Klausur Informatik 2
13. Oktober 2001

Biedl / Flik
Schöner / 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	16	
2	10	
3	7	
4	5	
5	6	
6	6	

1. Aufgabe (16 Punkte): Algorithmen und Datenstrukturen

1.1. Module (2 Punkte) Welche Gründe kennen Sie, die für eine Modularisierung von Programmen sprechen, in Java z.B. das Zerlegen in mehrere Klassen, Quelldateien und Packages?

Lösung:

- *Code Sharing – Zwischen verschiedenen Projekten*
- *Strukturierung – Verbessert Übersicht, Verständlichkeit, Dokumentation des Codes*
- *Verteilte Codeentwicklung – Jede Datei ist einem Projektmitarbeiter zugeordnet*
- *Information Hiding – Von einem Modul ist nur die Schnittstelle bekannt, die Implementierung kann sich ändern*
- *Zugriffsrechte – Kapselung von package-internen Schnittstellen, Variablen, etc. (indem* `public`*, etc. weggelassen werden)*

1.2. Datenstrukturen (3 Punkte) Klassen, Arrays und (verkettete) Listen erlauben jeweils die Aggregation (das Zusammenfassen) von mehreren Datenelementen. Je nach Eigenschaften (Art, Anzahl wechselnd) der zu speichernden Datenelemente ist eine andere dieser Datenstrukturen besser geeignet. Unter welchen Bedingungen würden Sie sich jeweils für welche der genannten Datenstrukturen entscheiden?

Lösung:

- ***Klassen:*** *Eine feste Anzahl von heterogenen Datenelementen (unterschiedliche Datentypen) soll zu einer Einheit zusammengefasst werden.*
- ***Arrays:*** *Eine Anzahl homogener Datenelemente (vom gleichen Datentyp) soll zusammengefasst werden. Die (maximale) Anzahl ist entweder im Voraus bekannt, oder ändert sich zumindest selten. Wahlfreier Zugriff auf die einzelnen Datenelemente ist nötig (im Unterschied zu sequentiellem Zugriff). Eingefügt oder entfernt werden Datenelemente hauptsächlich am/vom Ende.*
- ***Listen:*** *Eine Anzahl homogener Datenelemente (vom gleichen Datentyp) soll zusammengefasst werden. Es ist nur sequentieller Zugriff auf die einzelnen Elemente nötig. Die Anzahl der Datenelemente muss nicht im voraus bekannt sein und kann sich während der Laufzeit stark verändern. Datenelemente können an jeder Stelle in der Liste eingefügt oder von dort entfernt werden.*

1.3. Stabilität (4 Punkte) Wenn ein Sortieralgorithmus *stabil* ist, bedeutet dies, dass an sich verschiedene Elemente, die aber durch das Sortierkriterium als gleich eingestuft werden, nach dem Sortieren untereinander die gleiche Reihenfolge haben, wie vor dem Sortieren. Relevant ist dies besonders, wenn z.B. Personen nur nach Nachnamen sortiert werden; Personen mit gleichem Nach-, aber unterschiedlichem Vornamen, sollen dann in der gleichen Reihenfolge wie vor dem Sortieren erscheinen.

Ist QuickSort stabil? Wie sieht es mit InsertionSort aus? Begründen Sie Ihre Antwort.

Lösung:

- ***QuickSort:*** *Nicht stabil.*
  Bei der Partionierung können Elemente über mehrere andere Elemente hinweg ausgetauscht werden, wobei unter diesen übersprungenen Elementen ein „gleiches“ sein kann. Selbst wenn dieses später auch noch ausgetauscht wird, kann es nicht noch einmal dem „gleichen“Element die Reihenfolge wechseln.
- ***InsertionSort:*** *Stabil.*
  Beim Einfügen des aktuellen Elements werden Elemente nur so lange in dem Array weiter nach hinten bewegt (und tauschen mit dem aktuellen Element die Reihenfolge), wie sie ungleich (sogar: größer) dem aktuellen Element sind. Gleiche Elemente tauschen also nie die Reihenfolge.

Vorgabe für 1.4 und 1.5:

Der Algorithmus von Euklid zur Bestimmung des ggT (größten gemeinsamen Teilers) zweier natürlicher Zahlen n, m funktioniert folgendermaßen (für n, m größer 0):

Wenn eine der beiden Zahlen 0 ist, ist die andere der ggT der beiden ursprünglichen Zahlen. Ansonsten subtrahiere die kleinere (oder gleiche) der beiden Zahlen von der größeren. Mit der kleineren der beiden ursprünglichen Zahlen und mit der eben gebildeten Differenz ist das Verfahren zu wiederholen.

1.4. Größter gemeinsamer Teiler (3 Punkte) Implementieren Sie diesen Algorithmus in rekursiver Form in einer Methode `int ggT(int n, int m)`. *Hinweis: Eine iterative Formulierung dieses Algorithmus' können Sie in Aufgabe 4.1 finden.*

Lösung:

```
static public int ggT(int n, int m) {
    if (m == 0)
        return n;
    if (n < m)
        return ggT(m, n);
    return ggT(m, n-m);
}
```

1.5. Teilerfremde Zahlen (4 Punkte) Implementieren Sie eine weitere Methode `teilerFremd`, unter Benutzung der Methode `ggT` aus der letzten Unteraufgabe. `teilerFremd` soll eine Tabelle berechnen, in der teilerfremde Zahlenpaare markiert sind. Teilerfremd sind Zahlen, wenn ihr GGT gleich 1 ist. Die Tabelle soll als zweidimensionaler Array von Elementen des Typs `boolean` zurückgegeben werden, wobei jedes Element genau dann ein `true` enthält, wenn die Zeilen- und die Spaltennummer (0-basiert, wie bei Java-Arrays) teilerfremd sind. Als Argumente erhält die Methode die Anzahl an Zeilen und Spalten (`n`), die diese Tabelle enthalten soll.

Lösung:

```
static public boolean[][] teilerFremd(int n) {
    boolean[][] arr = new boolean[n][n];
    for (int i=0; i<arr.length; ++i) {
        for (int j=0; j<arr[i].length; ++j)
            arr[i][j] = (ggT(i, j) == 1);
    }
    return arr;
}
```

2. Aufgabe (10 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:

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

Lösung:

- *$f(n) = 5n^{\frac{3}{2}}$, $g(n) = n\sqrt{n}$:*

$$\lim_{n\to\infty} \frac{5n^{\frac{3}{2}}}{n\sqrt{n}} = \lim_{n\to\infty} \frac{5n\sqrt{n}}{n\sqrt{n}} = \lim_{n\to\infty} \frac{5}{1} = 5$$

Der Grenzwert f/g ist größer als 0, aber kleiner als unendlich. Damit ist $f \in \Theta(g)$, und damit auch $f \in \mathcal{O}(g)$ und $f \in \Omega(g)$.

- *$f(n) = 2n^{\frac{1}{2}} - 4$, $g(n) = \ln n$:*

$$\lim_{n\to\infty} \frac{2n^{\frac{1}{2}} - 4}{\ln n} = \lim_{n\to\infty} \frac{2n^{\frac{1}{2}}}{\ln n} - \lim_{n\to\infty} \frac{4}{\ln n} \stackrel{\text{(de l'Hospital)}}{=} \lim_{n\to\infty} \frac{n^{-\frac{1}{2}}}{n^{-1}} - 0 = \lim_{n\to\infty} n^{\frac{1}{2}} \longrightarrow \infty$$

Der Grenzwert f/g divergiert. Damit ist $f \in \Omega(g)$, aber $f \notin \mathcal{O}(g)$ und $f \notin \Theta(g)$.

2.2. Aufwand von foobar (6 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, ...).

```
1 static public void foobar(int [] arr ) {
2     for ( int i =1; i<arr.length; ++i ) {
3         for ( int j =0; j<i; ++j ) {
4             if ( arr [ i-j-1] < arr[i-j]) {
5                 arr [ i-j-1] = arr[ i-j-1] + arr[ i-j];
6             }
7         }
8     }
9 }
```

Lösung:

$$A_{\textsf{foobar}}(n) = \underbrace{3}_{2} + \sum_{i=1}^{n-1} \Big(\underbrace{3+2}_{2,3} + \sum_{j=0}^{i-1} \big(\underbrace{2+6}_{3,4} + q \cdot \underbrace{10}_{5} \big) \Big)$$

q ist eine (unbekannte) Variable $\in [0,1]$, die angibt, wie häufig der if-Zweig der Fallunterscheidung ausgeführt wird. Da wir darüber keine Informationen haben, nehmen wir im folgenden an (Worst-Case): $q = 1$.

$$\begin{aligned}
&= 3 + \sum_{i=1}^{n-1} \Big(5 + \sum_{j=0}^{i-1} (18) \Big) = 3 + \sum_{i=1}^{n-1} \Big(5 + \big((i-1) - 0 + 1 \big) \cdot 18 \Big) \\
&= 3 + \sum_{i=1}^{n-1} 5 + \sum_{i=1}^{n-1} 18i = 3 + 5(n-1) + 18 \sum_{i=1}^{n-1} i = 5n - 2 + 18 \frac{(n-1)n}{2} \\
&= 9n^2 - 4n - 2
\end{aligned}$$

Regel 9, denn $k = 2$ ($a_2 = 9 > 0$):

$$A_{\textsf{foobar}}(n) \in \mathcal{O}(n^2)$$

3. Aufgabe (7 Punkte): Programmverifikation

3.1. Verstärkungs- / Abschwächungsregel (2 Punkte) Erklären Sie die Bedeutung der Verstärkungs- / Abschwächungsregel des Hoare-Kalküls für Korrektheitsbeweise von Programmen. Wann wird sie verwendet, und weshalb ist sie nötig?

Lösung:

Sie erlaubt es, Vorbedingungen einer Anweisung(-sfolge) zu verstärken (schärfer zu formulieren, z.B. eine Konjunktion zu ergänzen oder eine Disjunktion zu streichen), bzw. eine Nachbedingung abzuschwächen. Damit ist es möglich, die Nachbedingung einer Anweisung an die Vorbedingung der nächsten anzupassen, z.B. im Rahmen der Sequenzregel. Oft sind diese nicht gleich; durch Verwendung der V/A-Regel reicht es nun zu zeigen, dass die Vorbedingung der zweiten Anweisung eine Implikation der Nachbedingung der ersten Anweisung ist, um die Beweiskette zu schließen. Auch bei der Angleichung der POST-Conditions des if- und des else-Zweigs bei der Regel der Fallunterscheidung ist die Regel oft nötig.

3.2. Partielle Korrektheit (5 Punkte) Beweisen Sie die partielle Korrektheit der folgenden Methode bezüglich der gegebenen POST-Condition, und leiten Sie die notwendige PRE-Condition her. Verwenden Sie dafür die Regeln des Hoare-Kalküls. Auf der folgenden Seite befindet sich ein Vordruck zum Eintragen der Aussagen.

```
static public int prodArr(int [] arr) {
    int res = 1;
    int i = 0;
    while (i < arr.length) {
        res = res * arr[i];
        i = i + 1;
    }
    // POST: res = ∏_{k=0}^{arr.length−1} arr[i]
    return res;
}
```

Lösung:

```
static public int prodArr(int [] arr) {
    // PRE: n = arr.length
    // (1 = 1) ∧ (0 ≤ n)
    int res = 1;
    // (res = 1) ∧ (0 ≤ n)
    // (res = ∏_{k=0}^{0−1} arr[k]) ∧ (0 ≤ n)
    int i = 0;
    // I : (res = ∏_{k=0}^{i−1} arr[k]) ∧ (i ≤ n)
    while (i < arr.length) {
        // I ∧ (i < n)
        // (res ∗ arr[i] = arr[i] ∗ ∏_{k=0}^{i−1} arr[k]) ∧ (i < n)
        // (res ∗ arr[i] = ∏_{k=0}^{i} arr[k]) ∧ (i < n)
        res = res ∗ arr[i];
        // (res = ∏_{k=0}^{i} arr[k]) ∧ (i < n)
        // (res = ∏_{k=0}^{(i+1)−1} arr[k]) ∧ ((i + 1) ≤ n)
        i = i + 1;
        // I
    }
    // I ∧ ¬(i < n)
    // (res = ∏_{k=0}^{i−1} arr[k]) ∧ (i ≤ n) ∧ (i ≥ n)
    // (res = ∏_{k=0}^{i−1} arr[k]) ∧ (i = n)
    // POST: res = ∏_{k=0}^{arr.length−1} arr[k]
    return res;
}
```

4. Aufgabe (5 Punkte): Assemblerprogrammierung

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

4.1. (3 Punkte) Gegeben ist die folgende JAVA-Methode zur Berechnung des größten gemeinsamen Teilers (GGT) zweier positiver Zahlen.

```
int ggt (int x, int y) {
  while (x != y) {
    if (x < y)
      y = y - x;
    else
      x = x - y;
  }
  return x;
}
```

Schreiben Sie ein entsprechendes VIP-Assemblerprogramm, indem Sie die folgende Vorgabe vervollständigen. Es wird empfohlen, das Programm mit Kommentaren zu versehen!

```
            ORG 0
x           RES 1
y           RES 1
ggt         RES 1
start       LDA x
while       CMP y
            BEQ equal
            BLT less
```

Lösung:

```
      ORG 0
x     RES 1
y     RES 1
ggt   RES 1
start LDA x       ; x wird im AC gehalten
while CMP y       ; ein Vergeich für while- und if-Bedingung
      BEQ equal   ; Abbruchbedingung x==y
      BLT less    ; Sprung zum if-Zweig
      SUB y       ; else-Zweig: x = x - y
      JMP while
less  STA x       ; if-Zweig: AC=x sichern
      LDA y       ; \
      SUB x       ;  y = y - x
      STA y       ; /
      LDA x       ; AC=x wiederherstellen
      JMP while
equal STA ggt     ; Ergebnis speichern
      HLT
      END start
```

4.2. (2 Punkte) Gegeben sind die folgenden VIP-Assemblerprogramme. Ermitteln Sie die Inhalte der Register AC und IX nach deren Ausführung.

```
      ORG 0
A     RES 1
B     DAT 3,4
C     EQU 2
start LDX #C
      LDA A[IX]
      HLT
      END start
```

AC = IX =

```
      ORG 0
A     RES 1
B     DAT 2,3
C     EQU 1
start LDA C
      LDX @B
      HLT
      END start
```

AC = IX =

Lösung:

AC = 4, IX = 2
AC = 2, IX = 3

Befehle ohne Operanden:

Operation	Code (hex)	SR-Inhalt				Adressierung				Wirkung
		z	n	c	v		#	@	[IX]	
HLT	00	-	-	-	-	-	-	-	-	Anhalten
NOP	01	-	-	-	-	-	-	-	-	Nichts
NOT	02	x	x	-	-	-	-	-	-	AC := ¬ AC, 1-Komplement
NEG	03	x	x	x	x	-	-	-	-	AC := 0 - AC, 2-Komplement
ASL	04	x	x	x	x	-	-	-	-	AC := AC $\times$ 2, c := Bit 15
ASR	05	x	x	x	-	-	-	-	-	AC := AC $\times$ 2^{-1}, c := Bit 0
LSL	06	x	x	x	-	-	-	-	-	AC := AC $\times$ 2, c := Bit 15
LSR	07	x	x	x	-	-	-	-	-	AC := AC $\times$ 2^{-1}, c := Bit 0, Bit 15 := 0

Befehle mit Operanden:

Operation	Code (hex)	SR-Inhalt				Adressierung				Wirkung
		z	n	c	v		#	@	[IX]	
JMP	10	-	-	-	-	✓	-	-	-	PC := Op
BPL	11	-	-	-	-	✓	-	-	-	PC := Op, falls positiv
BMI	12	-	-	-	-	✓	-	-	-	PC := Op, falls negativ
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 nicht Null ($\neq$)
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 + (0..3)	x	x	0	0	✓	✓	✓	✓	AC := Op
LDX	28 + (0..3)	x	x	0	0	✓	✓	✓	✓	IX := Op
STA	30 + (0..3)	-	-	-	-	✓	-	✓	✓	Op := AC
STX	38 + (0..3)	-	-	-	-	✓	-	✓	✓	Op := IX
ADD	40 + (0..3)	x	x	x	x	✓	✓	✓	✓	AC := AC + Op
ADDX	48 + (0..3)	x	x	x	x	✓	✓	✓	✓	IX := IX + Op
SUB	50 + (0..3)	x	x	x	x	✓	✓	✓	✓	AC := AC - Op
SUBX	58 + (0..3)	x	x	x	x	✓	✓	✓	✓	IX := IX - Op
CMP	60 + (0..3)	x	x	x	x	✓	✓	✓	✓	AC - Op
CMPX	68 + (0..3)	x	x	x	x	✓	✓	✓	✓	IX - Op
AND	70 + (0..3)	x	x	-	-	✓	✓	✓	✓	AC := AC $\wedge$ Op (bitpaarweise)
OR	80 + (0..3)	x	x	-	-	✓	✓	✓	✓	AC := AC $\vee$ Op (bitpaarweise)
XOR	90 + (0..3)	x	x	-	-	✓	✓	✓	✓	AC := AC $\oplus$ Op (bitpaarweise)

Tabelle 1: VIP-Befehlssatzreferenz

5. Aufgabe (6 Punkte): Erweiterung des VIP

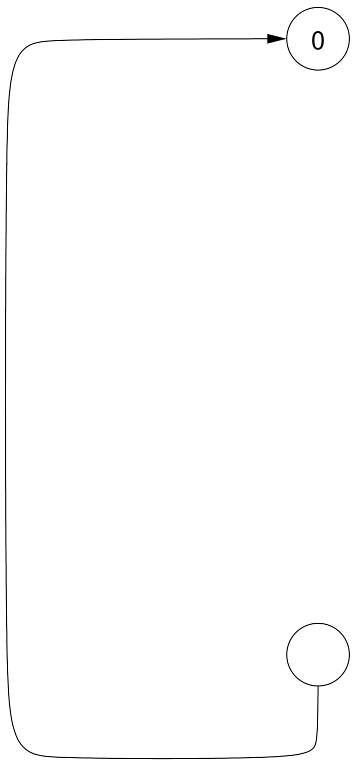
Der VIP soll um einen Swap-Befehl `SWP M` erweitert werden, der den Inhalt des Akkumulators AC mit dem einer Speicherzelle M vertauscht. Das Statusregister wird davon nicht beeinflußt.

5.1. (1 Punkte) Beschreiben Sie die Abarbeitung des Befehls mit Registerzuweisungen.

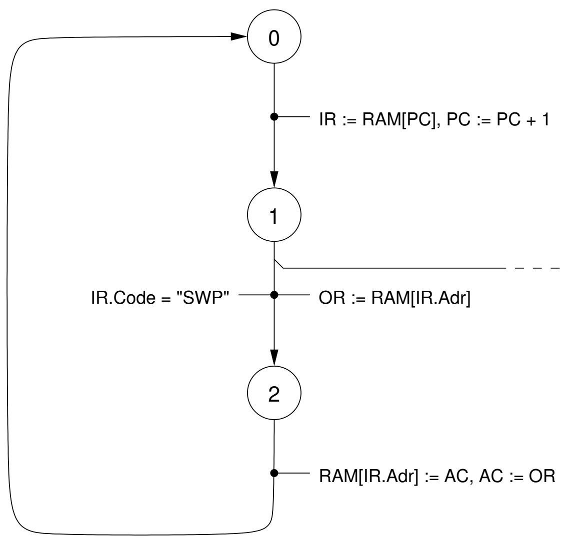
Lösung:

```
IR := RAM[PC], PC := PC+1
OR := RAM[IR.Adr]
RAM[IR.Adr] := AC, AC := OR
```

5.2. (2 Punkte) Vervollständigen Sie den Zustandsgraphen für den `SWP`-Befehl mit minimaler Anzahl von Zuständen (unter Ausnutzung parallel ausführbarer Operationen).



Lösung:

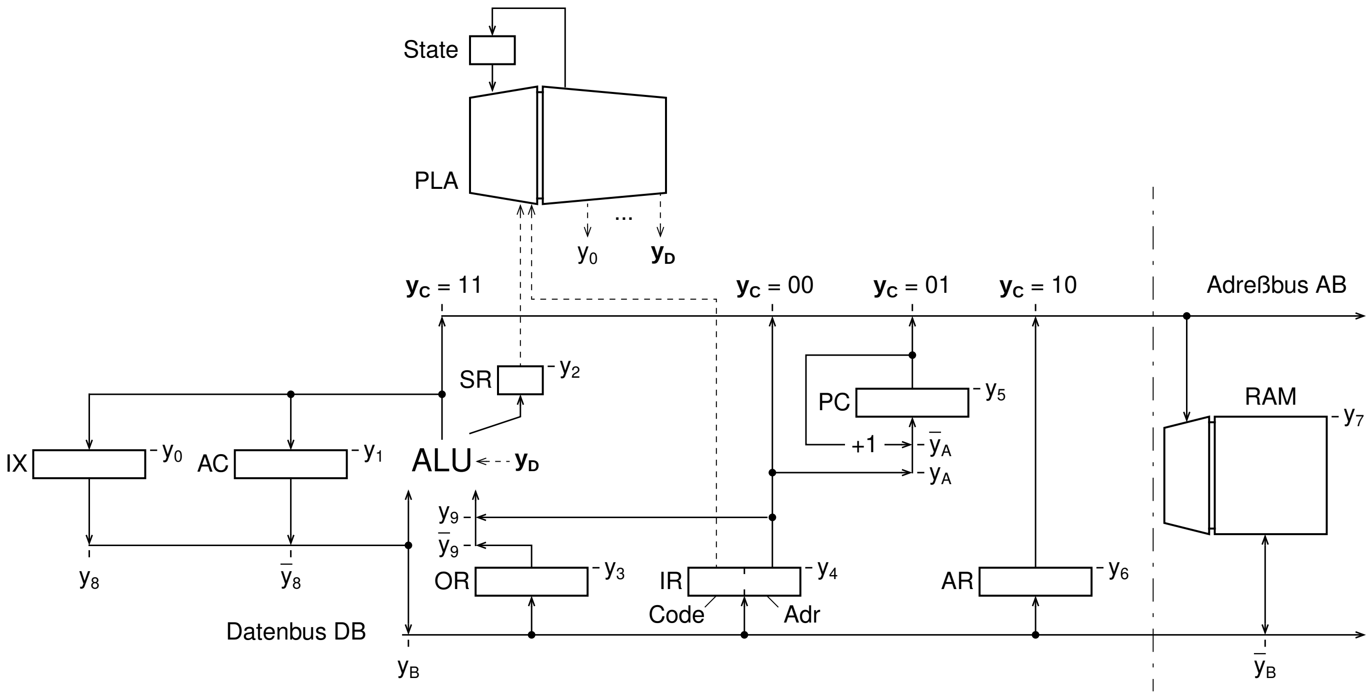


5.3. (3 Punkte) Füllen Sie die PLA-Tabelle aus. Irrelevante Werte können freigelassen werden.

	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	y_D
0	-	-	-	-	-	1														
1	-	-	-	-	SWP	2														

Lösung:

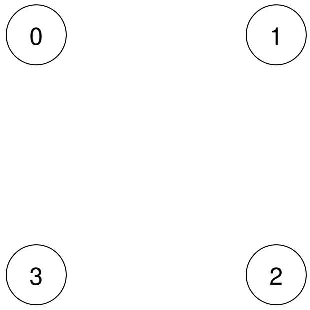
	SR				*IR*		*Register/Speicher*								*Multiplexer*					*ALU*
Z_i	z	n	c	v	*Code*	Z_j	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_A	y_B	y_C	y_D
0	-	-	-	-	-	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*			*0*	*0*	*01*	
1	-	-	-	-	*SWP*	*2*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*				*0*	*00*	
2	-	-	-	-	-	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*		*1*	*00*	*001100*



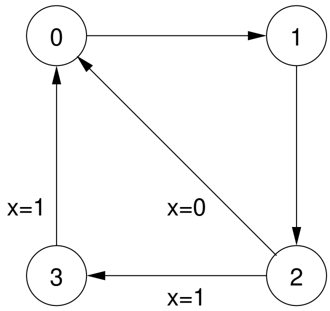
6. Aufgabe (6 Punkte): Schaltwerksentwurf

Ein synchroner 2-Bit-Dualzähler soll in Abhängigkeit von einer Eingangsleitung x die Zustandsfolge 0, 1, 2 (bei x=0) bzw. 0, 1, 2, 3 (bei x=1) zyklisch durchlaufen. Der Übergang vom Zustand 3 aus ist bei x=0 undefiniert.

6.1. (1 Punkte) Vervollständigen Sie den Zustandsgraphen.



Lösung:



6.2. (2 Punkte) Ergänzen Sie die Tabelle für die Übergangs- und Ansteuerfunktionen für eine Realisierung mit SR-Flipflops.

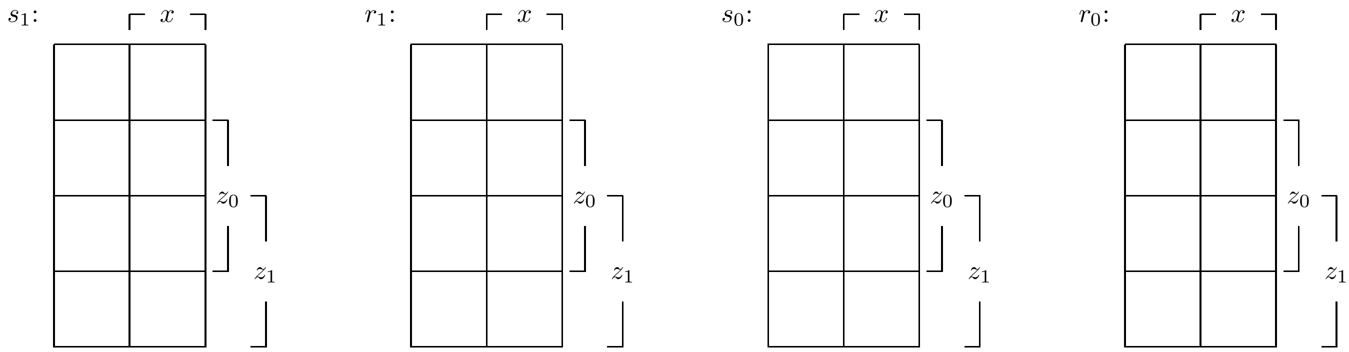
v → v^+	s	r
0 → 0	0	*
0 → 1	1	0
1 → 0	0	1
1 → 1	*	0

z_1	z_0	x	z_1^+	z_0^+	s_1	r_1	s_0	r_0
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

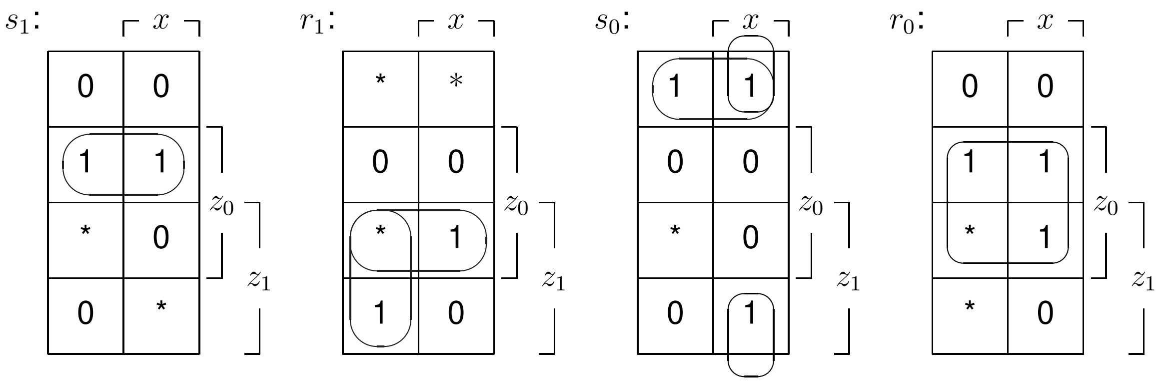
Lösung:

z_1	z_0	x	z_1^+	z_0^+	s_1	r_1	s_0	r_0
0	0	0	0	1	0	*	1	0
0	0	1	0	1	0	*	1	0
0	1	0	1	0	1	0	0	1
0	1	1	1	0	1	0	0	1
1	0	0	0	0	0	1	0	*
1	0	1	1	1	*	0	1	0
1	1	0	*	*	*	*	*	*
1	1	1	0	0	0	1	0	1

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

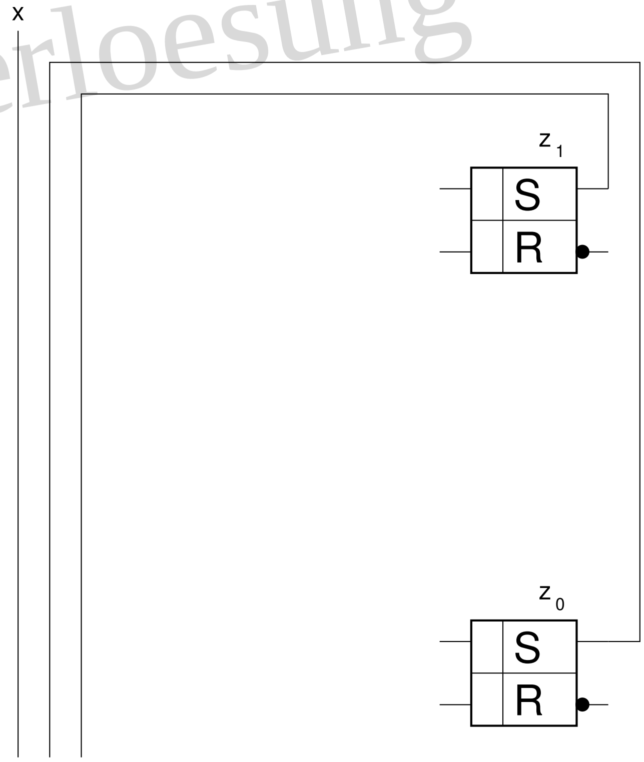


Lösung:



$$
\begin{aligned}
s_1 &= z_0 \cdot \overline{z_1} \\
r_1 &= z_0 \cdot z_1 + z_1 \cdot \overline{x} \\
s_0 &= \overline{z_0} \cdot \overline{z_1} + \overline{z_0} \cdot x \\
r_0 &= z_0
\end{aligned}
$$

6.4. (1 Punkte) Vervollständigen Sie das Blockbild.



Lösung: