$v \to v^+$	j	k	d
$0 \to 0$	0	*	0
$0 \to 1$	1	*	1
$1 \to 0$	*	1	0
$1 \to 1$	*	0	1

u_1	u_0	x_1	x_0	u_1^+	u_0^+	j_1	k_1	d_0
0	0	0	0					
0	0	0	1					
0	0	1	0					
0	0	1	1	*	*	*	*	*
0	1	0	0					
0	1	0	1					
0	1	1	0					
0	1	1	1	*	*	*	*	*
1	0	0	0					
1	0	0	1					
1	0	1	0					
1	0	1	1	*	*	*	*	*
1	1	0	0					
1	1	0	1					
1	1	1	0					
1	1	1	1	*	*	*	*	*

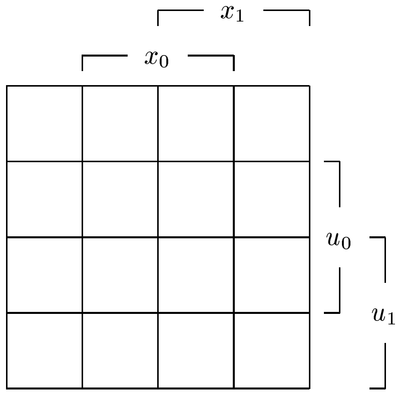
j_1:



k_1:



d_0:



6.3. **(2 Punkte)** Ermitteln Sie die minimalen Ansteuergleichungen der beiden Flipflops mit Hilfe der obigen KV-Tafeln.

6.4. (1 Punkte) Vervollständigen Sie das Blockbild zur Realisierung des Schaltwerks.